Journal of Artificial Intelligence Research 78 (2023) 69-141 Submitted 02/2023; published 09/2023

Sequence-Oriented Diagnosis of Discrete-Event Systems

Gianfranco Lamperti GIANFRANCO.LAMPERTI@UNIBS.IT
Stefano Trerotola S.TREROTOLA@STUDENTI.UNIBS.IT
Marina Zanella MARINA.ZANELLA@UNIBS.IT
Department of Information Engineering, University of Brescia
Via Branze 38, 25123 Brescia, Italy

Xiangfu Zhao XIANGFUZHAO@GMAIL.COM
School of Computer and Control Engineering, Yantai University
30, Qingquan RD, Laishan District, Yantai 264005, China

Abstract

Model-based diagnosis has always been conceived as *set-oriented*, meaning that a candidate is a *set* of faults, or faulty components, that explains a collection of observations. This perspective applies equally to both static and dynamical systems. Diagnosis of discrete-event systems (DESs) is no exception: a candidate is traditionally a set of faults, or faulty events, occurring in a trajectory of the DES that conforms with a given sequence of observations. As such, a candidate does not embed any temporal relationship among faults, nor does it account for multiple occurrences of the same fault. To improve diagnostic explanation and support decision making, a *sequence-oriented* perspective to diagnosis of DESs is presented, where a candidate is a *sequence* of faults occurring in a trajectory of the DES, called a *fault sequence*. Since a fault sequence is possibly unbounded, as the same fault may occur an unlimited number of times in the trajectory, the set of (output) candidates may be unbounded also, which contrasts with set-oriented diagnosis, where the set of candidates is bounded by the powerset of the domain of faults. Still, a possibly unbounded set of fault sequences is shown to be a regular language, which can be defined by a regular expression over the domain of faults, a property that makes sequence-oriented diagnosis feasible in practice. The task of *monitoring-based* diagnosis is considered, where a new candidate set is generated at the occurrence of each observation. The approach is based on three different techniques: (1) *blind* diagnosis, with *no* compiled knowledge, (2) *greedy* diagnosis, with *total* knowledge compilation, and (3) *lazy* diagnosis, with *partial* knowledge compilation. By *knowledge* we mean a data structure slightly similar to a classical DES diagnoser, which can be generated (compiled) either entirely offline (greedy diagnosis) or incrementally online (lazy diagnosis). Experimental evidence suggests that, among these techniques, only lazy diagnosis may be viable in non-trivial application domains.

1. Introduction

The purpose of a diagnosis task is to explain the behavior of a (natural or artificial) system or process based on a collection of observations. Since the system/process is almost invariably only partially observable, different explanations can be formulated for the same set of observations. In medical diagnosis, for instance, the observed behavior consists of a pool of recorded symptoms, while the explanation consists of the possible causes occurred inside a human body that can justify these symptoms. The diagnosis task has always been (and still is) a challenge to AI. The first attempts to face this challenge date back to the middle 1970s, when diagnostic rule-based systems were developed, such as *MYCIN* (Shortliffe, 1976). A decade later, *rule-based* diagnosis was replaced by *model-based* diagnosis (Reiter, 1987; de Kleer & Williams, 1987; Hamscher, Console, & de Kleer,

1992), which exploits the model of the inner behavior of the system in order to find the causes of its observed behavior. A system model is seldom *monolithic*, however. In fact, it is usually *distributed* (or *compositional*), consisting of several *components*, where the model of the inner behavior of each component is defined explicitly. What remains unspecified is the behavior of the system as a whole, which, at least in theory, can be automatically inferred based on the component models and the mode in which components are connected to one another.

The alternative explanations of a collection of observations are called *candidates*; this is why the diagnosis output is in general a *set* of candidates. When the system is static, each component model is represented as a time-independent mapping from the input(s) to the output(s). Since, as a whole, a static system, such as a combinational circuit, has some input and output pins, the diagnosis task processes the observed behavior consisting of a set of the values for all the system inputs and outputs.

In a dynamical system, time comes into play, and model-based diagnosis needs to account for the state of the system also (Struss, 1997). At a certain abstraction level, time tags can be ignored, while retaining only the chronological order of state changes. Hence, a dynamical system may be conveniently modeled as a *discrete-event system* (DES), where the state changes over qualitative time (Cassandras & Lafortune, 2008). The observed behavior of a DES, called a *temporal observation*, consists of a temporally ordered sequence of events, called *observations*, that have been perceived while the system was being operated.

Depending on whether a temporal observation is given altogether, or several chunks are progressively provided so as to follow the system while being operated, the diagnosis task is either *a posteriori* or *monitoring-based*, respectively. A posteriori diagnosis is meant to find out what has happened to the system in the time interval in which the temporal observation was generated. This task may be performed long after the temporal observation is recorded, typically when the system came to a halt, unexpectedly or even catastrophically, and the relevant causes need to be uncovered based on a collection of records. Monitoring-based diagnosis, instead, is performed while the system is being operated, possibly in an abnormal way, with sensor readings being sampled and periodically sent to a control unit so that the relevant set of candidates may be updated continuously. The output of monitoring-based diagnosis may be exploited in order to make decisions about the feedback to be provided to the system as well as the maintenance/repair actions to perform.

This paper presents some techniques for monitoring-based diagnosis of a class of distributed DESs, called *active systems* (AS) in the literature (Lamperti, Zanella, & Zhao, 2018). Each candidate produced as an output every time a new observation chunk has been processed is not a *set* of faults, however, instead it is a *fault sequence*, namely the sequence of faults occurred in a sequence of state transitions that generates the sequence of observations perceived so far. We say that a *sequence-oriented* (rather than a *set-oriented*) perspective is adopted here. The notion of a fault sequence was first introduced by Bertoglio et. al. (2020c, 2020a, 2020b, 2020d), called a *temporal fault* there, where relevant diagnosis engines were proposed for both a posteriori (2020a, 2020d) and monitoring-based diagnosis (2020b, 2020c). The present paper integrates and extends these initial works by focusing on monitoring-based diagnosis of fault sequences in distributed DESs, with the following main contributions:

- It systematizes the task of monitoring-based diagnosis in a coherent reference framework based on three different techniques, namely (1) without compiled knowledge, called *blind* diagnosis, (2) by exploiting total knowledge compilation, called *greedy* diagnosis, and (3) by exploiting only partial knowledge compilation, called *lazy* diagnosis. Adopting a sequence-

oriented perspective in blind diagnosis is totally new (no previous work by the authors is concerned with it). As to greedy and lazy diagnoses, this paper is the first to provide the pseudocode of an algorithm that computes a fault space (cf. Definition 8), by extending the *state elimination* technique presented by Brzozowski and McCluskey (1963), which performs the conversion from a finite automaton (FA) to an equivalent regular expression; almost all subsequent conversion methods can be recast as its variants (Sakarovitch, 2009).

- It illustrates the setup and the outcomes of an extensive experimental activity, by showing that both blind and greedy diagnoses are bound to be impractical, even for DESs of a few components: only lazy diagnosis may be viable in real application domains. No experimental results about sequence-oriented diagnosis of DESs can be found in previous works, let alone a comparative empirical analysis as illustrated in this paper.

In the remainder of the paper, Section 2 provides a conceptual contextualization of the work, and describes the motivations behind it. Section 3 recalls the notion of a (distributed) DES, and defines some major concepts. Sections 4 through 6 deal with monitoring-based diagnosis of fault sequences in DESs, presenting the approach to the task in the three variants: blind, greedy, and lazy diagnosis, respectively. Section 7 presents a sample application for sequence-oriented diagnosis of DESs. Section 8 outlines the implementation of a relevant software system (available online), and shows the experimental results. Section 9 positions the current contributions with respect to previous papers by the authors and a classical seminal approach in the literature. Related work is analyzed in Section 10. Conclusions are drawn in Section 11. The Appendix presents some additional hints about the asymptotic time complexity of the algorithms.

2. Context and Motivation

Artificial reasoning in model-based diagnosis can be either *consistency-based* or *abduction-based*. In consistency-based diagnosis each candidate is consistent with the observed behavior, while in abduction-based diagnosis each candidate entails the observed behavior. *Strong fault models*, which specify both the normal and abnormal inner behavior of the system (components), can be adopted for both forms of explanation. *Weak fault models*, which specify only the normal inner behavior of the system (components), can instead be used just for consistency-based diagnosis.

In static systems, when the fault models are weak, a candidate is a (possibly empty) set of *faulty* components, namely components that do not operate in the normal way, whereas, when the models are strong, a candidate specifies the (possibly *faulty*) behavioral mode of each component. A faulty mode describes the behavior of a component when a specific *fault* is present, hence faults are part of the component modeling. An individual fault is the possible cause of a specific misbehavior in a system component, and, generally speaking, several faults are bound to cause the misbehavior of the system as a whole. The set of faults that can affect the system components is assumed to be finite.

Considering DESs, the way a DES is represented is varying: either as a monolithic finite-state machine (Lunze, 2000), as a Petri net (Benveniste, Fabre, Haar, & Jard, 2003; Jiroveanu, Boel, & Bordbar, 2008; Cabasino, Giua, & Seatzu, 2010; Basile, 2014; Yin & Lafortune, 2017; Cong, Fanti, Mangini, & Li, 2018; Ran, Su, Giua, & Seatzu, 2018; Li, Khlif-Bouassida, & Toguyéni, 2019), or as a network of components, each of them being modeled either as a synchronous FA (Sampath, Sengupta, Lafortune, Sinnamohideen, & Teneketzis, 1995, 1996; Sampath, Lafortune, &

Teneketzis, 1998), or as an asynchronous FA, like in the current paper. Being untimed, these models do not explicitly consider any time length, with the only temporal information being relevant to the reciprocal order of the state transitions. Although a notable exception can be recorded (Pencolé, Steinbauer, Mühlbacher, & Travé-Massuyès, 2018), DESs usually adopt strong fault models, where each component transition is either *normal* or *faulty*, as in the seminal work by (Sampath et al., 1995, 1996, 1998), in the AS approach (Lamperti et al., 2018), as well as in this paper, and a specific fault is assigned to each faulty transition, the same fault being possibly shared by several transitions.

As introduced above, the solution of a DES diagnosis problem is a set of candidates. A candidate has to *explain* a temporal observation, where the notion of an explanation depends on the abstraction level considered. At the lowest level, a candidate is a *trajectory*, namely a sequence of (component) state transitions, that produces the given temporal observation. However, when solving a diagnosis problem, we are not usually interested in all events within a trajectory, but rather in the faulty events only. Hence, candidates (and consequently diagnosis) can be provided at different (higher) abstraction levels, as discussed by Grastien et al. (2011). At a level above the trajectory, a candidate is the *sequence* of faults relevant to a trajectory, as in the current paper. At a further higher abstraction level, a candidate is the *multiset* of faults relevant to a trajectory, as in the work by Jiang et.al. (2003) and further works about DES diagnosability, including the one by Yoo and Garcia (2009). At a still higher abstraction level, a candidate is the *set* of faults relevant to a trajectory, as in the diagnoser approach (Sampath et al., 1995, 1996), in the traditional AS approach (Lamperti et al., 2018), and in most works about DES diagnosis. At the highest abstraction level, a candidate is just a tag, either *normal* or *abnormal*, corresponding to a trajectory. The tag is *abnormal* if the relevant trajectory includes some faulty events, otherwise it is *normal*. Notably, at this abstraction level we are interested just in fault detection, not in fault isolation/identification, which is traditionally the core of a diagnosis task.

In a set-oriented perspective, which, as remarked, is the level of abstraction that is usually adopted, a DES candidate is a *set* of faults occurring in a trajectory of the DES that produces the given temporal observation. Since the domain of faults is finite, both each single candidate and each diagnosis output (whole set of candidates) are finite and bounded. Diagnosing a DES becomes a sort of *abductive* reasoning (McIlraith, 1998), inasmuch the candidates are generated based on the trajectories of the DES (sequences of state transitions) that entail a given temporal observation.

While appropriate for static systems, set-orientation makes diagnosis of DESs narrow in explainability. After all, it sounds odd that a candidate is a set, considering that a candidate is somewhat the projection of a trajectory, which is not a set, but a *sequence* of state transitions. The point is, since a set is a collection of unordered elements without duplicates, both the temporal ordering of faults and their multiple occurrences are completely lost within a set, whereas in the real world faults occur sequentially, possibly within a causal chain, and are not necessarily permanent. When the diagnosis task focuses on critical systems or processes, such as a large power network, a nuclear plant, or even the evolution of a pandemic like COVID-19, set-oriented diagnosis may be less than optimal in explaining a supposedly abnormal behavior.

One may argue that, since in set-oriented monitoring-based diagnosis a new set of candidates is output upon reception of a new observation chunk, it is possible to ascertain whether some additional faults have possibly occurred or not. However, one cannot ascertain whether a fault occurred previously has occurred again, in other words, no clear piece of information about the possible *intermittency* of the pinpointed faults is provided by set-oriented diagnosis. In fact, the inclusion of a

fault in a candidate means that, according to the considered candidate, the fault has occurred at least once.

A fault sequence differs from a classical candidate mainly in three ways: (1) a fault sequence includes the *multiset* of faults occurred in the trajectory, where all the occurrences of the same fault are encompassed, (2) in a fault sequence, the relative temporal order of the occurrences of each fault is clearly shown, that is, a total temporal order of faults is provided, and (3) the length of a fault sequence may be unbounded. Therefore, the diagnosis output is the (possibly infinite) set of all distinct fault sequences, each relevant to a (possibly infinite) set of trajectories of the DES that produce a given temporal observation.

Fortunately, the (possibly infinite) set of candidates that explain a temporal observation turns out to be a regular language over the alphabet of faults, thus it can be represented as a regular expression. In other words, each fault sequence is a string matching a regular expression defining the diagnosis output of a DES. The additional temporal information embedded in fault sequences may be essential for ranking purposes, as well as for helping diagnosticians and/or troubleshooting algorithms make decisions in critical scenarios. In this sense, fault sequences support the explainability of the diagnosis output, precisely because they do not merely provide information on the faults occurred in the DES (like in set-oriented diagnosis), but because these faults are placed within a temporal sequence that makes crystal clear what happened to the DES (assuming that candidate). These pieces of information could help also in taking some sequential diagnosis steps (de Kleer & Williams, 1987; Feldman, Provan, & van Gemund, 2010b).

The topic of explainability is broad in AI and has drawn considerable attention in the last few years to the extent that the expression *explainable AI* (XAI) has been coined. The aim of the XAI research area is to provide more transparency to the algorithms used in AI, so as to increase the users' trust in intelligent agents. More specifically, according to Miller (2019), *explainability* refers to the processes in which one of the criteria taken into account during the computation is the degree to which a human can understand the outputs in the given context, whereas the term *explanation* refers to the way the outputs are explicitly explained to humans. Research about explanations is multifaceted: explanations can be defined, generated, selected, presented, and evaluated. More precisely, an *explanation function* can be defined, where this function generates an explanation given the original problem instance and the output produced by an intelligent agent as the solution of that instance. About the evaluation of explanations, this is aimed at assessing the interpretability of explanations themselves, which is an issue (Marques-Silva & Ignatiev, 2022).

Our paper does not deal with XAI. The term *explanation*, which is quite overloaded, has been used since the advent of the theory of model-based diagnosis (Reiter, 1987), and even before it (Reggia, Nau, & Wang, 1983, 1985): each candidate diagnosis is somehow an 'explanation' in that it justifies what has been observed by hypothesizing what has possibly happened. In XAI, an explanation is an output produced by a software system in order to justify and make more trustworthy the output produced by another software system. In other words, in XAI both the system whose outputs have to be explained and the system that generates the explanations are software systems. In model-based diagnosis, an explanation is instead an output produced by a diagnostic software system in order to justify the output (i.e., the observation) produced by a system whose model is given (and it has been processed by the diagnostic system itself). In other words, in model-based diagnosis the system whose outputs have to be explained is not necessarily a software application. More specifically, in model-based diagnosis of DESs, the system whose outputs have to be explained has been modeled as a DES, and this model is exploited by the diagnosis engine in order to produce the

explanations; hence, the DES model can be considered as an *explanation model*. This is in line with Lundberg and Lee (2017), according to which, in order to explain a complex system, we must use an explanation model that is simpler than it. This is exactly what is performed in model-based diagnosis: in order to explain the observed behavior of a complex (either physical or artificial) system, we use a simpler model of the system itself (e.g. a DES model).

The need for explanations that are both sound and not redundant is the motivation behind the emerging discipline of *formal XAI* (Marques-Silva & Ignatiev, 2022), which focuses on explanation approaches that offer formal guarantees of rigor. Formal rigor is reckoned the only viable alternative for computing explanations in high-risk and safety critical settings. In the same paper, the notion of an *abductive explanation* is introduced as a (subset-minimal) set of inputs that, if these inputs are assigned the values in the considered instance, then the set of these input values entails the output to be explained, regardless of the values assigned to the remaining inputs. Notice that there may exist several distinct abductive explanations for the same pair (input values, output), like in abductive diagnosis there may be several candidates relevant to the same diagnosis problem instance.

Traditionally, in model-based diagnosis a candidate consists in a *set* of a faults; we can ascertain that the hypothesized set of faults entails the given observations only if the system inputs that have generated the observed outputs are known, that is, if the given observation includes both the inputs and the outputs, as is usually the case with diagnosis of static systems. Unfortunately, for DESs, the given observation does not include (all) the input values (which encompass a whole time interval), as (some) inputs are unobservable. Hence, in abduction-based diagnosis of DESs, the only candidate that can entail the given observation is a trajectory that moves the DES from the initial state to other states within a discrete space, as a trajectory implicitly hypothesizes not only the faults but also the system input values over time. However, abduction-based diagnosis of DESs does not output the (possibly unbounded) set of trajectories that entail the given sequence of observable events, namely a temporal observation, instead traditional approaches output the collection of the sets of faults relevant to these trajectories while a sequence-oriented approach outputs all the sequences of faults relevant to these trajectories. In other words, a candidate is the set (or sequence) of faults occurring in a trajectory that entails the given temporal observation. Once a candidate has been computed, we can be sure that there are some trajectories, affected by such set (or sequence) of faults, that produce the given temporal observation, and one of these trajectories has necessarily been followed by the system; however, there may be some other trajectories, affected by the same set (or sequence) of faults, that do not produce the given temporal observation.

The whole set of sequence-oriented candidates relevant to a temporal observation is a regular expression over the alphabet of faults. A regular expression is endowed with a content that has a richer significance and usefulness with respect to a collection of sets of faults. Assume, for instance, that, given a DES diagnosis problem, the set-oriented output and the sequence-oriented output are $\{\{a, b\}, \{b, c\}\}$ and $b(aaa \mid c)b$, respectively. According to set-oriented diagnosis, the DES is affected by a pair of faults, either $\{a, b\}$ or $\{b, c\}$. A diagnostician may wonder whether fault b, which has certainly occurred since it belongs to both candidates, was the first to occur or not, these two cases having quite different meaning and severity. The sequence-oriented diagnosis, in addition to the piece of information about the two alternative sets of faults, lets the diagnostician know that fault b was the first to occur (so now they know whether the situation is serious or not), that fault b has necessarily repeated itself, and that, in between the two occurrences of b, either fault a has occurred three times or fault c has occurred once. An expert can draw some useful hints both from the repetition of b, e.g. for it rules out some evolutions, and from the faults in between, e.g. the

diagnostician knows that it is easy enough to verify whether sequence bcb has taken place, hence they will check it. This check will enable the diagnostician to discard one alternative, thus singling out the real candidate, whereas the process for identifying the real candidate will be more difficult and lengthy if only the set-oriented diagnosis output is available.

Since the output of sequence-oriented diagnosis includes additional pieces of information with respect to the output of set-oriented diagnosis, two questions raise quite naturally: *how can the sequence-oriented task be performed, and which is its performance?* This paper addresses the former question in three ways, and shows some experimental evidence to answer the latter.

In the paragraphs above we have mentioned only explicit DES models, as an explicit (operational) model (a network of communicating FAs) is adopted in this paper. However, in the literature some contributions about diagnostic reasoning ignore any explicit DES models, instead, they consider some specifications. The specification of a dynamical system (i.e., the properties the system has to exhibit over time) can be given as a formula in a temporal logic (Emerson, 1990), such as Linear Temporal Logic (LTL) (Pnueli, 1977). An LTL specification is the implicit representation of an automaton (Courcoubetis, Vardi, Wolper, & Yannakakis, 1991; Kesten, Manna, McGuire, & Pnueli, 1993; Gerth, Peled, Vardi, & Wolper, 1996; Daniele, Giunchiglia, & Vardi, 1999), where LTL operators describe the state transitions. Typical diagnostic tasks are aimed at finding out whether a given behavioral evolution (called a *trace*) satisfies the specification formula (Pill & Quaritsch, 2013; Pill & Wotawa, 2018) and/or uncovering the causes for a trace violates the specification formula, where such causes can be searched for either in the trace (Beer, Ben-David, Chockler, Orni, & Trefler, 2009) or in the specification (Pill & Quaritsch, 2013; Pill & Wotawa, 2018), that is, the specification may be wrong. These and other related works are considered in Section 10.

3. Preliminaries

Let $\mathcal{X}$ denote a (distributed) DES, namely a network of components where the behavior of each component is modeled as a communicating automaton (Brand & Zafiropulo, 1983). A component is equipped with input and output pins, where each output (input) pin is connected with an input (output) pin of another component by a link. A link is a communication channel in which events can be sent from one component to another. A component transition can be triggered either by an external event coming from the outside of $\mathcal{X}$ or by an internal event coming from another component in $\mathcal{X}$. When an event occurs, a component may react by performing a transition based on the model of its communicating automaton. When performing a transition, a component consumes the triggering (input) event and possibly generates new events on its output pins, which, owing to links, are placed on input pins of other components, thereby possibly triggering the transitions of other components. A transition generating an event on an output pin can occur only if this pin is not occupied by another event.

Example 1 (*DES*) Depicted on the left of Figure 1 is a physical device that is designed to control a cooling system. The device involves a transducer,[1] incorporating a temperature sensor, and a valve. When the temperature becomes high, the transducer commands the valve to open in order to let the cooling fluid flow. Vice versa, when the temperature returns to normal, the transducer commands the valve to close. The device is modeled as the DES shown in the center of the figure, called $\mathcal{P}$

1. A transducer is a device that converts variations in a physical quantity, such as pressure or temperature, into an electrical signal, or vice versa.

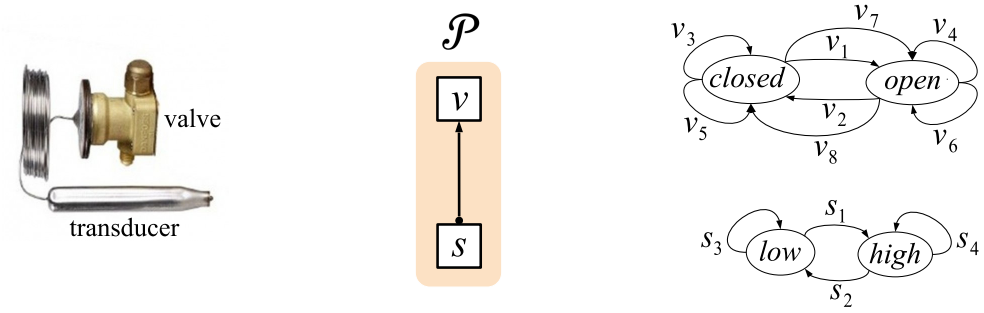


Figure 1: From left to right: protection physical system; DES $\mathcal{P}$ composed of a transducer s, a valve v, and a link from s to v; and communicating automata of v (top) and s (bottom).

Component transition	*Description*
$v_1 = \langle closed, (op, \emptyset), open \rangle$	The valve reacts to the open event by opening
$v_2 = \langle open, (cl, \emptyset), closed \rangle$	The valve reacts to the close event by closing
$v_3 = \langle closed, (op, \emptyset), closed \rangle$	The valve does not react to the open event and remains closed
$v_4 = \langle open, (cl, \emptyset), open \rangle$	The valve does not react to the close event and remains open
$v_5 = \langle closed, (cl, \emptyset), closed \rangle$	The valve reacts to the close event by remaining closed
$v_6 = \langle open, (op, \emptyset), open \rangle$	The valve reacts to the open event by remaining open
$v_7 = \langle closed, (cl, \emptyset), open \rangle$	The valve reacts to the close event by opening
$v_8 = \langle open, (op, \emptyset), closed \rangle$	The valve reacts to the open event by closing
$s_1 = \langle low, (ko, \{op\}), high \rangle$	On high temperature (ko), the transducer generates the open event
$s_2 = \langle high, (ok, \{cl\}), low \rangle$	On low temperature (ok), the transducer generates the close event
$s_3 = \langle low, (ko, \{cl\}, low \rangle$	On high temperature (ko), the transducer generates the close event
$s_4 = \langle high, (ok, \{op\}), high \rangle$	On low temperature (ok), the transducer generates the open event

Table 1: Details of transitions of the components v and s (cf. communicating automata in Figure 1).

(protection), which includes two components, a transducer s and a valve v, and one link connecting the (single) output pin of s with the (single) input pin of v. Outlined on the right of the figure are the models (communicating automata) of the two components. The model of s (on the bottom) involves two states (denoted by circles) and four transitions (denoted by arcs). The model of v (on the top) involves two states and eight transitions. Each component transition from a state x to a state x' that is triggered by an input event e and generates a set E of output events, denoted by a triple $\langle x, (e, E), x' \rangle$, is detailed in Table 1.[2]

2. The specification of component transitions in Table 1 may be questionable in a real setting: they are defined this way here for the sake of simplicity.

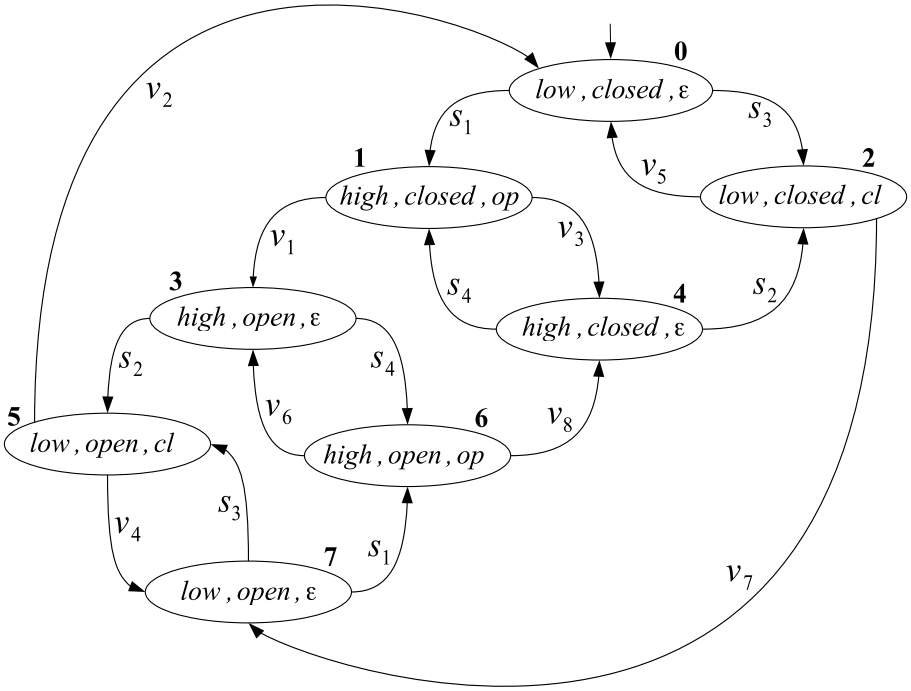


Figure 2: Space of the DES $\mathcal{P}$ (cf. Figure 1), with initial state $0 = (low, closed, \varepsilon)$.

Assuming that only one component transition at a time can occur (asynchronism), the process that moves a DES $\mathcal{X}$ from its initial state can be represented as a sequence of component transitions, called a *trajectory* of $\mathcal{X}$. At the occurrence of a component transition in a trajectory, $\mathcal{X}$ changes state. A contiguous subsequence of a trajectory is called a *trajectory segment*. The (possibly infinite) set of trajectories of $\mathcal{X}$, which can be generated based on the components and links of $\mathcal{X}$, can be represented as a deterministic FA (DFA), called the *space* of $\mathcal{X}$.

Definition 1 (Space (Lamperti et al., 2018)**)** *The* space *of a DES $\mathcal{X}$ is a DFA,*

$$Space(\mathcal{X}) = (\Sigma, X, \tau, x_0) \tag{1}$$

where Σ (the alphabet) is the set of component transitions, X is the set of states, where a state is a pair (C, L), with C being the array of the states of the components and L being the array of the (possibly empty[3]) events within links, τ is the deterministic transition function[4] mapping a state and a component transition into a new state, $\tau : X \times \Sigma \mapsto X$, and x_0 is the initial state.

Example 2 (*Space*) The space of $\mathcal{P}$, namely $Space(\mathcal{P})$, is depicted in Figure 2, where each state is identified by a triple $(\bar{s}, \bar{v}, e)$, with $\bar{s}$ being a state of the transducer, $\bar{v}$ a state of the valve, and e an event within the link (ε indicates no event). To simplify referencing, the space states are renamed

3. Formally, an empty link contains an empty event, denoted ε.
4. The (behavior of the) DES is not assumed to be deterministic, even if the DES space is formally deterministic, because the alphabet is the set of component transitions, whose identifiers are unique. In fact, the nondeterminism of the DES behavior stems from several transitions, exiting the same state, that are triggered by the same event, for example, transitions s_1 and s_3 of the transducer, which are both triggered by the *ko* event (cf. Figure 1 and Table 1).

t	o	f
s_1	*sns*	ε
s_2	*sns*	ε
s_3	ε	α
s_4	ε	β
v_1	*vlv*	ε
v_2	*vlv*	ε
v_3	ε	γ
v_4	ε	δ
v_5	*vlv*	ε
v_6	*vlv*	ε
v_7	*vlv*	λ
v_8	*vlv*	σ

o	*Observation description*
sns	The transducer performs a normal action
vlv	The valve reacts (possibly abnormally) to an event

f	*Fault description*
α	The transducer generates the *cl* event instead of *op*
β	The transducer generates the *op* event instead of *cl*
γ	The valve remains closed upon the *op* event
δ	The valve remains open upon the *cl* event
λ	The valve opens upon the *cl* event
σ	The valve closes upon the *op* event

Figure 3: Mapping table $Map(\mathcal{P})$ (left), and details of the observations and faults involved (right).

$0 \cdots 7$, where 0 is the initial state. Owing to cycles, the set of possible trajectories of $\mathcal{P}$ is infinite, one of them being $T = [s_3, v_5, s_1, v_3, s_4, v_3, s_2]$, which ends in state $2 = (low, closed, cl)$.

For diagnosis purposes, the specification of a DES needs to be enriched with information about the observability (what is observable and what is not) and the abnormality (what is normal and what is faulty) of the DES. Both observability and abnormality are specified in a *mapping table*.

Definition 2 (Mapping Table (Bertoglio, Lamperti, & Zanella, 2019a)) *Let* $\mathbf{T}$ *be the set of component transitions in a DES* $\mathcal{X}$*, let* $\mathbf{O}$ *be a finite set of* observations*, and let* $\mathbf{F}$ *be a finite set of* faults*. The* mapping table *of* $\mathcal{X}$ *is a function*

$$Map(\mathcal{X}) : \mathbf{T} \mapsto (\mathbf{O} \cup \{\varepsilon\}) \times (\mathbf{F} \cup \{\varepsilon\}) \tag{2}$$

where ε *is the* empty *symbol.*

The table $Map(\mathcal{X})$ is represented as a finite set of triples (t, o, f), where $t \in \mathbf{T}$, $o \in \mathbf{O} \cup \{\varepsilon\}$, and $f \in \mathbf{F} \cup \{\varepsilon\}$. A triple (t, o, f) defines the observability and normality of t. Specifically, if $o \neq \varepsilon$, then t is *observable*, else t is *unobservable*; likewise, if $f \neq \varepsilon$, then t is *faulty*, else t is *normal*.

Example 3 (*Mapping Table*) The mapping table of the DES $\mathcal{P}$ (cf. Example 1), namely $Map(\mathcal{P})$, is shown on the left of Figure 3, where $\mathbf{O} = \{sns, vlv\}$ and $\mathbf{F} = \{\alpha, \beta, \gamma, \delta, \lambda, \sigma\}$, which are described on the right of the figure. Only one observation is provided for both the transducer and the valve, namely *sns* and *vlv*, respectively, each being associated with several (still not all) transitions. For instance, transition s_1 is observable and normal, s_3 is unobservable and faulty, while v_7 is both observable and faulty. Owing to the association of the same observation with several transitions, uncertainty remains in determining the actual transition based solely on the observation perceived.

Based on a mapping table $Map(\mathcal{X})$, each trajectory of $\mathcal{X}$ can be associated with a *temporal observation* and a *fault sequence*.

Definition 3 (Temporal Observation (Bertoglio et al., 2020b)**)** *Let T be a trajectory of a DES $\mathcal{X}$. The* temporal observation *of T is the sequence of the observations involved in T, namely*

$$Obs(T) = [\, o \mid t \in T, (t, o, f) \in Map(\mathcal{X}), o \neq \varepsilon \,]\,. \tag{3}$$

If $\mathcal{O} = Obs(T)$, then the trajectory T is said to conform *with the temporal observation $\mathcal{O}$. The set of temporal observations relevant to all the trajectories in $Space(\mathcal{X})$ is the* observation language *of $Space(\mathcal{X})$, namely*

$$OBS(Space(\mathcal{X})) = \{Obs(T) \mid T \text{ is a trajectory of } \mathcal{X}\}\ . \tag{4}$$

Example 4 (*Temporal Observation*) According to the mapping table $Map(\mathcal{P})$ displayed in Figure 3 and considering the trajectory $T = [s_3, v_5, s_1, v_3, s_4, v_3, s_2]$ of $\mathcal{P}$ introduced in Example 2, we have $Obs(T) = [vlv, sns, sns]$, where the observations correspond to the occurrence of the observable transitions v_5, s_1, ans s_2, respectively.

In the literature, a trajectory T is also associated with a diagnosis, namely the set of faults involved in T. In that set-oriented perspective, a diagnosis does not indicate the time precedences between faults, nor does it account for multiple occurrences of the same fault. On the other hand, treating a diagnosis as a set of faults guarantees that the domain of possible diagnoses is finite, being bounded by the powerset of the domain of faults. In contrast with this classical perspective, we introduce a more informative notion of diagnosis for DESs, called a *fault sequence*.

Definition 4 (Fault Sequence (Bertoglio et al., 2020b)**)** *Let T be a trajectory of a DES $\mathcal{X}$. The* fault sequence *of T is the sequence of the faults involved in T,*

$$Flt(T) = [\, f \mid t \in T, (t, o, f) \in Map(\mathcal{X}), f \neq \varepsilon \,]\,. \tag{5}$$

A contiguous subsequence of a fault sequence is a fault-sequence segment.

Example 5 (*Fault Sequence*) According to the mapping table $Map(\mathcal{P})$ in Figure 3 and considering the trajectory $T = [s_3, v_5, s_1, v_3, s_4, v_3, s_2]$ of $\mathcal{P}$ in Example 2, we have $Flt(T) = [\alpha, \gamma, \beta, \gamma]$, where the faults correspond to the occurrence of the faulty transitions s_3, v_3, s_4, and v_3, respectively.

Notice that, since the length of T is in general unbounded, the length of both $Obs(T)$ and $Flt(T)$ is in general unbounded also (yet finite).

Given a temporal observation $\mathcal{O}$ of a DES $\mathcal{X}$, a diagnosis task can be applied in order to find possible fault sequences of $\mathcal{X}$. Roughly, this requires determining the trajectories of $\mathcal{X}$ that conform with $\mathcal{O}$, thereby allowing for the generation of the relevant fault sequences.

Definition 5 (Candidate Set (Bertoglio et al., 2020a)**)** *Let $\mathcal{O}$ be a temporal observation of a DES $\mathcal{X}$. The* candidate set *of $\mathcal{O}$, denoted $\Delta(\mathcal{O})$, is the set of fault sequences relevant to the trajectories that conform with $\mathcal{O}$,*

$$\Delta(\mathcal{O}) = \{\, Flt(T) \mid T \text{ is a trajectory of } \mathcal{X}, Obs(T) = \mathcal{O} \,\}\,. \tag{6}$$

Example 6 (*Candidate Set*) Let $\mathcal{O} = [vlv, sns, sns]$ be the temporal observation of $\mathcal{P}$ considered in Example 4. According to Definition 5, the computation of $\Delta(\mathcal{O})$ requires determining the set $\mathbb{T}$ of trajectories of $\mathcal{P}$ such that $Obs(T) = \mathcal{O}$. Based on the space of $\mathcal{P}$ in Figure 2 and the mapping table $Map(\mathcal{P})$ in Figure 3, the set $\mathbb{T}$ can be specified as a regular expression over component transitions[5], namely

$$\mathbb{T} = s_3 v_5 s_1 v_3 (s_4 v_3)^* s_2$$

which includes $T = [s_3, v_5, s_1, v_3, s_4, v_3, s_2]$ (cf. Example 4). Then, $\Delta(\mathcal{O})$ can be derived from $\mathbb{T}$ by replacing the faulty component transitions with the corresponding faults in $Map(\mathcal{P})$ and eventually removing the empty faults ε, namely

$$\Delta(\mathcal{O}) = \alpha\varepsilon\varepsilon\gamma\,(\beta\gamma)^*\,\varepsilon = \alpha\gamma\,(\beta\gamma)^*\ .$$

In plain language, based on the description of faults in Figure 3, the candidate set of $\mathcal{O}$ includes the fault sequences where: (α) the transducer commands the valve to close rather than to open when the temperature becomes high; (γ) the valve keeps being closed when commanded to open by the transducer; (($\beta\gamma$)*) zero or more repetitions of the pair of faults β (the transducer commands the valve to open rather than to close when the temperature becomes low) and γ (the valve keeps being closed when commanded to open by the transducer).

Likewise, with $\mathcal{O}' = [sns, vlv]$ we have $\mathbb{T}' = s_1\,(v_3 s_4)^*\,v_1\,s_4?$. Hence, $\Delta(\mathcal{O}') = (\gamma\beta)^*\,\beta?$. In other words: zero or more repetitions of the pair of faults γ (the valve keeps being closed when commanded to open by the transducer) and β (the transducer commands the valve to open rather than to close when the temperature becomes low), possibly followed by a further occurrence of β. Notice how $\Delta(\mathcal{O}')$ includes the empty fault-sequence also (no faults).

When monitoring a DES, a new candidate set is generated upon reception of each new observation, which accounts for all the trajectories that conform with the temporal observation generated so far. The sequence of these candidate sets is called the *explanation* of the temporal observation.

Definition 6 (Explanation) *Let $\mathcal{O} = [o_1, \ldots, o_n]$ be a temporal observation of a DES $\mathcal{X}$. The* explanation *of $\mathcal{O}$, denoted $Expl(\mathcal{O})$, is a sequence of candidate sets of the prefixes of $\mathcal{O}$, namely*

$$Expl(\mathcal{O}) = [\Delta_0, \Delta_1, \ldots, \Delta_n] \tag{7}$$

where each Δ_i, $i \in [0\,..\,n]$, is the candidate set of the prefix $\mathcal{O}_i = [o_1, \ldots, o_i]$, with $\mathcal{O}_0 = [\,]$.

Example 7 (*Explanation*) Let $\mathcal{O} = [vlv, sns, sns]$ be a temporal observation of the DES $\mathcal{P}$ (cf. Example 4). Based on the space of $\mathcal{P}$ in Figure 2 and $Map(\mathcal{P})$ in Figure 3, the sequence of regular expressions defining the sets of trajectories associated with the prefixes of $\mathcal{O}$, namely $\mathcal{O}_0 = [\,], \mathcal{O}_1 = [vlv], \mathcal{O}_2 = [vlv, sns]$, and $\mathcal{O}_3 = [vlv, sns, sns]$, is $[\mathbb{T}_0, \mathbb{T}_1, \mathbb{T}_2, \mathbb{T}_3]$, where $\mathbb{T}_0 = s_3?$, $\mathbb{T}_1 = s_3\,(v_5 s_3? \,|\, v_7((s_3 v_4)^* \,|\, s_3(v_4 s_3)^*))$, $\mathbb{T}_2 = s_3\,((v_5 s_1((v_3 s_4)^* \,|\, v_3(s_4 v_3)^*)) \,|\, v_7(s_3 v_4)^* s_1)$, and $\mathbb{T}_3 = s_3 v_5 s_1 v_3 (s_4 v_3)^* s_2$. Hence, according to Definition 6, we have:

$$\begin{aligned} Expl(\mathcal{O}) &= [\Delta_0, \Delta_1, \Delta_2, \Delta_3] = \\ &= \left[\alpha?,\ \ \alpha\left(\alpha? \,|\, \lambda\left((\alpha\delta)^* \,|\, \alpha(\delta\alpha)^*\right)\right),\ \ \alpha\left((\gamma\beta)^* \,|\, \gamma(\beta\gamma)^* \,|\, \lambda(\alpha\delta)^*\right),\ \ \alpha\gamma(\beta\gamma)^*\right]. \end{aligned}$$

5. A regular expression can be defined inductively over an alphabet Σ of symbols as follows. The empty symbol ε is a regular expression. If $a \in \Sigma$, then a is a regular expression. If x and y are regular expressions, then the followings are regular expressions: $x \mid y$ (alternative), xy (concatenation), $x?$ (optionality), x^* (zero or more repetitions), and x^+ (one or more repetitions). When parentheses are missing, the concatenation has precedence over the alternative, while repetition has the highest precedence; for example, $ab^* \mid c$ equates to $(a(b)^*) \mid c$.

One may ask what is the relation between two consecutive candidate sets in an explanation. Proposition 1 provides an answer to that question.

Proposition 1 *Let $\mathcal{O} = [o_1, \ldots, o_n]$ be a temporal observation and let $Expl(\mathcal{O}) = [\Delta_0, \ldots, \Delta_n]$. Each fault sequence in Δ_i, $i \in [1\,..\,n]$, is a (possibly empty) extension of a fault sequence in Δ_{i-1}.*

This property is grounded on the fact that a new observation o_i may make some trajectories relevant to the fault sequences in Δ_{i-1}, which were consistent with the prefix $\mathcal{O}_{i-1} = [o_1, \ldots, o_{i-1}]$, no longer consistent with the prefix $\mathcal{O}_i = [o_1, \ldots, o_i]$. Besides, the suffix of the trajectories that keep being consistent with $\mathcal{O}_i$ may involve some faults that extend the corresponding fault sequence in Δ_i.

As pointed out, the explanation of a temporal observation $\mathcal{O}$ is expected to be generated incrementally, by appending a new candidate set after the reception of a new observation. That *monitoring-based* diagnosis of fault sequences is implemented by three alternative techniques:

1. *Blind* diagnosis, without knowledge compilation: the diagnosis engine only relies on the model of the DES, without any additional data structure generated (compiled) offline.

2. *Greedy* diagnosis, with total knowledge compilation: knowledge compilation is carried out offline for generating data structures based on the model of the DES, which are meant to speed up the online diagnosis engine, as in the diagnoser approach (Sampath et al., 1995, 1996).

3. *Lazy* diagnosis, with partial knowledge compilation: initial partial knowledge is compiled offline and possibly extended online by the diagnosis engine when necessary (like a partial diagnoser being extended on-demand).

These diagnosis techniques are the subject of the next three sections, respectively.

4. Blind Diagnosis

When the diagnosis of fault sequences is carried out without compiled knowledge, the diagnosis engine is required to generate the explanation based on the description of the DES only. Consequently, the diagnosis engine needs to construct the portion of the space of a DES $\mathcal{X}$ that conforms with a temporal observation $\mathcal{O}$, called the *$\mathcal{O}$-constrained space* of $\mathcal{X}$.

Definition 7 ($\mathcal{O}$-Constrained Space (Lamperti, Zanella, & Zhao, 2020)**)** *Let $\mathcal{O}$ be a temporal observation of $\mathcal{X}$. The* $\mathcal{O}$-constrained space *of $\mathcal{X}$, denoted $Cspace(\mathcal{X}, \mathcal{O})$, is a DFA whose language equals the set of the trajectories T of $\mathcal{X}$ where $Obs(T) = \mathcal{O}$.*

Operationally, $Cspace(\mathcal{X}, \mathcal{O})$ is generated based on the component models and the links in $\mathcal{X}$, in a way similar to the (possible) construction of $Space(\mathcal{X})$. Specifically, a state of $Cspace(\mathcal{X}, \mathcal{O})$ is a pair (x, i), where x is a state of $\mathcal{X}$ and i is an *index* in the range $[0\,..\,n]$, where n is the number of observations in $\mathcal{O}$. Starting from the initial state $(x_0, 0)$, the transition function of $Cspace(\mathcal{X}, \mathcal{O})$ is constructed by taking into account the component transitions that are triggerable in the state considered. When an observable transition t is triggerable in a state (x, i), with x' being the new state of $\mathcal{X}$ reached by t, a transition $\langle (x, i), t, (x', (i + 1)\rangle$ is created in $Cspace(\mathcal{X}, \mathcal{O})$ iff $(t, o_{i+1}, f) \in Map(\mathcal{X})$, $i < n$, where o_{i+1} is the $(i + 1)$-th observation in $\mathcal{O}$. If, instead, t is

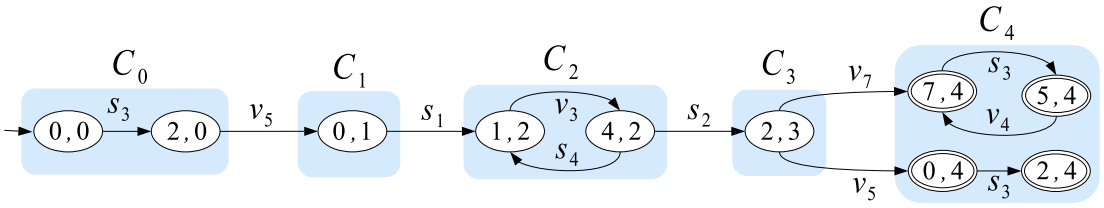


Figure 4: $Cspace(\mathcal{P}, \mathcal{O})$, where $\mathcal{O} = [vlv, sns, sns, vlv]$.

unobservable, the index i keeps unchanged. When $i = n$, that is, when all the observations in $\mathcal{O}$ are matched, the state (x, i) is final.

Consequently, the topology of $Cspace(\mathcal{X}, \mathcal{O})$ is stratified into $n + 1$ strata, called *clusters*, namely $C_0, C_1, \ldots, C_n$, where each cluster C_i, $i \in [0\,..\,n]$, only includes the states that are associated with the same index i, namely (x, i). Besides, each cluster C_i, $i \in [0\,..\,(n-1)]$, is exited by one or several observable transitions, relevant to observation o_{i+1}, that enter the next cluster C_{i+1}, which in turn involves the states $(x, i+1)$. Hence, even if cycles of unobservable transitions may occur within a cluster, owing to the monotonic increment of the observation index, clusters are connected to one another linearly, from C_0 to C_n.

Example 8 ($\mathcal{O}$*-Constrained Space*) Consider the temporal observation $\mathcal{O} = [vlv, sns, sns, vlv]$ for the DES $\mathcal{P}$. The $\mathcal{O}$-constrained space of $\mathcal{O}$, namely $Cspace(\mathcal{P}, \mathcal{O})$, is displayed in Figure 4. It is composed of the clusters $C_0 \ldots C_4$, where each cluster C_i, $i \in [0\,..\,3]$, is connected with the successive cluster C_{i+1} by arcs that are marked with observable transitions, all of them being relevant to the observation o_{i+1} of $\mathcal{O}$. For instance, the observable transitions marking the arcs exiting C_3 are v_5 and v_7, both of them generating the fourth observation *vlv*. It is easy to check that the language[6] of $Cspace(\mathcal{P}, \mathcal{O})$ equals the set of trajectories in $Space(\mathcal{P})$ that conform with $\mathcal{O}$.

Based on eqn. (6) and Definition 7, since a trajectory T in $Cspace(\mathcal{X}, \mathcal{O})$ is such that T is in $Space(\mathcal{X})$ and $Obs(T) = \mathcal{O}$, we have $Flt(T) \in Cand(\mathcal{O})$. In fact, $Cand(\mathcal{O})$ is exactly the set of fault sequences relevant to the set of trajectories in $Cspace(\mathcal{X}, \mathcal{O})$. Still, this approach is impractical as it requires the consideration of a possibly infinite set of trajectories in order to generate $Cand(\mathcal{O})$. So, what to do in order to compute $Cand(\mathcal{O})$ based on Definition 5? Fortunately, a formal property of $Cand(\mathcal{O})$ claimed in Proposition 2 is key to overcoming this supposedly computational obstacle.

Proposition 2 *Let $\mathcal{O}$ be a temporal observation of a DES $\mathcal{X}$. The candidate set $\Delta(\mathcal{O})$ is a regular language on the set of faults of $\mathcal{X}$ involved in the mapping table $Map(\mathcal{X})$.*

Proof. Let $Cspace(\mathcal{X}, \mathcal{O})$ be the $\mathcal{O}$-constrained space of $\mathcal{X}$. Thus, the set of trajectories in $Cspace(\mathcal{X}, \mathcal{O})$ equals the set of trajectories T in $Space(\mathcal{X})$ such that $Obs(T) = \mathcal{O}$. Let $\mathcal{N}$ be the nondeterministic FA (NFA) obtained from $Cspace(\mathcal{X}, \mathcal{O})$ by substituting f for each component transition t marking an arc in $Cspace(\mathcal{X}, \mathcal{O})$, where $(t, o, f) \in Map(\mathcal{X})$. The set of strings marking a path from the initial state to a final state in $\mathcal{N}$ is the candidate set $\Delta(\mathcal{O})$. Since it is accepted by an NFA, this language is a regular language. □

6. Roughly, the (regular) language of an FA is the set of strings on its alphabet that can be generated by traversing the automaton from the initial state to a final state. Since the alphabet of $Cspace(\mathcal{P}, \mathcal{O})$ is the whole set of the component transitions, each such string is a trajectory of $\mathcal{P}$.

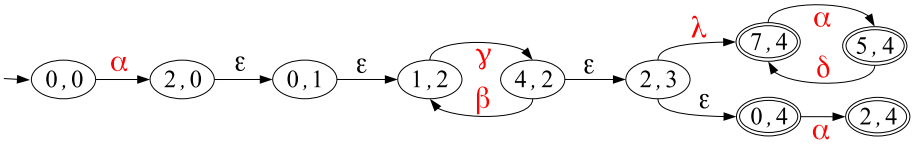


Figure 5: NFA obtained from $Cspace(\mathcal{P}, \mathcal{O})$ in Figure 4, by replacing the symbols of the alphabet (component transitions) with the faults associated in the mapping table $Map(\mathcal{P})$ in Figure 3.

What makes Proposition 2 practical is that a possibly infinite regular language can be always represented as a regular expression. Hence, the set of fault sequences in $\Delta(\mathcal{O})$ can be represented as a regular expression over the alphabet of the faults of $\mathcal{X}$.

Example 9 (*Regular Language*) Consider the $\mathcal{O}$-constrained space of $\mathcal{P}$ introduced in Example 8 and depicted in Figure 4. Displayed in Figure 5 is the NFA obtained by replacing the component transition t marking each transition in $Cspace(\mathcal{P}, \mathcal{O})$ with the (possibly empty) fault associated with t in the mapping table $Map(\mathcal{P})$ in Figure 3. As such, a fault sequence is a string in the language of that NFA. The candidate set is therefore the whole (infinite) set of fault sequences, which is identified by the regular expression

$$\Delta(\mathcal{O}) = \alpha\gamma(\beta\gamma)^* \left(\lambda \left((\alpha\delta)^* \mid \alpha(\delta\alpha)^*\right) \mid \alpha\right) .$$

The candidate set in Example 9 is determined based on inspection of the NFA that is obtained from the $\mathcal{O}$-constrained space by substituting the symbols marking the transitions. What we need, however, is a general technique allowing for the automatic generation of the regular expression of $\Delta(\mathcal{O})$ based on this NFA. To this end, we have exploited an algorithm proposed in the context of sequential circuit state diagrams (Brzozowski & McCluskey, 1963). Essentially, this algorithm takes as input an NFA and generates the regular expression of the language accepted by this NFA. This is exactly what we need to automatize the process of generating the regular language of $\Delta(\mathcal{O})$. Our implemented algorithm is called CANDIDATES, which takes as input an $\mathcal{O}$-constrained space of a DES and generates a regular expression whose language equals the relevant candidate set $\Delta(\mathcal{O})$.

Since the *blind* diagnosis engine is expected to update the explanation upon reception of each new observation o, the $\mathcal{O}$-constrained space needs to be continuously updated based on o. This allows the explanation $Expl(\mathcal{O})$ to be updated when the temporal observation $\mathcal{O}$ is extended with o.

The pseudocode of the BLIND DIAGNOSIS ENGINE algorithm (lines 1–3) takes as input the $\mathcal{O}$-constrained space of a DES $\mathcal{X}$, namely $\mathcal{C} = Cspace(\mathcal{X}, \mathcal{O})$, and a newly-received observation o. As a result, BLIND DIAGNOSIS ENGINE updates $\mathcal{C}$, and generates the candidate set of the extended temporal observation $\mathcal{O} \cup [o]$. First, it extends $\mathcal{C}$ based on the newly-received observation o, thereby creating a new cluster (cf. Figure 6). However, some trajectories in $\mathcal{C}$ before the extension may no longer conform with the extended temporal observation. Consequently, the states and transitions that are no longer connected with the new cluster need to be pruned up to a certain cluster. Subsequently, the algorithm CANDIDATES is applied to the extended $\mathcal{C}$, thereby generating a regular expression $\mathcal{R}$ representing the candidate set $\Delta(\mathcal{O} \cup [o])$, the last element of $Expl(\mathcal{O} \cup [o])$ (cf. Definition 6).

Algorithm 1: BLIND DIAGNOSIS ENGINE

input : $\mathcal{C} = Cspace(\mathcal{X}, \mathcal{O})$, the $\mathcal{O}$-constrained space of a DES $\mathcal{X}$
o, a newly-received observation of $\mathcal{X}$
output: $\mathcal{C}$ is updated based on o
The candidate set Δ of the extended temporal observation $\mathcal{O} \cup [o]$ is generated

1 Extend $\mathcal{C}$ based on o, while pruning the portion of $\mathcal{C}$ that is no longer consistent with o
2 Let $\mathcal{R}$ be the regular expression resulting from calling CANDIDATES on $\mathcal{C}$, where the final states are all the states in the newly-created cluster
3 $\Delta(\mathcal{O} \cup [o]) = \mathcal{R}$.

Example 10 (*Algorithm* BLIND DIAGNOSIS ENGINE) Let $\mathcal{O} = [o_1, o_2] = [vlv, sns]$ be a temporal observation of $\mathcal{P}$. Shown on top of Figure 7 is $Cspace(\mathcal{P}, \mathcal{O})$, which is composed of the clusters C_0, C_1, and C_2. According to Example 7, the explanation of $\mathcal{O}$ is

$$Expl(\mathcal{O}) = [\Delta_0, \Delta_1, \Delta_2] = \left[\alpha?,\ \alpha\left(\alpha? \mid \lambda\left((\alpha\delta)^* \mid \alpha(\delta\alpha)^*\right)\right),\ \alpha\left((\gamma\beta)^* \mid \gamma(\beta\gamma)^* \mid \lambda(\alpha\delta)^*\right)\right].$$

Now, assume that a new observation o_3 occurs, namely *sns*, leading $\mathcal{O}$ to $\mathcal{O}' = [vlv, sns, sns]$. Based on line 1 of BLIND DIAGNOSIS ENGINE, the $\mathcal{O}$-constrained space is updated as shown on the bottom of Figure 7, where a new cluster C_3 is appended and connected with C_2 by an arc marked with the observable transition s_2, which generates the new observation *sns*. Moreover, the part of the graph depicted in gray is spurious (to be pruned), as the three states involved are no longer connected with the (unique) final state in C_3. Notice how the faulty transitions involved are marked with the relevant faults, namely α, β, and γ. Based on line 2, the regular expression $\mathcal{R}$ computed by the CANDIDATES algorithm on $\mathcal{C}'$ will be $\mathcal{R} = \alpha\gamma\,(\beta\gamma)^*$.

Hence, based on line 3, the explanation of $\mathcal{O}$ may be extended with $\mathcal{R}$, thereby becoming the explanation of $\mathcal{O}'$,

$$Expl(\mathcal{O}') = \left[\alpha?,\ \alpha\left(\alpha? \mid \lambda\left((\alpha\delta)^* \mid \alpha(\delta\alpha)^*\right)\right),\ \alpha\left((\gamma\beta)^* \mid \gamma(\beta\gamma)^* \mid \lambda(\alpha\delta)^*\right),\ \underbrace{\alpha\gamma\,(\beta\gamma)^*}_{\mathcal{R}}\right]$$

which equals the explanation of $[vlv, sns, sns]$ determined in Example 7 based on Definition 6.

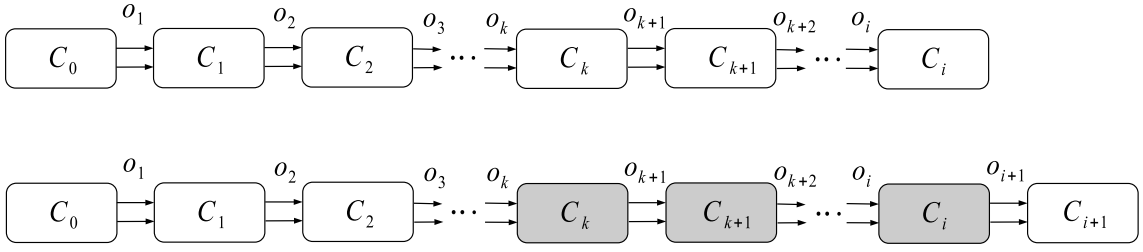


Figure 6: $Cspace(\mathcal{X}, \mathcal{O})$ (top) and $Cspace(\mathcal{X}, \mathcal{O}')$ (bottom), where $\mathcal{O}'$ is the extension of $\mathcal{O}$ by the observation o_{i+1}, with clusters $C_k \ldots C_i$ being pruned after the generation of C_{i+1}.

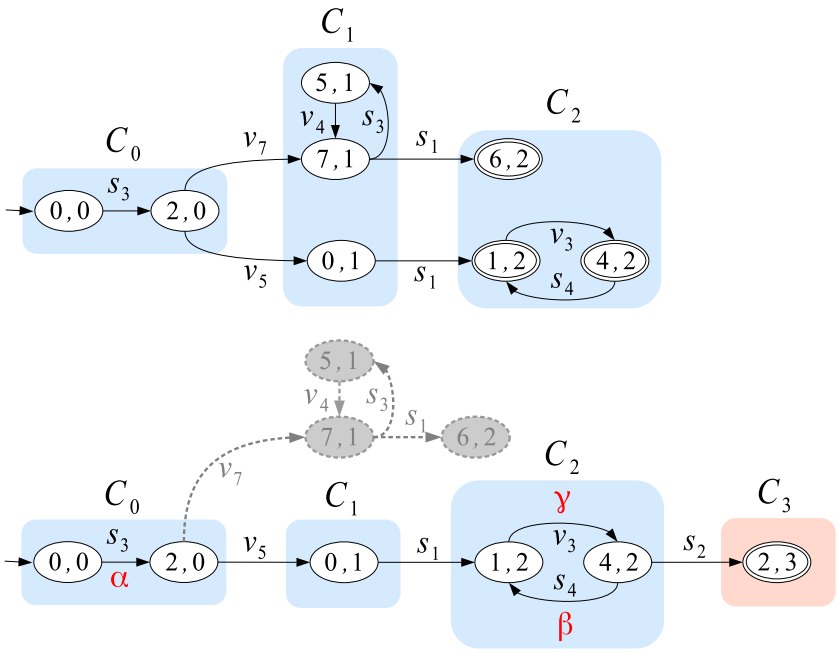


Figure 7: $Cspace(\mathcal{P}, \mathcal{O})$, where $\mathcal{O} = [vlv, sns]$, (top), and relevant update into $Cspace(\mathcal{P}, \mathcal{O}')$, where $\mathcal{O}' = [vlv, sns, sns]$, by the BLIND DIAGNOSIS ENGINE algorithm (bottom).

Next, two additional techniques for diagnosis of fault sequences are presented, which are based on either *total* or *partial* knowledge compilation, called *greedy* diagnosis (Section 5) and *lazy* diagnosis (Section 6), respectively.

5. Greedy Diagnosis

The technique presented in Section 4 for generating an explanation $Expl(\mathcal{O})$ does not exploit any compiled knowledge of a DES $\mathcal{X}$. This is why the BLIND DIAGNOSIS ENGINE algorithm requires the generation of the $\mathcal{O}$-constrained space of $\mathcal{X}$, which may be a problem when the explanation is expected in tight time. To alleviate this drawback, the notion of an *explainer* is introduced (Definition 9) and subsequently exploited for fast diagnosis. Roughly, the explainer of a DES $\mathcal{X}$, denoted $Expr(\mathcal{X})$, is an NFA resulting from preprocessing the specification of $\mathcal{X}$ based on the mapping table $Map(\mathcal{X})$. The alphabet of $Expr(\mathcal{X})$ is a set of triples $(o, \mathcal{L}, f)$, where o is an observation of $\mathcal{X}$, $\mathcal{L}$ is a regular language over the faults of $\mathcal{X}$, and f is a (possibly empty) fault. Intuitively, each state of $Expr(\mathcal{X})$, called a *fault space*, embodies local diagnosis information defined as regular expressions over faults. When a temporal observation $\mathcal{O}$ occurs, $Expr(\mathcal{X})$ allows for the generation of $\Delta(\mathcal{O})$.

Definition 8 (Fault Space (Bertoglio et al., 2020b)) *Let $\bar{x}$ be a state of a DES $\mathcal{X}$, with the mapping table $Map(\mathcal{X})$ involving the set of faults* **F**. *The* fault space *of $\bar{x}$, denoted $Fspace(\bar{x})$, is an NFA*

$$Fspace(\bar{x}) = (\Sigma, X, \tau, x_0) \tag{8}$$

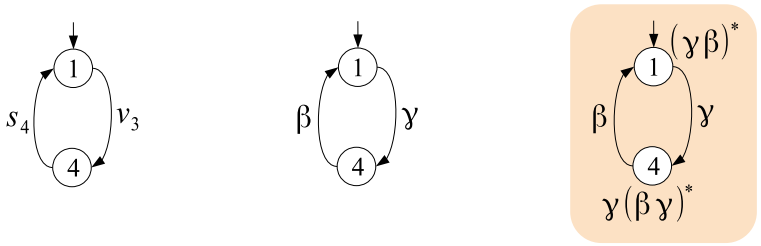


Figure 8: Genesis of *Fspace*(1) based on Definition 8, where 1 is a state in $Space(\mathcal{P})$ (cf. Figure 2).

where $\Sigma = \mathbf{F} \cup \{\varepsilon\}$ *is the alphabet,* X *is the subset of the states of* $Space(\mathcal{X})$ *that are reachable from* $\bar{x}$ *by unobservable transitions only,* $x_0 = \bar{x}$ *is the initial state, and* $\tau : X \times \Sigma \mapsto 2^X$ *is the transition function, where* $\langle x_1, f, x_2 \rangle$ *is one possible arc in* τ *exiting* x_1 *and marked with* f *iff* $\langle x_1, t, x_2 \rangle$ *is a transition in* $Space(\mathcal{X})$ *and* $(t, \varepsilon, f) \in Map(\mathcal{X})$*. Each state* $x \in X$ *is marked with the regular language of the fault-sequence segments of the trajectory segments in* $Space(\mathcal{X})$ *from* $\bar{x}$ *to* x*, denoted* $\mathcal{L}(x)$*. The* diagnosis language *of* $Fspace(\bar{x})$*, denoted* $\mathcal{L}\,(Fspace(\bar{x}))$*, is a regular language on the set of faults of* $\mathcal{X}$ *defined as*

$$\mathcal{L}\,(Fspace(\bar{x})) = \begin{cases} \mathcal{L}(x) & \text{if } X = \{x\} \\ \mathcal{L}(x_1) \mid \ldots \mid \mathcal{L}(x_n) & \text{if } X = \{x_1, \ldots, x_n\}, n \geq 2. \end{cases} \tag{9}$$

Example 11 (*Fault Space*) Displayed on the left of Figure 8 is the unobservable subgraph of $Space(\mathcal{P})$ rooted in the state $1 = (high, closed, op)$ (cf. Figure 2). Based on the mapping table $Map(\mathcal{P})$ in Figure 3, the component transitions s_4 and v_3 are replaced with the relevant faults, as shown in the center of the figure. Eventually, both states 1 and 4 are marked with a regular expression on faults β and γ, indicating the fault-sequence segments generated from 1 to either 1 or 4, thereby giving rise to the fault space *Fspace*(1) depicted on the right of Figure 8. According to eqn. (9), the diagnosis language of the fault space is $\mathcal{L}(Fspace(1)) = \mathcal{L}(1) \mid \mathcal{L}(4) = (\gamma\beta)^* \mid \gamma(\beta\gamma)^*$.

In order to mark the internal states of a fault space with the required regular expressions, we cannot apply the CANDIDATES algorithm as is because several states are involved in the marking process and, in general, each of them is associated with a distinct regular expression. Therefore, we need to extend the algorithm by Brzozowski and McCluskey (1963) to cope with multiple regular expressions, one for each state in the fault space. This extended algorithm is called FAULT SPACE.

5.1 Algorithm FAULT SPACE

The FAULT SPACE algorithm (lines 1–22) takes as input a state $\bar{x}$ in $Space(\mathcal{X})$ and generates as output the fault space $Fspace(\bar{x})$. First, $Fspace(\bar{x})$ is initialized as the unobservable subspace of $Space(\mathcal{X})$ rooted in $\bar{x}$, where $\bar{X}$ is the set of states included. After the substitution of each component transition with the corresponding (possibly empty) fault (line 2), a copy of the current instance of $Fspace(\bar{x})$ is assigned to $\mathcal{N}$, the NFA on which the transformations will be actually applied. Then, a new initial state α_0 is inserted into $\mathcal{N}$ along with an ε-transition $\langle \alpha_0, \varepsilon, \bar{x} \rangle$ (line 4). Furthermore, for each state $\bar{x}_i \in \bar{X}$, $i \in [1 \,..\, n]$, a new final state α_{q_i} is inserted into $\mathcal{N}$ along with an

Algorithm 2: FAULT SPACE

input : $\bar{x}$, a state in $Space(\mathcal{X})$
output: $Fspace(\bar{x})$, the fault space of $\bar{x}$

1 Initialize $Fspace(\bar{x})$ as the unobservable subspace of $Space(\mathcal{X})$ rooted in $\bar{x}$, with $\bar{X} = \{\bar{x}_1, \ldots, \bar{x}_n\}$ being the set of states included
2 Substitute each transition $\langle x, t, x' \rangle$ in $Fspace(\bar{x})$ with $\langle x, f, x' \rangle$, where $(t, \varepsilon, f) \in Map(\mathcal{X})$
3 $\mathcal{N} \leftarrow Fspace(\bar{x})$
4 Insert into $\mathcal{N}$ a new initial state α_0 and an ε-transition $\langle \alpha_0, \varepsilon, \bar{x} \rangle$
5 **foreach** *state* $\bar{x}_i \in \bar{X}$ *of* $\mathcal{N}$, $i \in [1 \,..\, n]$ **do**
6 Insert a final state α_{q_i} and a corresponding ε-transition $\langle \bar{x}_i, \varepsilon, \alpha_{q_i} \rangle$
7 **while** $\mathcal{N}$ *includes a state that is neither the initial state* α_0 *nor a final state* α_{q_i}, $i \in [1 \,..\, n]$, **or** *there are several transitions from* α_0 *to the same final state* α_{q_i}, $i \in [1 \,..\, n]$ **do**
8 **if** *there is a sequence* $Q = [\langle x, r_1, x_1 \rangle, \langle x_1, r_2, x_2 \rangle, \ldots, \langle x_{k-1}, r_k, x' \rangle]$ *of transitions,* $k \geq 2$, *where each* x_i, $i \in [1 \,..\, (k-1)]$, *is neither entered nor exited by any other transition* **then**
9 Substitute the transition $\langle x, (r_1 r_2 \cdots r_k), x' \rangle$ for Q
10 **else if** *there is a set* $S = \{\langle x, r_1, x' \rangle, \langle x, r_2, x' \rangle, \ldots, \langle x, r_k, x' \rangle\}$ *of transitions from* x *to* x' **then**
11 Substitute the transition $\langle x, (r_1 | \cdots | r_k), x' \rangle$ for S
12 **else**
13 Let x be a state of $\mathcal{N}$ where $x \neq \alpha_0$ and $x \neq \alpha_{q_i}$, $i \in [1 \,..\, n]$
14 **foreach** *transition* $\langle x', r', x \rangle$ *entering* x, *where* $x' \neq x$ **do**
15 **foreach** *transition* $\langle x, r'', x'' \rangle$ *exiting* x, *where* $x'' \neq x$ **do**
16 **if** *there is a loop transition* $\langle x, r, x \rangle$ *for* x **then**
17 Insert a transition $\langle x', (r'(r)^* r''), x'' \rangle$ into $\mathcal{N}$
18 **else**
19 Insert a transition $\langle x', (r' r''), x'' \rangle$ into $\mathcal{N}$
20 Remove x and all its entering/exiting transitions
21 **foreach** *transition* $\langle \alpha_0, r_i, \alpha_{q_i} \rangle$ *in* $\mathcal{N}$, $i \in [1 \,..\, n]$ **do**
22 Mark the state $\bar{x}_i \in \bar{X}$ of $Fspace(\bar{x})$ with the regular expression r_i.

ε-transition $\langle \bar{x}_i, \varepsilon, \alpha_{q_i} \rangle$ (lines 5–6). This results in an NFA $\mathcal{N}$ with initial state α_0 and final states $\alpha_{q_1}, \ldots, \alpha_{q_n}$. The actual transformation of $\mathcal{N}$ is performed by the loop in lines 7–20, where the processing continues while either there is a state in $\mathcal{N}$ that is neither initial nor final, or there are several transitions from α_0 to the same final state α_{q_i}, $i \in [1 \,..\, n]$ (line 7). The key idea of Brzozowski and McCluskey (1963) is to simplify the NFA by progressively eliminating states and transitions while preserving the regular language accepted. This can be achieved by changing the alphabet of the NFA (being initially the set of faults) into a set of regular expressions over such faults. There are three simplification rules which are coded within the loop in lines 7–20. In the first rule (lines 8–9), a sequence of transitions, with intermediate states that are neither entered nor exited by other transitions, is replaced with a single transition that is marked with the concatenation of the regular expressions

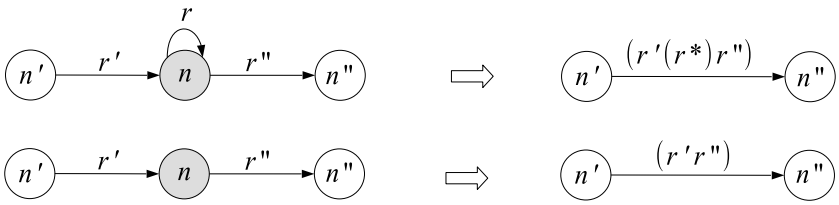


Figure 9: Transformations by FAULT SPACE when removing node n (lines 13–20).

marking the original transitions. In the second rule (lines 10–11), a set of (parallel) transitions is replaced with a single transition marked with the alternative of the regular expressions marking the original transitions. In the third rule (lines 13–20), an internal node n of $\mathcal{N}$ is removed, along with its entering/exiting transitions, and replaced by a set of equivalent transitions. With reference to Figure 9, if there is a loop transition $\langle n, r, n\rangle$ for n, then, for each transition $\langle n', r', n\rangle$ entering n and for each transition $\langle n, r'', n''\rangle$ exiting n, a new transition $\langle n', (r'(r^*)r''), n''\rangle$ is created. Instead, if the loop transition $\langle n, r, n\rangle$ is missing, then the new transition is $\langle n', (r'r''), n''\rangle$. Eventually, n and all its entering/exiting transitions are eliminated. At the end of the loop (lines 21–22), $\mathcal{N}$ necessarily includes only transitions from the initial state to a final state, namely $\langle \alpha_0, r_i, \alpha_{q_i}\rangle$, $i \in [1\,..\,n]$, where r_i represents the regular expression on faults that is used to mark state $\bar{x}_i$ in $Fspace(\bar{x})$.

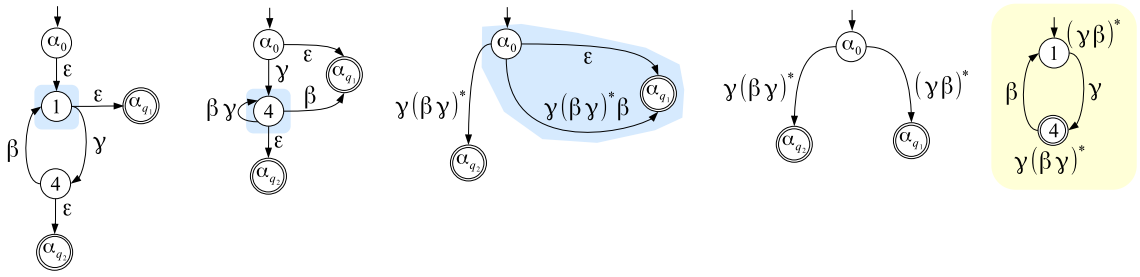


Figure 10: Tracing of the FAULT SPACE algorithm for generating $Fspace(1)$ (cf. Figure 8).

Example 12 (*Algorithm* FAULT SPACE) Traced in Figure 10 is the generation of $Fspace(1)$ (cf. Figure 8) by the FAULT SPACE algorithm. The NFA $\mathcal{N}$ determined in line 3 of the algorithm is displayed on the left side of the figure. Then, states 1 and 4 are removed one after another, leading to the NFA depicted on the center of the figure. At this point, the two parallel transitions from α_0 to α_{q_1} are merged into a single transition marked with $(\gamma\beta)^*$, which is equivalent to $\gamma(\beta\gamma)^*\beta \mid \varepsilon$. Eventually, the main loop terminates and, based on lines 21–22, states 1 and 4 of $Fspace(1)$ are marked with the regular expressions labeling the relevant transitions in the final configuration of $\mathcal{N}$, namely $(\gamma\beta)^*$ and $\gamma(\beta\gamma)^*$, respectively. As expected, the resulting automaton, highlighted on the right side of Figure 10, equals the fault space $Fspace(1)$ in Figure 8.[7]

7. The equality of regular expressions is not strictly necessary, as a regular expression marking a state of the fault space may differ from the regular expression we have foreseen intuitively for that state. However, the languages of these different regular expressions need to be the same. In other words, what is actually essential is the *equivalence* of possible different regular expressions marking the same state (same regular language, which is guaranteed by the algorithm), not their equality.

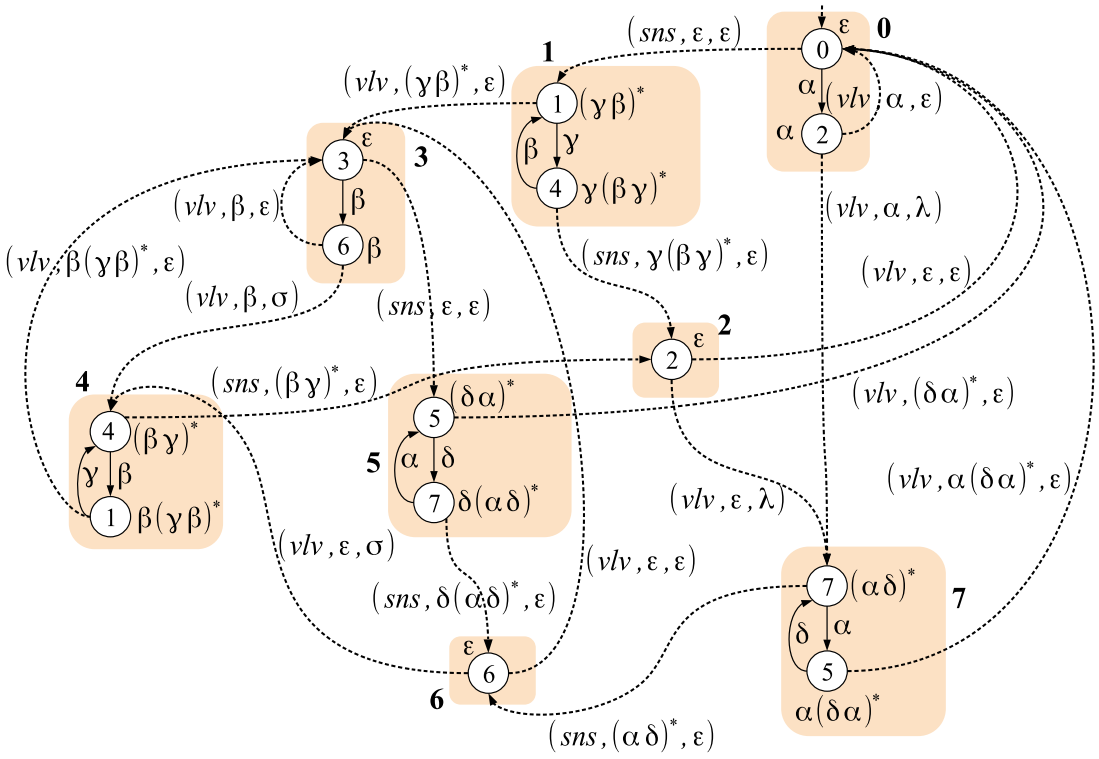


Figure 11: $Expr(\mathcal{P})$, the explainer of the DES $\mathcal{P}$.

A fault space is the building block of a knowledge structure called an *explainer*, which allows for the online generation of the explanation of a temporal observation.

Definition 9 (Explainer (Bertoglio et al., 2020b)) *Let $\mathcal{X}$ be a DES with $Space(\mathcal{X}) = (\Sigma, X, \tau, x_0)$, let $\mathbf{O}$ be the set of observations of $\mathcal{X}$, let $\mathbf{F}$ be the set of faults of $\mathcal{X}$, and let $\mathbf{L}$ be the set of regular languages over $\mathbf{F}$. The* explainer *of $\mathcal{X}$, denoted $Expr(\mathcal{X})$, is an NFA*

$$Expr(\mathcal{X}) = \left(\Sigma', X', \tau', x'_0\right) \tag{10}$$

where $\Sigma' \subseteq \mathbf{O} \times \mathbf{L} \times (\mathbf{F} \cup \{\varepsilon\})$ is the alphabet, X' is the set of states, where each state is a fault space of a specific state of $Space(\mathcal{X})$, $x'_0 = Fspace(x_0)$ is the initial state, and τ' is the transition function, $\tau' : (X' \times X) \times \Sigma' \mapsto 2^{(X' \times X)}$, where $\langle (x'_1, x_1), (o, \mathcal{L}(x_1), f), (x'_2, x_2) \rangle$ is an arc in τ' iff x_1 is a state in x'_1, $\langle x_1, t, x_2 \rangle \in \tau$, $(t, o, f) \in Map(\mathcal{X})$, $o \neq \varepsilon$, and $x'_2 = Fspace(x_2)$.

Example 13 (*Explainer*) Displayed in Figure 11 is the explainer of the DES $\mathcal{P}$, namely $Expr(\mathcal{P})$ (cf. $Space(\mathcal{P})$ in Figure 2). It includes eight states (renamed $\mathbf{0} \ldots \mathbf{7}$), incidentally, one state $Fspace(p)$ for each state p in $Space(\mathcal{P})$. For instance, there are three arcs exiting the initial state $\mathbf{0} = Fspace(0)$: $\langle (\mathbf{0}, 0), (sns, \varepsilon, \varepsilon), (\mathbf{1}, 1) \rangle$, $\langle (\mathbf{0}, 2), (vlv, \alpha, \varepsilon), (\mathbf{0}, 0) \rangle$, and $\langle (\mathbf{0}, 2), (vlv, \alpha, \lambda), (\mathbf{7}, 7) \rangle$. Notice how, to distinguish, the arcs (transitions) between the states of $Expr(\mathcal{P})$ are dashed, while the arcs (transitions) between states within each fault space are plain.

A trajectory in $Expr(\mathcal{X})$ is a contiguous sequence of transitions in $Expr(\mathcal{X})$ starting from the initial state. Likewise, the temporal observation of a trajectory T in $Expr(\mathcal{X})$ is defined as

$$Obs(T) = [\, o \mid (o, \mathcal{L}, f) \text{ is the triple marking a transition in } T \,]\,. \tag{11}$$

The notion of the observation language of the space of $\mathcal{X}$ defined in eqn. (4) can be naturally extended to $Expr(\mathcal{X})$ as follows:

$$OBS\,(Expr(\mathcal{X})) = \{Obs(T) \mid T \text{ is a trajectory in } Expr(\mathcal{X})\}\,. \tag{12}$$

Proposition 3 *The observation language of the explainer of $\mathcal{X}$ equals the observation language of the space of $\mathcal{X}$, namely*

$$OBS\,(Expr(\mathcal{X})) = OBS(Space(\mathcal{X}))\,. \tag{13}$$

This can be proven easily by considering that each sequence of observations generated by a trajectory in $Expr(\mathcal{X})$ can be generated by a trajectory in $Space(\mathcal{X})$ and vice versa. In fact, each state in $Expr(\mathcal{X})$ is a fault space, namely, decoration aside, a subspace of $Space(\mathcal{X})$ involving unobservable trajectory segments.

Given an explainer $Expr(\mathcal{X})$, the explanation of a temporal observation $\mathcal{O}$ is generated by tracing $\mathcal{O}$ on $Expr(\mathcal{X})$, thereby yielding a *trace*, as formalized below.

Definition 10 (Trace (Bertoglio et al., 2020b)) *Let $Expr(\mathcal{X}) = (\Sigma, X, \tau, x_0)$ be an explainer and let $\mathcal{O} = [o_1, \ldots, o_n]$ be a temporal observation of the DES $\mathcal{X}$. The* trace *of $\mathcal{O}$ is a directed graph*

$$Trace(\mathcal{O}) = (N, A, \mu_0) \tag{14}$$

where $N = \{\mu_0, \mu_1, \ldots, \mu_n\}$ is the multiset of nodes, A is the multiset of arcs, and μ_0 is the initial node. Each node $\mu_i \in N$, $i \in [0\,..\,n]$, is a subset of the states of $Expr(\mathcal{X})$, with $\mu_0 = \{x_0\}$. Each node $\mu_i \neq \mu_0$ contains the states of $Expr(\mathcal{X})$ that are reached in $Expr(\mathcal{X})$ from the states in μ_{i-1} via a transition marked with a triple where the observation is o_i. Each arc exiting a state $x \in \mu_i$ is marked with a pair $(\mathcal{L}, \mathcal{L}')$, where $\mathcal{L}$ and $\mathcal{L}'$ are languages of fault-sequence segments. There is an arc from a state $x \in \mu_i$ to a state $x' \in \mu_{i+1}$, $i \in [0\,..\,(n-1)]$, marked with $(\mathcal{L}, \mathcal{L}')$, iff there is a transition in $Expr(\mathcal{X})$ from a fault-space state in x to a fault-space state in x' that is marked with $(o_{i+1}, \mathcal{L}, f)$, whereas $\mathcal{L}'$ is defined as follows. Let $\mathcal{R}$ be a regular expression over the faults of $\mathcal{X}$, which is either ε, when $i = 0$, or $(\mathcal{L}'_1 | \mathcal{L}'_2 | \ldots | \mathcal{L}'_k)$, when $i \neq 0$, where $(\mathcal{L}_j, \mathcal{L}'_j)$ is the pair marking the j-th arc entering $x \in \mu_i$, $j \in [1\,..\,k]$. Then, $\mathcal{L}' = \mathcal{R}\mathcal{L}f$.

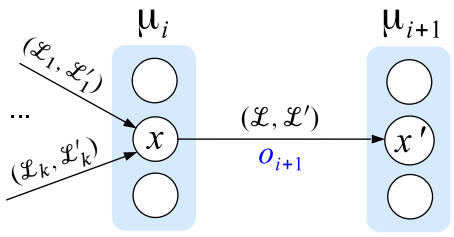


Figure 12: Fragment of a generic trace.

Intuitively, given a temporal observation $\mathcal{O}$ of a DES $\mathcal{X}$, *Trace*$(\mathcal{O})$ traces on the explainer *Expr*$(\mathcal{O})$ the trajectories of $\mathcal{X}$ that conform with the prefixes of $\mathcal{O}$ by marking the arcs with relevant regular expressions over faults. Consider Figure 12, which represents a fragment of a generic trace. Specifically, it represents an arc from a state x in a node μ_i to a state x' in a node μ', which is marked with the pair $(\mathcal{L}, \mathcal{L}')$. As such, the arc corresponds to the $(i+1)$-th observation of $\mathcal{O}$, namely o_{i+1}. According to Definition 10, language $\mathcal{L}$ equals the language marking a state x_1 of $\mathcal{X}$ within x that is exited by an observable transition in *Space*$(\mathcal{X})$ generating the observation o_{i+1} and entering a state x_2 of $\mathcal{X}$ within x'. Thus, $\mathcal{L}$ is the language of the fault-sequence segments of the trajectory segments from the initial state of x to x_1. The actual trace of the fault sequences is represented by the language $\mathcal{L}'$, which accounts for the trajectories of $\mathcal{X}$ from the initial state of $\mathcal{X}$ to x_2. In fact, if $i = 0$, that is, if x is the initial state x_0 of *Expr*$(\mathcal{O})$, then $\mathcal{R} = \varepsilon$ and, hence, $\mathcal{R}\mathcal{L}f = \mathcal{L}f$ is the language of the fault sequences of the trajectories of $\mathcal{X}$ up to state x_2, as f is the (possibly empty) fault associated with the transition $\langle x_1, t, x_2\rangle$ in *Space*$(\mathcal{X})$, namely (t, o_{i+1}, f) is a triple in the mapping table of $\mathcal{X}$. If, instead, $i > 0$, then $\mathcal{L}'$ needs to account for all the paths entering x, whose fault sequence languages are expressed by $\mathcal{L}'_1, \ldots, \mathcal{L}'_k$. Thus, $\mathcal{R}$ is the alternative of such languages, namely $(\mathcal{L}'_1|\mathcal{L}'_2|\ldots|\mathcal{L}'_k)$. Hence, we have $\mathcal{L}' = \mathcal{R}\mathcal{L}f = (\mathcal{L}'_1|\mathcal{L}'_2|\ldots|\mathcal{L}'_k)\mathcal{L}f$, the regular expression identifying the language of the fault sequences of the trajectories of $\mathcal{X}$ from the initial state to x_2.

Example 14 (*Trace*) With reference to the explainer in Figure 11, let $\mathcal{O} = [vlv, sns]$ be a temporal observation of the DES $\mathcal{P}$. Outlined in Figure 13 is the trace of $\mathcal{O}$. There is an arc $\langle 0, (\alpha, \alpha), 0\rangle$ from μ_0 to μ_1 since there is a transition $\langle(\mathbf{0}, 2), (vlv, \alpha, \varepsilon), (\mathbf{0}, 0)\rangle$ in *Expr*$(\mathcal{P})$. There is an arc $\langle 0, (\alpha, \alpha\lambda), 7\rangle$ since there is a transition $\langle(\mathbf{0}, 2), (vlv, \alpha, \lambda), (\mathbf{7}, 7)\rangle$ in *Expr*$(\mathcal{P})$. There is an arc $\langle 0, (\varepsilon, \alpha), 1\rangle$ from μ_1 to μ_2 as there is a transition $\langle(\mathbf{0}, 0), (sns, \varepsilon, \varepsilon), (\mathbf{1}, 1)\rangle$ in *Expr*$(\mathcal{P})$. There is an arc $\langle 7, ((\alpha\delta)^*, \alpha\lambda(\alpha\delta)^*), 6\rangle$ from μ_1 to μ_2 since there is a transition $\langle(\mathbf{7}, 7), (sns, (\alpha\delta)^*, \varepsilon), (\mathbf{6}, 6)\rangle$ in *Expr*$(\mathcal{P})$.

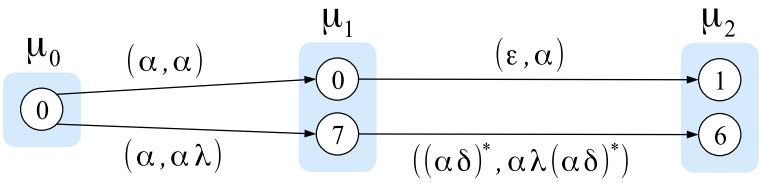


Figure 13: Trace of temporal observation $\mathcal{O} = [vlv, sns]$ (cf. *Expr*$(\mathcal{P})$ in Figure 11).

Definition 11 (Candidate Language) *Let $\mu = \{x_1, \ldots, x_m\}$ be a node of Trace$(\mathcal{O})$. Let x be a fault-space state in μ. Let $\mathcal{L}_{\text{in}}(x)$ denote either ε, when $\mu = \mu_0$, or the alternative $\mathcal{L}'_1|\mathcal{L}'_2|\ldots|\mathcal{L}'_p$, when $\mu \neq \mu_0$, where $(\mathcal{L}_j, \mathcal{L}'_j)$, $j \in [1\,..\,p]$, is the pair marking the j-th arc entering x in μ. Let $\mathcal{L}_{\text{out}}(x)$ be the alternative $\mathcal{L}(\bar{x}_1)|\mathcal{L}(\bar{x}_2)|\ldots|\mathcal{L}(\bar{x}_r)$, where $\bar{x}_1, \ldots, \bar{x}_r$ are the states within the fault space x and $\mathcal{L}(\bar{x}_k)$, $k \in [1\,..\,r]$, is the language marking $\bar{x}_k$. The* candidate language *of μ, denoted $\Delta(\mu)$, is an alternative of pairwise concatenated languages, specifically*

$$\Delta(\mu) = \mathcal{L}_{\text{in}}(x_1)\mathcal{L}_{\text{out}}(x_1) \mid \ldots \mid \mathcal{L}_{\text{in}}(x_m)\mathcal{L}_{\text{out}}(x_m)\,. \tag{15}$$

Example 15 (*Candidate Language*) With reference to the trace shown in Figure 13 and according to Definition 11, we have (cf. Figure 11):

$$\Delta(\mu_0) = \varepsilon\mathcal{L}(\mathbf{0}) = \varepsilon\,(\mathcal{L}(0)\,|\,\mathcal{L}(2)) = \varepsilon(\varepsilon\,|\,\alpha) = \alpha?$$
$$\Delta(\mu_1) = \alpha\mathcal{L}(\mathbf{0})\,|\,\alpha\lambda\mathcal{L}(\mathbf{7}) = \alpha\alpha?\,|\,\alpha\lambda\left((\alpha\delta)^*\,|\,\alpha(\delta\alpha)^*\right) = \alpha\left(\alpha?\,|\,\lambda\left((\alpha\delta)^*\,|\,\alpha(\delta\alpha)^*\right)\right)$$
$$\Delta(\mu_2) = \alpha\mathcal{L}(\mathbf{1})\,|\,\alpha\lambda(\alpha\delta)^*\mathcal{L}(\mathbf{6}) = \alpha\left((\gamma\beta)^*\,|\,\gamma(\beta\gamma)^*\right)\,|\,\alpha\lambda(\alpha\delta)^*\varepsilon = \alpha\left((\gamma\beta)^*\,|\,\gamma(\beta\gamma)^*\,|\,\lambda(\alpha\delta)^*\right)$$

Comparing these results with the explanation computed in Example 7, notice how $\Delta(\mu_0)$, $\Delta(\mu_1)$, and $\Delta(\mu_2)$ equal Δ_0, Δ_1, and Δ_2, respectively. In other words, $\Delta(\mu_i)$ equals the candidate set $\Delta(\mathcal{O}_i)$, $i \in [0\,..\,2]$. This is no coincidence, as claimed in Proposition 4.

Proposition 4 *Let $\mu_0, \mu_1, \ldots, \mu_n$ be the nodes in Trace$(\mathcal{O})$, where $\mathcal{O} = [o_1, \ldots, o_n]$. Let $\mathcal{O}_i = [o_1, \ldots, o_i]$, $i \in [0\,..\,n]$. We have*

$$\Delta(\mu_i) = \Delta(\mathcal{O}_i)\,. \tag{16}$$

Proof (*sketch*). According to Definition 5, $\Delta(\mathcal{O}_i)$, $i \in [0\,..\,n]$, is the candidate set of a prefix $\mathcal{O}_i = [o_1, \ldots, o_i]$ of $\mathcal{O}$. More specifically, $\Delta(\mathcal{O}_i)$ is the language of the fault sequences of the trajectories T_i that conform with $\mathcal{O}_i$. There is an isomorphism between the trajectories T_i and the paths in *Trace*$(\mathcal{O})$, including the transitions within the nodes of *Trace*$(\mathcal{O})$, from the initial state of μ_0 to a state in μ_i. In fact, each node μ_i in *Trace*$(\mathcal{O})$ is a fault space (Definition 8), which involves the portion of *Space*$(\mathcal{X})$ that is reachable from the initial state of μ_i by unobservable transitions only, each transition being marked with a (possibly empty) fault. Let $(\mathcal{L}, \mathcal{L}')$ be the pair marking an arc in *Trace*$(\mathcal{O})$ entering the initial state $\bar{x}$ of a state in μ_i. As such, $\mathcal{L}'$ is the language of the fault sequences relevant to any trajectory of $\mathcal{X}$ ending in $\bar{x}$. This can be proven by induction. According to Definition 10, if the considered arc exits μ_0, then $\mathcal{L}' = \mathcal{L}f$, where $\mathcal{L}$ is the language of fault sequences of the unobservable trajectories up to the state exited by the arc and f is the (possibly empty) fault associated with the observable transition corresponding to the arc. Hence, $\mathcal{L}f$ is the language of the fault sequences relevant to the trajectories up to the transition corresponding to the arc. If this property holds for an arc entering a state in μ_i, $i \in [1\,..\,(n-1)]$, then it also holds for a successive arc entering a state in μ_{i+1}. In fact, based on Definition 10, $\mathcal{L}' = \mathcal{R}\mathcal{L}f$, where $\mathcal{R} = (\mathcal{L}'_1|\mathcal{L}'_2|\ldots|\mathcal{L}'_k)$, with $(\mathcal{L}_j, \mathcal{L}'_j)$, $j \in [1\,..\,k]$, being the pair marking an arc in *Trace*$(\mathcal{O})$ entering a state in μ_i. Thus, $\mathcal{R}$ accounts for the fault sequences relevant to the trajectories up to the initial state of the fault space x in μ_i that is exited by the arc. Moreover, $\mathcal{L}$ accounts for the fault-sequence segments of the trajectory segments from the initial state of x to the state in x that is exited by the arc (marked by $\mathcal{L}$). Hence, $\mathcal{L}'$ accounts for the fault sequences of the trajectories up to the initial state of the state in μ_{i+1} that is entered by the arc. Based on Definition 10, $\Delta(\mu_i)$ is the alternative of the concatenated languages $\mathcal{L}_{\text{in}}(x)\mathcal{L}_{\text{out}}(x)$, where x is a state in μ_i. Besides, if $i > 0$, then $\mathcal{L}_{\text{in}}(x)$ is the alternative of the languages in second position within the pairs marking the arcs entering x; otherwise, if $i = 0$, then $\mathcal{L}_{\text{in}}(x) = \varepsilon$. Thus, $\mathcal{L}_{\text{in}}(x)$ accounts for the fault sequences relevant to the trajectories up to the arcs entering x. Similarly, $\mathcal{L}_{\text{out}}(x)$ is the alternative of the languages marking the states within x. Thus, $\mathcal{L}_{\text{out}}(x)$ accounts for the fault-sequence segments relevant to the trajectory segments within x starting in the initial state of x. Hence, $\Delta(\mu_i)$ is the language of fault sequences relevant to the trajectories T_i that conform with $\mathcal{O}_i$; in other words, according to Definition 5, $\Delta(\mu_i) = \Delta(\mathcal{O}_i)$. □

Corollary 4.1 below is a direct consequence of Proposition 4 and Definition 6.

Corollary 4.1 *Let $\mu_0, \mu_1, \ldots, \mu_n$ be the nodes in Trace($\mathcal{O}$), where $\mathcal{O} = [o_1, \ldots, o_n]$. We have*

$$Expl(\mathcal{O}) = [\Delta(\mu_0), \Delta(\mu_1), \ldots, \Delta(\mu_n)]\,. \tag{17}$$

Based on Corollary 4.1, once the explainer of a DES $\mathcal{X}$ has been constructed offline, it can be exploited online by the diagnosis engine to generate the explanation of any specific temporal observation of $\mathcal{X}$. The GREEDY DIAGNOSIS ENGINE algorithm serves this purpose more efficiently than BLIND DIAGNOSIS ENGINE does (cf. Section 4), as low-level model-based reasoning is avoided altogether and, moreover, diagnosis information is available directly within the fault spaces and connecting arcs.

5.2 Algorithm GREEDY DIAGNOSIS ENGINE

The GREEDY DIAGNOSIS ENGINE algorithm assumes the availability of a *complete* explainer. This assumption is to be considered realistic for DESs with a few components only, however, owing to the exponential explosion of the number of states in the explainer.[8]

Algorithm 3: GREEDY DIAGNOSIS ENGINE

input : $Expr(\mathcal{X})$, the explainer of a DES $\mathcal{X}$
$\mathcal{T} = Trace(\mathcal{O})$, the trace of a temporal observation $\mathcal{O}$ of $\mathcal{X}$
o, a newly-received observation of $\mathcal{X}$
output: $\mathcal{T}$ is extended based on o
The candidate set Δ of the extended temporal observation $\mathcal{O} \cup [o]$ is generated

1 Extend $\mathcal{T}$ by a new node μ based on the new observation o, as specified in Definition 10
2 Generate the candidate language $\Delta(\mu)$, as specified in Definition 11
3 $\Delta(\mathcal{O} \cup [o]) \leftarrow \Delta(\mu)$.

GREEDY DIAGNOSIS ENGINE (lines 1–3) takes as input an explainer $Expr(\mathcal{X})$, the trace $\mathcal{T}$ of a temporal observation $\mathcal{O}$, and a new observation o. As a result, it extends the trace $\mathcal{T}$ with a new node and generates the candidate set of the extended temporal observation $\mathcal{O} \cup [o]$, namely the new element of the explanation of the (extended) temporal observation. First, in line 1, the trace $\mathcal{T}$ is extended by a new node μ, as detailed in Definition 10. Then, in line 2, the candidate language of μ is computed, as specified in Definition 11. According to Proposition 4, the candidate language $\Delta(\mu)$ equals the candidate set of the extended temporal observation $\mathcal{O} \cup [o]$, as stated in line 3.

Example 16 (*Algorithm* GREEDY DIAGNOSIS ENGINE) With reference to the explainer displayed in Figure 11, let $\mathcal{O} = [vlv, sns, sns]$ be a temporal observation of $\mathcal{P}$. Shown in Table 2 are the results of the GREEDY DIAGNOSIS ENGINE algorithm when executed on each observation o in $\mathcal{O}$. Each row of the table indicates the newly-received observation, the trace that has been extended with the new node μ, and the diagnosis set $\Delta(\mu)$. As expected, the sequence of $\Delta(\mu)$'s generated at each execution equals the explanation of $\mathcal{O}$ computed in Example 7.

The diagnosis technique presented in this section is called *greedy* inasmuch it assumes the availability of an *entire* explainer, an over-assumption in real applications. To mitigate the complexity

8. A more viable approach to knowledge compilation for DESs of more realistic size is presented in Section 6, where a partial explainer is exploited and possibly upgraded online by a *lazy* diagnosis engine.

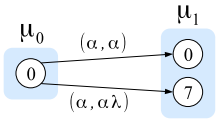
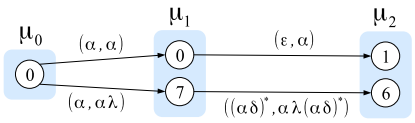
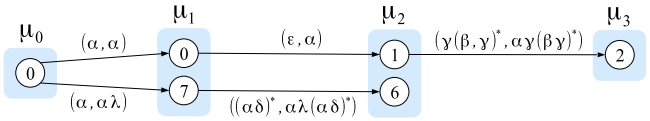
o	*Trace*	$\Delta(\mu)$
		$\alpha?$
vlv		$\alpha\,(\alpha? \mid \lambda\,((\alpha\delta)^* \mid \alpha(\delta\alpha)^*))$
sns		$\alpha\,((\gamma\beta)^* \mid \gamma(\beta\gamma)^* \mid \lambda(\alpha\delta)^*)$
sns		$\alpha\gamma\,(\beta\gamma)^*$

Table 2: Outputs of the multiple executions of the GREEDY DIAGNOSIS ENGINE algorithm when receiving the observations sequentially, namely [*vlv*, *sns*, *sns*].

of the explainer, an alternative technique is presented in the next section, which is called *lazy* inasmuch it does not require total knowledge compilation. Instead, only a (possibly tiny) portion of an explainer is compiled offline, which is then exploited online by a *lazy* diagnosis engine. The idea is to postpone the extension of the partial explainer while performing the diagnosis task, but only to an extent that is necessary and sufficient for the computation of the relevant candidate set.

6. Lazy Diagnosis

In real applications, assuming that an explainer is available in its entirety is impractical, even if generated offline, because of the exponential explosion of the set of states involved. Hence, we propose a viable approach in which a *partial explainer* is generated upfront and subsequently extended either offline or when being operated online. A similar consideration applies to $Space(\mathcal{X})$, whose construction is assumed to be impractical. Hence, hereafter, a notation like $\langle x, t, x' \rangle$ in $Space(\mathcal{X})$ does not assume that $Space(\mathcal{X})$ is available: it is only a shorthand for stating that component transition t is triggerable at state x of $\mathcal{X}$, thereby leading $\mathcal{X}$ to a new state x'.

Definition 12 (Partial Explainer (Bertoglio et al., 2020b)) *Let $Expr(\mathcal{X})$ be an explainer. A* partial explainer *of $\mathcal{X}$, denoted $Pexpr(\mathcal{X})$, is a connected subgraph of $Expr(\mathcal{X})$ that includes the initial state of $Expr(\mathcal{X})$.*

Example 17 (*Partial Explainer*) Consider the explainer of $\mathcal{P}$ displayed in Figure 11. A partial explainer of $\mathcal{P}$ is shown in Figure 14, which involves three states, namely **0**, **1**, and **7**, and three transitions, namely $\langle(\mathbf{0}, 0), (sns, \varepsilon, \varepsilon), (\mathbf{1}, 1)\rangle$, $\langle(\mathbf{0}, 2), (vlv, \alpha, \varepsilon), (\mathbf{0}, 0)\rangle$, and $\langle(\mathbf{0}, 2), (vlv, \alpha, \lambda), (\mathbf{7}, 7)\rangle$.

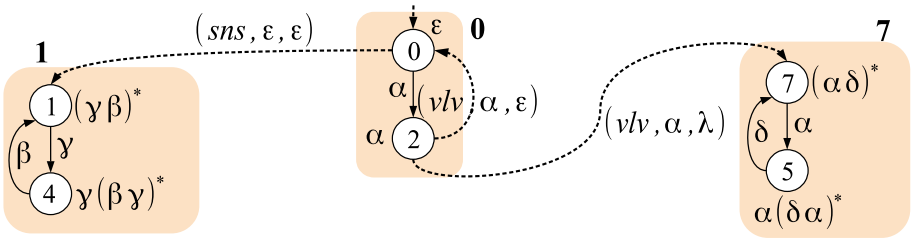


Figure 14: *Pexpr*($\mathcal{P}$), a partial explainer of $\mathcal{P}$ (cf. Figure 11).

A partial explainer of a DES can be initialized (and, possibly, subsequently extended) offline based on meaningful *behavioral scenarios* of the DES. The idea is to be sure that the initial partial explainer embodies all the states and transitions that are necessary for generating the candidate set of any temporal observation relevant to these scenarios. The rationale is that, if a scenario is critical for the DES, it should be diagnosed efficiently online, which is only possible when the temporal observation can be matched on the partial explainer (intuitively, when the temporal observation is in the language of the partial explainer). Otherwise, the matching process may require the online generation of the missing states (fault spaces) and transitions, a time-consuming task.

Definition 13 (Behavioral Scenario (Bertoglio et al., 2020b)**)** *Let $\mathcal{X}$ be a DES and $\mathbb{T}$ a subset of the component transitions in $\mathcal{X}$. A* behavioral scenario *of $\mathcal{X}$ is a pair $(\mathbb{T}, \mathcal{L})$, where $\mathcal{L}$ is a regular language over $\mathbb{T}$.*

Notice that the regular language $\mathcal{L}$ of a behavioral scenario is defined over a *subset* of the component transitions $\mathbf{T}$ of $\mathcal{X}$, namely $\mathbb{T}$. One may ask why not considering $\mathbf{T}$ instead of $\mathbb{T}$. The reason is that, focusing on $\mathbb{T}$ rather than $\mathbf{T}$, the specification of $\mathcal{L}$ is more concise and readable, as the irrelevant transitions are missing. In other words, $\mathbb{T}$ only includes the transitions in $\mathbf{T}$ that are strictly relevant to the definition of the scenario.

Example 18 (*Behavioral Scenario*) With reference to DES $\mathcal{P}$, a scenario where the *only* malfunction is *the valve being stuck closed* can be defined as $\mathcal{S} = (\mathbb{T}, \mathcal{L})$, where $\mathbb{T} = \{s_3, s_4, v_1, v_2, v_3, v_4, v_7, v_8\}$ and $\mathcal{L} = v_3 v_3^+$ (repetition at least twice of the faulty transition v_3). Notice that $\mathbb{T}$ does not include all the transitions of the transducer and the valve. Specifically, the transitions s_1, s_2, v_5, and v_6 are missing, as they are irrelevant to the scenario. Instead, all the faulty transitions are included in $\mathbb{T}$ as it is essential to be sure that just the transition v_3 occurs (at least twice). In other words, $\mathbb{T}$ needs to include not only the component transitions that are involved in the regular expression of $\mathcal{L}$, but also the component transitions that cause a mismatch of the scenario. For instance, if the transition v_4 occurs, it means that the assumption of the single fault based on v_3 is violated (a mismatch). Likewise, if the occurring transition is v_1, it means that the valve is not stuck closed because it opens correctly, again a violation of the scenario. By contrast, the occurrence of any of the transitions that are missing in $\mathbb{T}$, such as s_1 or v_7, is irrelevant to the specification of the scenario, and this is precisely why they are missing.

To upgrade a partial explainer *Pexpr*($\mathcal{X}$) so that it embeds the set of temporal observations generated by a scenario $\mathcal{S}$, we need to *synchronize* $\mathcal{S}$ with the behavior of $\mathcal{X}$. Roughly, this resembles the generation of an $\mathcal{O}$-constrained space (Definition 7), where the scenario plays the role of the temporal observation $\mathcal{O}$. The result of this synchronization is called a *scenario abduction*.

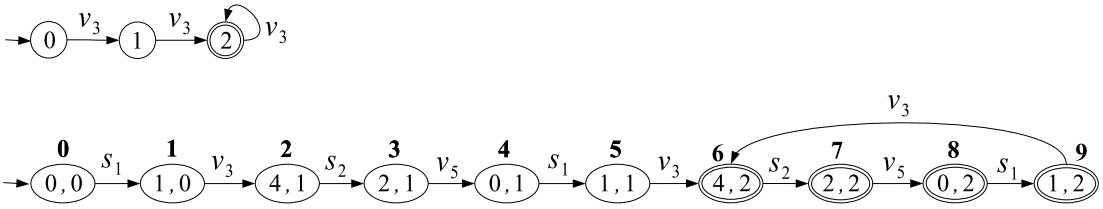


Figure 15: DFA recognizing the scenario $\mathcal{S}$ defined in Example 18 (top) and $Abd(\mathcal{P}, \mathcal{S})$, the abduction of $\mathcal{S}$ in $\mathcal{P}$ (bottom).

Definition 14 (Scenario Abduction (Bertoglio et al., 2020b)) *Let $\mathcal{S} = (\mathbb{T}, \mathcal{L})$ be a scenario of a DES $\mathcal{X}$. The* restriction *of a trajectory T in $Space(\mathcal{X})$ on $\mathbb{T}$ is a sequence $T_{\mathbb{T}} = [\, t \mid t \in T, t \in \mathbb{T}\,]$. The* abduction *of $\mathcal{S}$ in $\mathcal{X}$, denoted $Abd(\mathcal{X}, \mathcal{S})$, is a DFA whose language is the set of the trajectories T in $Space(\mathcal{X})$ where $T_{\mathbb{T}} \in \mathcal{L}$.*

Definition 14 is declarative in nature, as it does not provide any practical hint on the structure of the states of $Abd(\mathcal{X}, \mathcal{S})$. On the one hand, since the language of the abduction is a set of trajectories of $\mathcal{X}$, the transitions of $Abd(\mathcal{X}, \mathcal{S})$ need to be marked with component transitions, exactly as in an $\mathcal{O}$-constrained space. On the other, since the synchronization is with a scenario rather than a temporal observation, a state of $Abd(\mathcal{X}, \mathcal{S})$ is expected to be a pair (x, d), where x is a state of $\mathcal{X}$ and d is a state of the DFA recognizing the language $\mathcal{L}$ of the scenario. A state (x, d) is final in $Abd(\mathcal{X}, \mathcal{S})$ when d is final in the recognizer of $\mathcal{L}$.

Example 19 (*Scenario Abduction*) With reference to the behavioral scenario $\mathcal{S}$ defined in Example 18, shown in Figure 15 are the DFA recognizing $\mathcal{S}$ (top) and the abduction $Abd(\mathcal{P}, \mathcal{S})$, where the states are renamed $\mathbf{0} \ldots \mathbf{9}$ (bottom). Each state of the abduction is a pair (p, d), where p is a state of $\mathcal{P}$ (cf. Figure 2) and d is a state of the recognizer (DFA). Since the only final state in the recognizer is 2, the final states of $Abd(\mathcal{P}, \mathcal{S})$ are $\mathbf{6} = (4, 2)$, $\mathbf{7} = (2, 2)$, $\mathbf{8} = (0, 2)$, and $\mathbf{9} = (1, 2)$. It is easy to check that any trajectory in $Abd(\mathcal{P}, \mathcal{S})$ gives rise to the *valve stuck to closed* scenario, with the transition v_3 of the valve occurring at least twice, without any other fault.

To guarantee that a scenario is somewhat incorporated in a partial explainer $Pexpr(\mathcal{X})$, that is, the temporal observation generated by any trajectory implying the scenario can be matched in $Pexpr(\mathcal{X})$, we need to distill the observation language of the abduction, namely the set of temporal observations generated by the trajectories in the abduction. The observation language of the abduction is recognized by a DFA called an *observation pattern*.

Definition 15 (Observation Pattern (Bertoglio et al., 2020b)) *Let $\mathcal{X}$ be a DES and $\mathbf{O}$ the domain of observations involved in the mapping table $Map(\mathcal{X})$. An* observation pattern *$\mathcal{O}^*$ of $\mathcal{X}$ is a DFA whose language is a set of sequences on $\mathbf{O}$.*

The definition of an observation pattern is general in nature. Still, meaningful observation patterns can be derived from scenario abductions. Specifically, each symbol t marking a transition $\langle a, t, a' \rangle$ in a scenario abduction $Abd(\mathcal{X}, \mathcal{S})$ is replaced with o, where $(t, o, f) \in Map(\mathcal{X})$. The

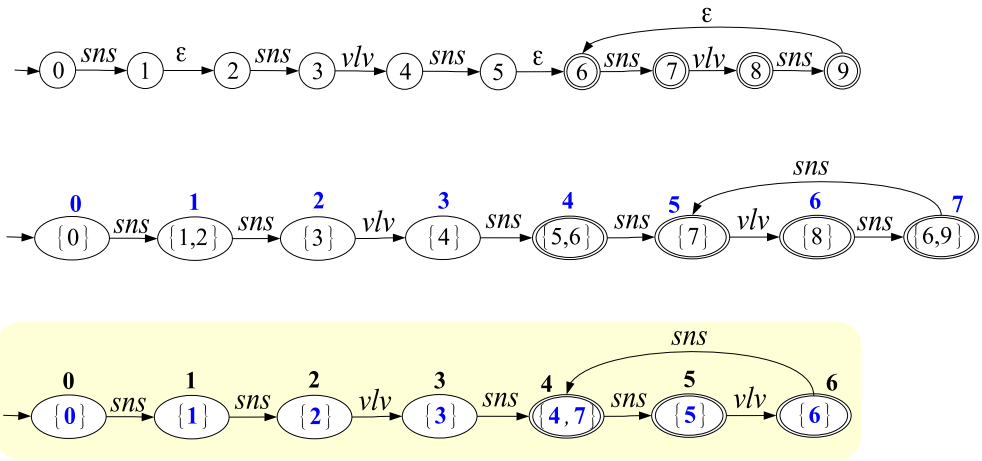


Figure 16: Generation of $Obs^*(\mathcal{S})$, the observation pattern of the behavioral scenario $\mathcal{S}$ defined in Example 18 (cf. $Abd(\mathcal{P}, \mathcal{S})$ displayed on the bottom of Figure 15).

resulting NFA is then determinized into an equivalent (possibly minimized) DFA, which is by definition the observation pattern of the scenario $\mathcal{S}$, denoted $Obs^*(\mathcal{S})$. Remarkably, the language of $Obs^*(\mathcal{S})$ is the set of temporal observations associated with the set of trajectories in the scenario abduction, with each trajectory being a mode in which the scenario $\mathcal{S}$ manifests itself in $Space(\mathcal{X})$.

Example 20 (*Observation Pattern*) Let $\mathcal{S}$ be the behavioral scenario of the DES $\mathcal{P}$ defined in Example 18, where $Abd(\mathcal{P}, \mathcal{S})$ is displayed on the bottom of Figure 15. The generation of the observation pattern of $\mathcal{S}$, namely $Obs^*(\mathcal{S})$, is outlined in Figure 16. On top of the figure is the NFA obtained from $Abd(\mathcal{P}, \mathcal{S})$ by replacing the component transition marking each arc with the corresponding (possibly empty) observation based on the mapping table $Map(\mathcal{P})$ in Figure 3. Then, the NFA is transformed into the equivalent DFA depicted on the center of the figure, where each state is identified by a subset of the states of the NFA.[9] Eventually, the DFA is minimized into the observation pattern $Obs^*(\mathcal{S})$ highlighted on the bottom of the figure (with states being renamed $\mathbf{0} \ldots \mathbf{6}$), where the equivalent states **4** and **7** of the DFA are merged into a single state, namely **4**.[10]

Once an observation pattern has been generated, a partial explainer can be extended in order to incorporate the temporal observations of the observation pattern by means of the PARTIAL EXPLAINER UPGRADE algorithm.

6.1 Algorithm PARTIAL EXPLAINER UPGRADE

To upgrade a partial explainer $Pexpr(\mathcal{X})$ based on an observation pattern $\mathcal{O}^*$, an algorithm named PARTIAL EXPLAINER UPGRADE is used (lines 1–18). Each state x^p in $Pexpr(\mathcal{X})$ is assumed to

9. This is the result of applying the standard SUBSET CONSTRUCTION determinization algorithm (Rabin & Scott, 1959), which generates a DFA that is equivalent to a given NFA.
10. Unlike NFA determinization, which may result in different equivalent DFAs (depending on the actual determinization algorithm adopted), DFA minimization always results in a single DFA, as there is just one minimal equivalent DFA. A taxonomy of minimization algorithms is presented by Watson (1995).

Algorithm 4: PARTIAL EXPLAINER UPGRADE

input : $Pexpr(\mathcal{X})$, a partial explainer of $\mathcal{X}$, having initial state x_0^p
$\mathcal{O}^*$, an observation pattern for $\mathcal{X}$, having initial state ω_o
output: $Pexpr(\mathcal{X})$ is upgraded based on $\mathcal{O}^*$

1 Insert the (unmarked) initial pattern state ω_0 into the (initially empty) labeling set $\Lambda\left(x_0^p\right)$
2 **repeat**
3 Let x^p be a state in $Pexpr(\mathcal{X})$ whose labeling set includes an unmarked pattern state ω
4 **foreach** *unmarked pattern state* $\omega \in \Lambda\left(x^p\right)$ **do**
5 **foreach** *transition* $\langle \omega, o, \omega' \rangle$ *in* $\mathcal{O}^*$ **do**
6 **if** $Pexpr(\mathcal{X})$ *includes a transition exiting* x^p *that is marked with a triple involving the observation* o **then**
7 **foreach** *transition* $\langle (x^p, x), (o, \mathcal{L}, f), (x_2^p, x_2) \rangle$ *in* $Pexpr(\mathcal{X})$ **do**
8 Insert ω' into $\Lambda\left(x_2^p\right)$, unless ω' is included in $\Lambda\left(x_2^p\right)$ already
9 **else**
10 **foreach** $x \in x^p, \langle x, t, x_2 \rangle \in Space(\mathcal{X}), (t, o, f) \in Map(\mathcal{X})$ **do**
11 Let x_2^p denote $Fspace(x_2)$, the fault space of x_2
12 **if** $Pexpr(\mathcal{X})$ *does not include the state* x_2^p **then**
13 Create a state $x_2^p = Fspace(x_2)$ in $Pexpr(\mathcal{X})$
14 Mark x_2^p with the (singleton) labeling set $\Lambda\left(x_2^p\right) = \{\omega'\}$
15 Create the transition $\langle x^p, (o, \mathcal{L}(x), f), x_2^p \rangle$ in $Pexpr(\mathcal{X})$
16 Mark ω in the labeling set $\Lambda\left(x^p\right)$
17 **until** *all the pattern states of all the labeling sets are marked*
18 Empty all the nonempty labeling sets in $Pexpr(\mathcal{X})$.

be marked with a *labeling set* (initially empty), denoted $\Lambda(x^p)$, which contains states of $\mathcal{O}^*$. This serves to synchronize $Pexpr(\mathcal{X})$ with $\mathcal{O}^*$ while avoiding duplications of $Pexpr(\mathcal{X})$ states, as well as endless loops caused by cycles in $\mathcal{O}^*$. An unmarked state ω in $\Lambda(x^p)$ means that the transitions exiting ω in $\mathcal{O}^*$ need to be synchronized with the transitions exiting x^p in $Pexpr(\mathcal{X})$. If a transition is missing in $Pexpr(\mathcal{X})$, then it is created, possibly along with its target state, a fault space, which is marked with the relevant labeling set (line 14). Once all transitions exiting ω in $\mathcal{O}^*$ have been processed, pattern state ω in $\Lambda(x^p)$ is marked (line 16). The algorithm ends when there is no further unmarked ω in any labeling set.

Example 21 (*Algorithm* PARTIAL EXPLAINER UPGRADE) Consider $Pexpr(\mathcal{P})$ in Figure 14 and the observation pattern $Obs^*(\mathcal{P})$ on the bottom of Figure 16. Illustrated in Figure 17 is the transformation of $Pexpr(\mathcal{P})$ based on $Obs^*(\mathcal{P})$ performed by the PARTIAL EXPLAINER UPGRADE algorithm. The initial configuration is depicted on the top-left of the figure, where the labeling sets marking each state (fault space) are highlighted. Initially, the only nonempty labeling set is $\Lambda(\mathbf{0}) = \{0\}$. A red state ω in a labeling set indicates that ω is the state selected in line 4. Once processed, ω becomes black and overlined, meaning that it is marked (line 16). Considering pattern state 0 in $\Lambda(\mathbf{0})$, since the only transition exiting 0 in $Obs^*(\mathcal{P})$ is $\langle 0, sns, 1 \rangle$, based on lines 6–8, pattern state 1 is inserted into $\Lambda(\mathbf{1})$, thereby leading to the configuration on top-right, where state

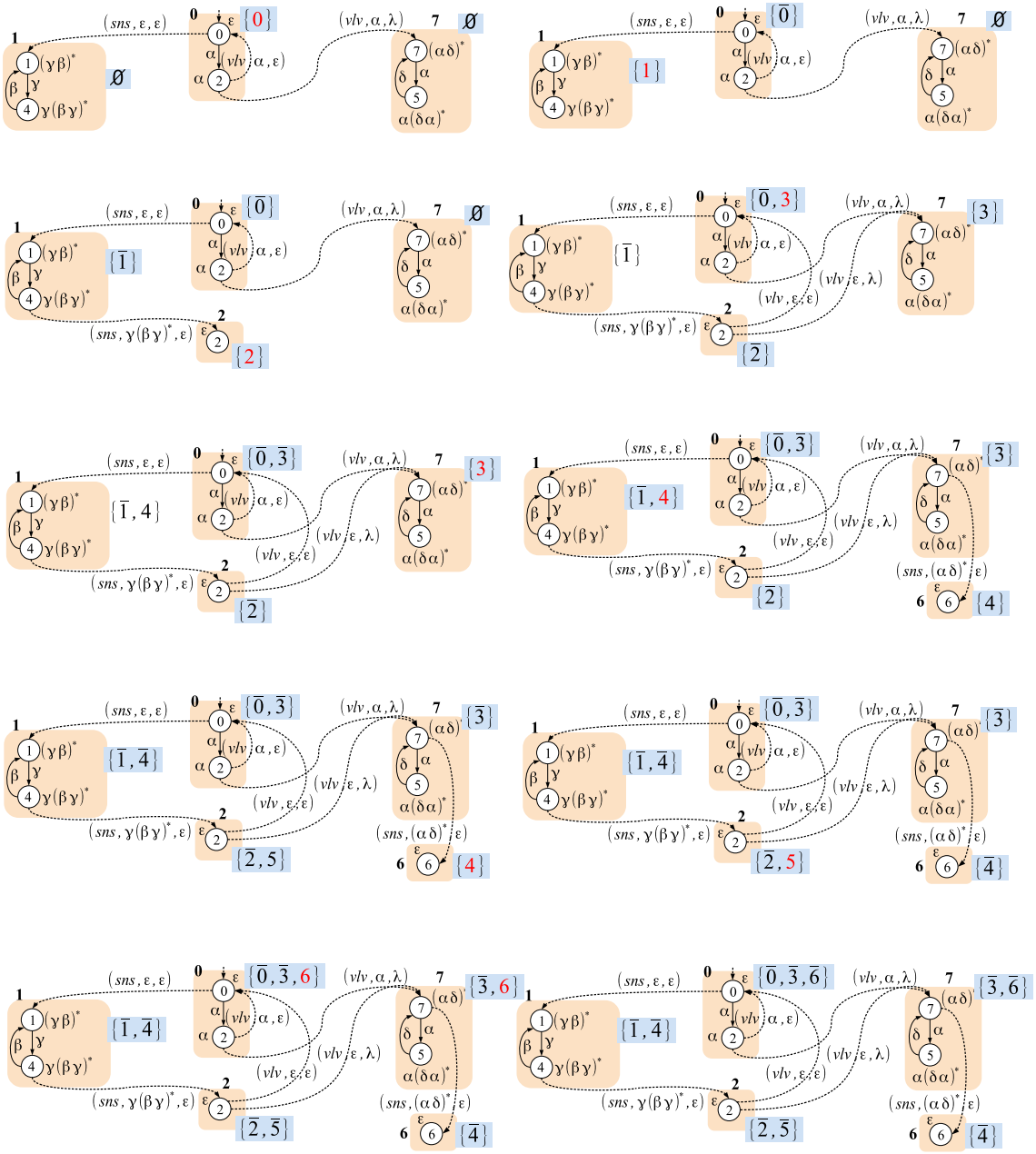


Figure 17: Tracing of the PARTIAL EXPLAINER UPGRADE algorithm applied to $Pexpr(\mathcal{P})$ in Figure 14, based on the observation pattern $Obs^*(\mathcal{S})$ in Figure 16 (bottom).

0 in $\Lambda(\mathbf{0})$ is eventually marked (line 16). At the next iteration, based on transition $\langle 1, sns, 2\rangle$ in $Obs^*(\mathcal{P})$, the processing of the unmarked pattern state 1 in $\Lambda(\mathbf{1})$ causes the creation of fault space **2** in $Pexpr(\mathcal{P})$ in lines 10–15. Eventually, considering the configuration on the bottom-left of the figure, where for the sake of simplicity we consider in parallel pattern state 6 in the labeling set of

Algorithm 5: LAZY DIAGNOSIS ENGINE

input : $Pexpr(\mathcal{X})$, a partial explainer of a DES $\mathcal{X}$
$\mathcal{T} = Trace(\mathcal{O})$, the trace of a temporal observation $\mathcal{O}$ of $\mathcal{X}$
o, a newly-received observation of $\mathcal{X}$
output: $\mathcal{T}$ is extended based on o
$Pexpr(\mathcal{X})$ is possibly upgraded
The candidate set Δ of the extended temporal observation $\mathcal{O} \cup [o]$ is generated

1 Let $\bar{\mu}$ be the last node of $\mathcal{T}$
2 **foreach** *explainer state* $x_p \in \bar{\mu}$ **do**
3 **if** *there is no transition* $\langle x_p, o, x'_p \rangle$ *in* $Pexpr(\mathcal{X})$ **then**
4 **foreach** *space state* $x \in x_p$, $\langle x, t, x' \rangle \in Space(\mathcal{X})$, $(t, o, f) \in Map(\mathcal{X})$ **do**
5 Let x'_p denote the fault space of x', namely $Fspace(x')$
6 **if** $x'_p \notin Pexpr(\mathcal{X})$ **then**
7 Create a state $x'_p = Fspace(x')$ in $Pexpr(\mathcal{X})$
8 Insert a transition $\langle x_p, (o, \mathcal{L}(x), f), x'_p \rangle$ into $Pexpr(\mathcal{X})$
9 Extend $\mathcal{T}$ by a new node μ based on the new observation o, as specified in Definition 10
10 Generate the candidate language $\Delta(\mu)$, as specified in Definition 11
11 $\Delta(\mathcal{O} \cup [o]) \leftarrow \Delta(\mu)$.

both fault spaces **0** and **7**, the only transition exiting state 6 in $Obs^*(\mathcal{P})$ is $\langle 6, sns, 4 \rangle$, which has no effect on the partial explainer, nor does it change the content of the labeling sets, thereby leading to the final configuration displayed on the bottom-right of the figure, where all pattern states of all the labeling sets are marked (line 17). Hence, once all the labeling sets have been emptied (line 18), that configuration represents the upgraded instance of $Pexpr(\mathcal{P})$, which has been extended with the new states **2** and **6** along with relevant transitions.

6.2 Algorithm LAZY DIAGNOSIS ENGINE

The GREEDY DIAGNOSIS ENGINE algorithm specified in Section 5.2 for monitoring-based diagnosis needs to be revised when the explainer is partial. Specifically, before line 1 of GREEDY DIAGNOSIS ENGINE, we need to be sure that the transition function of each state in $\mathcal{T}$ of the partial explainer is complete as far as the observation o is concerned. The new algorithm, called LAZY DIAGNOSIS ENGINE (lines 1–11) differs from GREEDY DIAGNOSIS ENGINE in two ways. First, it takes as input a *partial* explainer $Pexpr(\mathcal{X})$ rather than a complete explainer. Second, it includes a new fragment of pseudocode (lines 1–8) aimed at upgrading $Pexpr(\mathcal{X})$ based on the observation o, specifically for the possibly missing transitions involving o and exiting the state x_p in $Pexpr(\mathcal{X})$. Notice how lines 9–11 in LAZY DIAGNOSIS ENGINE parallel lines 1–3 in GREEDY DIAGNOSIS ENGINE. This way, $Pexpr(\mathcal{X})$ is possibly upgraded based on the new observation o.

Example 22 (*Algorithm* LAZY DIAGNOSIS ENGINE) Consider the partial explainer $Pexpr(\mathcal{P})$ on the left side of Figure 18. Let $\mathcal{O} = [vlv, sns, sns]$ be the temporal observation of $\mathcal{P}$ considered in Example 16 for GREEDY DIAGNOSIS ENGINE, whose execution is traced in Table 2. Outlined in

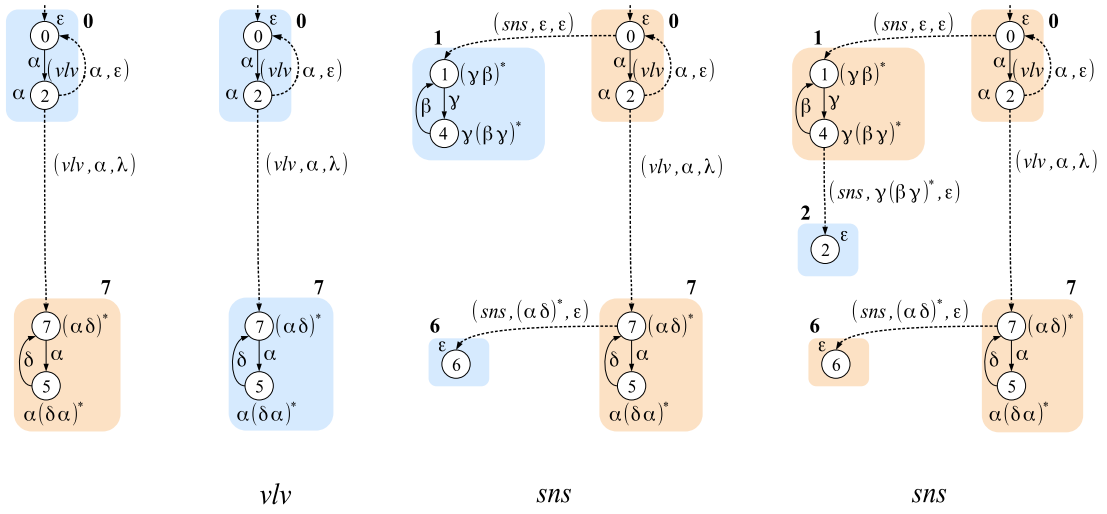


Figure 18: Upgrade actions on a partial explainer $Pexpr(\mathcal{P})$ (left) performed by the LAZY DIAGNOSIS ENGINE algorithm based on the temporal observation $\mathcal{O} = [vlv, sns, sns]$.

Figure 18 are the upgrade actions performed by LAZY DIAGNOSIS ENGINE on $Pexpr(\mathcal{P})$ when the observations in $\mathcal{O}$ are received one by one, where the explainer states in blue are those included in the last (newly-created) state of the trace (cf. Table 2). Initially, μ_0 includes just the explainer initial state **0**. Upon reception of the first observation $o = vlv$, since the transition function relevant to vlv in **0** is defined already in $Pexpr(\mathcal{P})$, no new state is generated in the latter, while the new trace state μ_1 includes the explainer states **0** and **7**. With the second observation, sns, $Pexpr(\mathcal{P})$ is extended by states **1** and **6**, along with transitions $\langle(\mathbf{0}, 0), (sns, \varepsilon, \varepsilon), (\mathbf{1}, 1)\rangle$ and $\langle(\mathbf{7}, 7), (sns, (\alpha\delta)^*, \varepsilon), (\mathbf{6}, 6)\rangle$, with **1** and **6** being the states within the new state μ_2 of the trace. With the last observation, sns, $Pexpr(\mathcal{P})$ is further extended by the new state **2**, along with transition $\langle(\mathbf{1}, 4), (sns, \gamma(\beta\gamma)^*, \varepsilon), (\mathbf{2}, 2)\rangle$. Eventually, the upgraded instance of $Pexpr(\mathcal{P})$ is displayed on the right side of Figure 18 (while the transformations of the trace are detailed in Table 2). As expected, the partial explainer has been upgraded just enough to match the temporal observation $\mathcal{O}$. From now on, each prefix of any temporal observation matching $\mathcal{O} = [vlv, sns, sns]$ will be processed efficiently because the relevant states in the partial explainer are materialized.

This concludes the presentation of sequence-oriented diagnosis of DESs, which has been implemented by three alternative techniques. In the next section, we present a sample application of sequence-oriented diagnosis.

7. Sample Application: The Labour Market Dataset

The sample application described in this section is inspired by a small real-world example, illustrated by Boselli et al. (2014), which is relevant to the Italian Labour Market dataset. Our aim is to show that in this domain, as well as in similar ones, sequence-oriented diagnosis is more convenient than set-oriented diagnosis is. According to the Italian law, whenever an employer hires or dismisses an employee, or an employment contract is modified (e.g. from part-time to full-time), a communi-

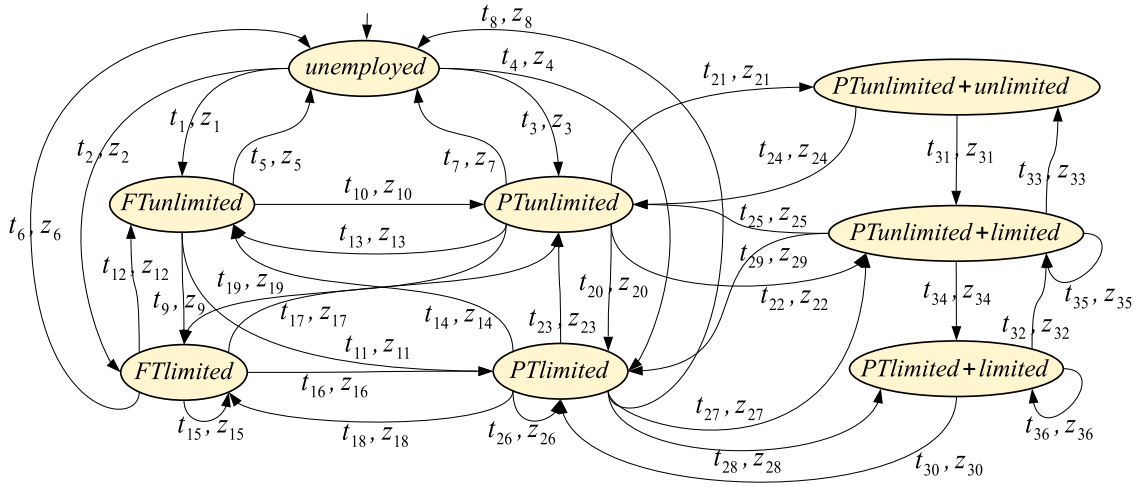


Figure 19: Labour Market DES.

cation is required to be sent to a job registry. Each communication is stored in a database as a record that specifies the parties involved (worker and employer), the date, and the kind of communicated event relevant to a job contract. There are four kinds of events:

- *Start*, which starts a new contract, this being either a limited (fixed-term) or an unlimited (unlimited-term) contract and, orthogonally, either a part-time or a full-time contract.
- *Cessation*, which ends an existing job contract.
- *Extension*, which puts off the term of a fixed-term job contract.
- *Conversion*, which changes the type of an existing job contract, from limited to unlimited or vice versa, and/or its modality, from part-time to full-time or vice versa.

The temporal sequence of communications relevant to single workers is their *career*, which is consistent only if it complies with some constraints drawn from the Italian labour law and the domain knowledge, specifically:

1. An employee cannot have further contracts if a full-time contract is active.
2. An employee cannot have more than k part-time contracts (signed by different employers); here we consider $k = 2$, the same as assumed by Boselli et al. (2014).
3. An unlimited term contract cannot be extended.
4. An extension can just postpone the deadline of an existing fixed-term contract, while it cannot change its type or modality.

All the dynamic evolutions of a career can be described by the DES shown in Figure 19, whose normal transitions are listed in Table 3. Each DES state represents a worker's possible employment (PT and FT are for part-time and full-time, respectively); in the initial state, the worker is unemployed. Each normal transition is triggered by an external input event, namely a communication,

Transition	*Observation*
$t_1 = \langle unemployed, (startFTunlimited, \emptyset), FTunlimited \rangle$	Start_FT_unlimited
$t_2 = \langle unemployed, (startFTlimited, \emptyset), FTlimited \rangle$	Start_FT_limited
$t_3 = \langle unemployed, (startPTunlimited, \emptyset), PTunlimited \rangle$	Start_PT_unlimited
$t_4 = \langle unemployed, (startPTlimited, \emptyset), PTlimited \rangle$	Start_PT_limited
$t_5 = \langle FTunlimited, (endFTunlimited, \emptyset), unemployed \rangle$	End_FT_unlimited
$t_6 = \langle FTlimited, (endtFTlimited, \emptyset), unemployed \rangle$	End_FT_limited
$t_7 = \langle PTunlimited, (endPTunlimited, \emptyset), unemployed \rangle$	End_PT_unlimited
$t_8 = \langle PTlimited, (endPTlimited, \emptyset), unemployed \rangle$	End_PT_limited
$t_9 = \langle FTunlimited, (convertUNLIMintoLIM, \{\emptyset\}), FTlimited \rangle$	From_unlimited_to_limited
$t_{10} = \langle FTunlimited, (convertFTintoPT, \{\emptyset\}), PTunlimited \rangle$	From_FT_to_PT
$t_{11} = \langle FTunlimited, (convertUNLIMintoLIMandFTintoPT, \{\emptyset\}, PTlimited \rangle$	From_FTunlimited_to_PTlimited
$t_{12} = \langle FTlimited, (convertLIMintoUNLIM, \{\emptyset\}), FTunlimited \rangle$	From_limited_to_unlimited
$t_{13} = \langle PTunlimited, (convertPTintoFT, \emptyset), FTunlimited \rangle$	From_PT_to_FT
$t_{14} = \langle PTlimited, (convertLIMintoUNLIMandPTintoFT, \emptyset), FTunlimited \rangle$	From_PTlimited_to_FTunlimited
$t_{15} = \langle FTlimited, (extend, \emptyset), FTlimited \rangle$	Extension
$t_{16} = \langle FTlimited, (convertFTintoPT, \emptyset), PTlimited \rangle$	From_FT_to_PT
$t_{17} = \langle FTlimited, (convertLIMintoUNLIMandFTintoPT, \emptyset), PTunlimited \rangle$	From_FTlimited_to_PTunlimited
$t_{18} = \langle PTlimited, (convertPTintoFT, \emptyset), FTlimited \rangle$	From_PT_to_FT
$t_{19} = \langle PTunlimited, (convertUNLIMintoLIMandPTintoFT, \emptyset), FTlimited \rangle$	From_PTunlimited_to_FTlimited
$t_{20} = \langle PTunlimited, (convertUNLIMintoLIM, \emptyset), PTlimited \rangle$	From_unlimited_to_limited
$t_{21} = \langle PTunlimited, (startPTunlimited, \emptyset), PTunlimited+unlimited \rangle$	Start_PT_unlimited
$t_{22} = \langle PTunlimited, (startPTlimited, \emptyset), PTunlimited+limited \rangle$	Start_PT_limited
$t_{23} = \langle PTlimited, (convertLIMintoUNLIM, \emptyset), PTunlimited \rangle$	From_limited_to_unlimited
$t_{24} = \langle PTunlimited+unlimited, (endPTunlimited, \emptyset), PTunlimited \rangle$	End_PT_unlimited
$t_{25} = \langle PTunlimited+limited, (endPTlimited, \emptyset), PTunlimited \rangle$	End_PT_limited
$t_{26} = \langle PTlimited, (extend, \emptyset), PTlimited \rangle$	Extension
$t_{27} = \langle PTlimited, (startPTunlimited, \emptyset), PTunlimited+limited \rangle$	Start_PT_unlimited
$t_{28} = \langle PTlimited, (startPTlimited, \emptyset), PTlimited+limited \rangle$	Start_PT_limited
$t_{29} = \langle PTunlimited+limited, (endPTunlimited, \emptyset), PTlimited \rangle$	End_PT_unlimited
$t_{30} = \langle PTlimited+limited, (endPTlimited, \emptyset), PTlimited \rangle$	End_PT_limited
$t_{31} = \langle PTunlimited+unlimited, (convertUNLIMintoLIM, \emptyset), PTunlimited+limited \rangle$	From_unlimited_to_limited
$t_{32} = \langle PTlimited+limited, (convertLIMintoUNLIM, \emptyset), PTunlimited+limited \rangle$	From_limited_to_unlimited
$t_{33} = \langle PTunlimited+limited, (convertLIMintoUNLIM, \emptyset), PTunlimited+unlimited \rangle$	From_limited_to_unlimited
$t_{34} = \langle PTunlimited+limited, (convertUNLIMintoLIM, \emptyset), PTlimited+limited \rangle$	From_unlimited_to_limited
$t_{35} = \langle PTunlimited+limited, (extend, \emptyset), PTunlimited+limited \rangle$	Extension
$t_{36} = \langle PTlimited+limited, (extend, \emptyset), PTlimited+limited \rangle$	Extension

Table 3: Details of normal transitions of the labour market DES (cf. Figure 19).

which is observable inasmuch the content of the communication is known. The (self-explaining) observation associated with each normal transition is shown in Table 3. The DES in Figure 19 is the only component of the Labour Market system, hence it is isomorphic to the space (cf. Definition 1) of the represented system. Since there is just one component, there are no output events.

The career of a worker recorded in the administrative archive is *consistent* if it does not violate the constraints listed above, in other words, if it is compliant with the DES normal behavior. For instance, given the dataset, assume to retrieve the records relevant to a worker's career according to their temporal order to check the consistency of the career. Let the first record be relevant to start a limited-term part-time contract (event *startPTlimited*, observation Start_PT_limited), the second one to convert the contract to an unlimited-term one (event *convertLIMtoUNLIM*, observation From_limited_to_unlimited), and the third one to convert the contract to a full-time one (event *convertPTtoFT*, observation From_PT_to_TF). So far the career is consistent and leads from the initial state, *unemployed*, to state *FTunlimited*, that is, the active job contract of the worker is unlimited-term full-time. Assume now that the fourth record is to start a limited-term part-time contract (event *startPTlimited*, observation Start_PT_limited), which makes the carrier inconsistent owing to the violation of the first constraint (cf. page 102). Notice that, if the temporal observation [Start_PT_limited, From_limited_to_unlimited, From_PT_to_TF, Start_PT_limited] is taken as input by the diagnosis algorithms described in this paper, then the run will stop after processing the third observation, as the fourth one cannot be processed (there is no transition exiting from state *FTunlimited* that generates observation Start_PT_limited).

If a recorded career is inconsistent, then some faults have occurred. There may be several kinds of causes for inconsistencies; however, here we consider just one kind of fault, which occurs when a record (corresponding to a communication) is missing. Hence, in an inconsistent career, one or several records are missing. We assume that, if a record is missing, we know that it is missing, as the record has actually been corrupted, so its content is unknown while its time tag is not. This is the reason for, in Figure 19, each normal transition t_i, $i \in [1\,..\,36]$, is sided by a faulty transition, z_i, that occurs if the communication relevant to the worker's transitioning from the source to the target state of z_i is not recorded in the archive. Transition z_i has the same source and target states as t_i, is not triggered by any event, and is affected by fault f_i, which means: *the content of the companion record is missing*.

All the faulty transitions are modeled as observable, the observation being 'Missing_record' for each and every of them. If a worker's career includes some corrupted records (and, hence, it is inconsistent), this modeling trick allows one to look for the contents of these records (typically, for the kind of each lost communication) that would make the career consistent. This search, which translates into the search for faults in DES diagnosis, is quite important, for instance in order to cleanse the dataset, that is, to remove all the inconsistencies by adding the content of missing records. Given an inconsistent career, however, there are several candidates, that is, several alternative ways to fix the dataset. Finding these alternatives/candidates may be hard for a human expert, while a DES diagnosis engine can compute them easily. The temporal dimension of these alternatives is of paramount importance since the records to be inserted in the dataset so as to make a worker's career consistent have to be temporally ordered.

Assume, for instance, that we need to fix the above inconsistent career, and that a single record, chronologically following observation From_PT_to_TF and preceding the observation that generated the inconsistency, is missing. Hence, we consider the temporal observation $\mathcal{O}$ = [Start_PT_limited, From_limited_to_unlimited, From_PT_to_TF, Missing_record, Start_PT_limited]. The explanation of $\mathcal{O}$ is the following:

$$Expl(\mathcal{O}) = [\Delta_0, \Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5] = [\varepsilon,\ \ \varepsilon,\ \ \varepsilon,\ \ \varepsilon,\ \ (f_5 \mid f_9 \mid f_{10} \mid f_{11}),\ \ (f_5 \mid f_{10} \mid f_{11})]\,.$$

Eventually, there are three alternatives: according to fault f_5, the missing record is inherent to an *endFTunlimited* communication; according to fault f_{10}, the missing record is inherent to a *convertFTintoPT* communication; while, according to fault f_{11}, the missing record is inherent to a *convertUNLIMintoLIMandFTintoPT* communication. One could object that the set-oriented diagnosis provides three singleton candidates, $\{f_5\}$, $\{f_{10}\}$, and $\{f_{11}\}$, which have the same expressiveness as the sequence-oriented output. However, things are different if there are two (or more) consecutive missing records, for instance when the temporal observation is $\mathcal{O}$ = [Start_PT_limited, From_limited_to_unlimited, From_PT_to_TF, Missing_record, Missing_record, Start_PT_limited], with the following explanation:

$$
\begin{aligned}
Expl(\mathcal{O}) &= [\Delta_0, \Delta_1, \Delta_2, \Delta_3, \Delta_4, \Delta_5, \Delta_6] \\
&= [\,\varepsilon,\ \ \varepsilon,\ \ \varepsilon,\ \ \varepsilon,\ \ (f_5 \mid f_9 \mid f_{10} \mid f_{11}), \\
&\quad (f_5 f_1 \mid f_5 f_2 \mid f_5 f_3 \mid f_5 f_4 \mid f_9 f_6 \mid f_9 f_{12} \mid f_9 f_{15} \mid f_9 f_{16} \mid f_9 f_{17} \mid f_{10} f_7 \mid f_{10} f_{13} \mid \\
&\quad f_{10} f_{19} \mid f_{10} f_{20} \mid f_{10} f_{21} \mid f_{10} f_{22} \mid f_{11} f_8 \mid f_{11} f_{14} \mid f_{11} f_{18} \mid f_{11} f_{23} \mid f_{11} f_{26} \mid f_{11} f_{28}), \\
&\quad \underbrace{(f_5 f_3 \mid f_5 f_4 \mid f_9 f_6 \mid f_9 f_{16} \mid f_9 f_{17} \mid f_{10} f_7 \mid f_{10} f_{20} \mid f_{11} f_8 \mid f_{11} f_{23} \mid f_{11} f_{26})}_{\Delta_6}\,].
\end{aligned}
$$

This means that, once the latest observation has been processed, there are ten candidates, each being a temporal sequence of two faults (i.e., specific missing records). By contrast, the corresponding set-oriented diagnosis output would include the candidates (sets of two faults): $\{f_3, f_5\}$, $\{f_4, f_5\}$, $\{f_6, f_9\}$, $\{f_9, f_{16}\}$, $\{f_9, f_{17}\}$, $\{f_7, f_{10}\}$, $\{f_{10}, f_{20}\}$, $\{f_8, f_{11}\}$, $\{f_{11}, f_{23}\}$, and $\{f_{11}, f_{26}\}$. Based on these data, the effort of a human expert in finding what actually happened would become more complicated, as they should trouble to establish the correct reciprocal temporal order of the pair of faults in each candidate, whereas no such effort is needed if the expert is given the correct sequence of faults in each alternative of the diagnosis output. Notice, however, that in the considered example, when there is one or two missing records, the state of the worker can not be uniquely identified based on the diagnosis candidates: in either case, once the latest observation has been processed, all we know is that the worker's state is *PTlimited* or *PTlimited+limited* or *PTunlimited+limited*. Given the same example, when three or more consecutive records are missing, the interpretation of a set-oriented candidate is even more critical. For instance, with three missing records, the interpretation of candidate $\{f_{10}, f_{13}\}$ requires the expert to find out that fault f_{10} has occurred twice, whereas the candidate expressed by the fault sequence $f_{10} f_{13} f_{10}$ is crystal clear.

The next section is devoted to the presentation of the actual implementation of the diagnosis software system, along with relevant experimental results.

8. Implementation and Experimentation

The diagnosis techniques presented in this paper have been implemented and experimented in a Master's thesis at the University of Brescia (Trerotola, 2022). Special attention was devoted to software design and coding in order for the diagnosis engines to be not only working, but also efficient, which required portions of the code to be rewritten several times.[11]

11. The software system is open source and available at `github.com/Stefano-BS/Explanatory-Diagnosis` (see the `README.md` file for usage details).

8.1 Implementation Issues

The software engines were implemented in the *C* programming language, mainly because of its flexibility and the ability to access the memory directly, even at byte level, by allocating, reallocating, and deallocating space explicitly and dynamically with the support of pointers, which is impossible in other higher-level languages, such as *Java*. This *modus operandi* allows the programmer to have full control on memory, which is essential for a software system that is supposed to construct and manipulate highly dynamical data structures under stringent time constraints. The relevant data structures generated by the software are shown to the user by exploiting the *Graphviz* package[12] along with the *dot* engine, which allow for a graphic representation of a DES, a (possibly $\mathcal{O}$-constrained) space, a (possibly partial) explainer, and a trace. Special attention was devoted to the design of the data structures, as they greatly influence the performance of the diagnosis engines, which include dynamical vectors, linked lists, and hash tables. Since regular expressions over faults are ubiquitous in sequence-oriented diagnosis of DESs, it was essential to make such expressions as comprehensible as possible to the user. To this end, a variety of simplification rules have been defined in order to mitigate the problem of redundancy.[13] The data structure representing a regular expression (a string of characters) was augmented with a set of attributes aimed at facilitating possible simplifications when that regular expression will be embedded within other regular expressions, for instance, as an alternative. For the sake of efficiency and code parallelization, the manipulation of regular expressions is required not to exploit any external buffers and, whenever possible, they have to be constructed in blocks of memory already allocated, thereby minimizing the need for continuous memory allocation and copying of memory blocks. Several portions of the code were rewritten in terms of parallel code, including the rendering of the output data structures by *Graphviz* and the (offline) generation of the explainer, which involves the construction of the regular expressions also.[14]

8.2 Experimental Results

The software system outlined above was exploited to test the viability of the diagnosis techniques presented in this paper, as well as to study relevant complexity issues empirically. Perhaps not surprisingly, the main experimental result is the actual impracticability of both *blind* and *greedy* diagnoses with a DES involving more than a few components. Hence, only *lazy* diagnosis looks to be possibly viable for real DESs. Experiments were run on a computer with CPU Intel Xeon Gold 6140 (1-7 cores available) and 128 GB of working memory.

8.2.1 DES GENERATION

In order to speed up the experimentation, the software system allows for the automatic generation of a DES based on ten parameters, namely:

1. The number m of components included in the DES.
2. The number of states included in the behavioral model of each component: this number is used as the mean value of a normal distribution from which the actual number is extracted based on the Box-Muller transform (Box & Muller, 1958).

12. *Graphviz* is a shorthand for *Graph Visualization Software*, which is available at `graphviz.org`.
13. Unfortunately, the literature does not offer a standard technique for the simplification of regular expressions.
14. This is why regular expressions are required to be manipulated without exploiting external buffers.

3. The internal connection degree of components, expressed as a percentage of $n \cdot (n-1)$, where n is the actual number of states of the component, indicating implicitly the number of transitions included in the behavioral model. Each and every component state is required to be reachable from the initial state.

4. The external connection degree of components, expressed as a percentage of $m \cdot (m-1)$, indicating implicitly the number of links between components.

5. The observability degree, indicating the percentage of observable transitions.

6. The abnormality degree, indicating the percentage of faulty transitions.

7. The number of (distinct) observable labels.

8. The number of (distinct) fault labels.

9. The number of (distinct) events.

10. The probability of generation or consumption of events.

A DES that is generated automatically based on these parameters is guaranteed to be working correctly. Moreover, results can be reproduced by extending the input parameters with the seed of the generator of the random values.

Below, the results relevant to four different experiments are presented, focusing on the processing (CPU) time. The first experiment (Section 8.2.2) is relevant to the execution of the *blind* diagnosis engine, therefore, without the support of any (total/partial) explainer. The second experiment (Section 8.2.3) is relevant to *greedy* diagnosis, focusing on both the (offline) construction of a complete explainer and the (online) execution of the *greedy* diagnosis engine, thereby exploiting an explainer. The third experiment (Section 8.2.4) is relevant to the execution of the *lazy* diagnosis engine based on a partial explainer. Eventually, in the fourth experiment (Section 8.2.5), the three diagnosis engines are compared.

8.2.2 EXPERIMENT 1: BLIND DIAGNOSIS

The first experiment is meant to show the performance of the *blind* diagnosis engine. Seven different DESs were generated with the same parameters, except for the number of components, which ranges from 9 to 15. The other parameters are: 3 states per component, 30% of internal connection in each component, 10% of external connection among components, 90% of observable transitions, 10% of faulty transitions, 2 observable labels, 1 fault label, 5 events, and 50% chance of event generation or consumption. The results of this experiment are shown in Figure 20, where each of the seven curves indicates the processing (CPU) time (in logarithmic scale) of the *blind* diagnosis engine for the computation of the candidate set upon reception of each new observation of the relevant DES. Notice how, from 10 components on, the processing time explodes after a few observations.

8.2.3 EXPERIMENT 2: GREEDY DIAGNOSIS

The second experiment is meant to show the performance of *greedy* diagnosis from two perspectives: the offline (parallel) construction of a complete explainer and the online generation of the

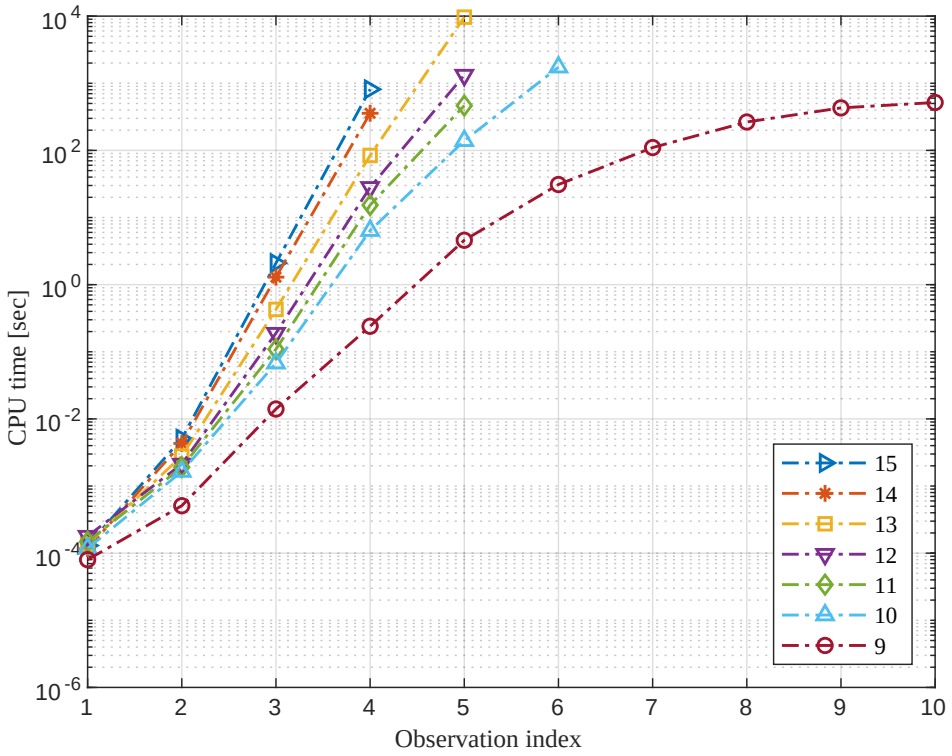


Figure 20: Processing time (in logarithmic scale) of the *blind* diagnosis engine upon reception of the i-th observation, $i \in [1 \,..\, 10]$, where the number of components ranges from 9 to 15.

candidate set by a *greedy* diagnosis engine. Six different DESs were generated with the same parameters, except for the number of components, which ranges from 1 to 6. The other parameters are: 3 states per component, 30% of internal connection in each component, 10% of external connection among components, 40% of observable transitions, 5% of faulty transitions, 2 observable labels, 2 fault labels, 10 events, and 50% chance of event generation or consumption.

Shown in Figure 21 is the processing time (in logarithmic scale) for the offline construction of a complete explainer, which is expressed both in CPU time and wall clock time. Since several states of an explainer are constructed in parallel by different cores, the CPU time corresponds to the summation of the times of each core. Notice how, for DESs including more than two components, the wall clock time for generating an explainer is about one order of magnitude less than the CPU time, which means that parallelization is effective, albeit both curves show that the construction time is exponential in the number of components.

Based on the explainers generated offline, the processing (CPU) times (in logarithmic scale) of the *greedy* diagnosis engine are shown in Figure 22, where each curve is relevant to a different DES, and each time within each curve corresponds to the computation of a diagnosis set upon reception of a new observation. Notice how, in contrast with *blind* diagnosis (cf. Figure 20), after a few observations, the processing time tends to remain substantially the same for each DES.

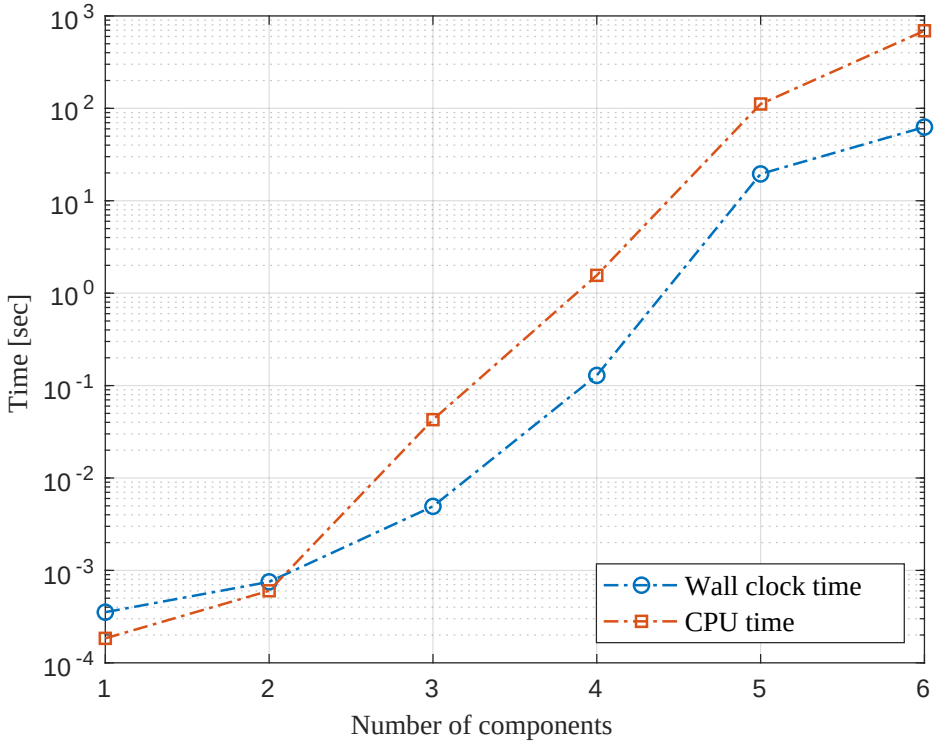


Figure 21: CPU and wall clock processing time (in logarithmic scale) for the parallel generation of an explainer in *greedy* diagnosis, where the number of components ranges from 1 to 6.

We were also interested in understanding how the observability degree may influence the processing time for the construction of an explainer. To this end, two DESs were generated based on the following common parameters: 4 components, 4 states (on average) per component, no links between components, no faulty transition, one observable label, and no generation/consumption of events. The only parameter differentiating the two DESs was the percentage of internal connections for each component, which was set to 100% and 10%, respectively. Then, the observability was set to different degrees in the two DESs, specifically, varying from 0% to 100% in the DES with 10% of internal connection, and from 50% to 100% in the other.[15] In order to possibly understand whether a different set of observable transitions might cause a variance in the processing time, three different instantiations of the observable transitions were assigned for each observability degree. Shown in Figure 23 are the results, where each curve indicates the average processing (CPU) time (in logarithmic scale) for constructing the explainer of each DES for a specific observability degree. Besides, the colored strip that accompanies each curve indicates the minimum and maximum time values for the three different instantiations of the set of observable transitions. Both curves clearly indicate that, the more observable the DES, the faster the generation of the relevant explainer. In

15. This discrepancy is grounded on the fact that, with 100% of internal connection, the processing time for constructing an explainer tends to explode with low observability.

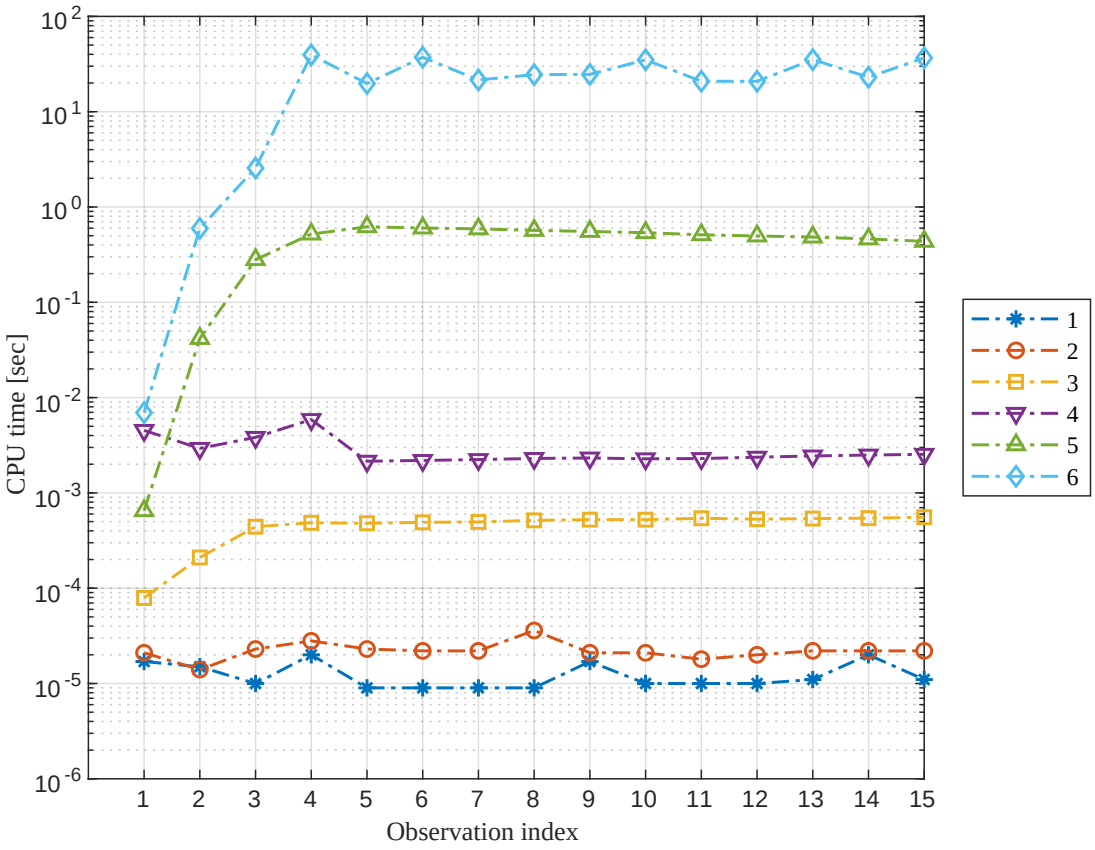


Figure 22: Processing time (in logarithmic scale) of the *greedy* diagnosis engine upon reception of the i-th observation, $i \in [1 .. 15]$, where the number of components ranges from 1 to 6.

fact, although a larger observability is bound to cause a proliferation of states (fault spaces) in the explainer, for the same reason, these states tend to be small. Each DES state within a fault space has to be marked with the relevant regular expression over the alphabet of faults: when a state entered by p transitions and exited by q transitions has to be removed, $p \cdot q$ transitions need to be generated in order to remove $p + q$ transitions (all entering/exiting transitions of the state). Consequently, notwithstanding the number of fault spaces may decrease with low observability, this reduction in number cannot balance the increased complexity in the generation of the fault spaces, specifically, the decoration of the (higher number of) internal DES states with regular expressions.

8.2.4 EXPERIMENT 3: LAZY DIAGNOSIS

This experiment is meant to show the performance of the *lazy* diagnosis engine. To this end, six different DESs were generated with the same parameters, except for the number of components, which ranges from 15 to 20. The other parameters are: 3 states per component, 30% of internal

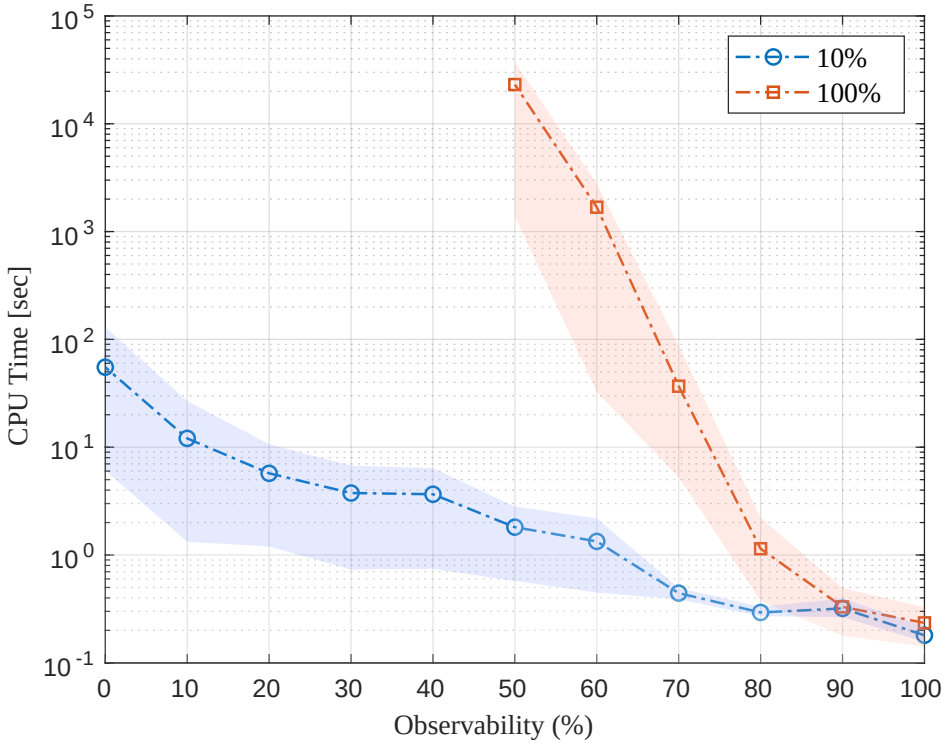


Figure 23: Influence of the observability on the construction of the explainers of two DESs, where the internal connection degree of components is 10% and 100%, respectively. Each curve indicates the average processing time of 3 different instantiations of the observable transitions with the same observability, while the accompanying colored regions are bounded by maximum and minimum time values for the three instantiations.

connection in each component, 10% of external connection among components, 80% of observable transitions, 10% of faulty transitions, 100 observable labels, 25 fault labels, 10 events, and 50% chance of event generation or consumption. As such, the DESs include several components with few transitions. The (external) connection degree among components is considerable, however.[16] The corresponding explainer incorporates many states, each including few space states, in other words, an explainer with a large number of small states, where each state is entered by several observable transitions. This allows a *lazy* diagnosis engine to materialize a few states based on each new observation. The results of this experiment are shown in Figure 24, where each of the six curves indicates the processing (CPU) time (in logarithmic scale) for the computation of a diagnosis set by a *lazy* engine upon reception of each new observation for a given DES. Remarkably, none of the DESs generated, not even the smallest (15 components), was manageable with non-lazy diagnosis, neither blind nor greedy. The results in Figure 24 enable us to draw some considerations. On the one hand, the computation time seems to remain almost constant after a certain number of observations,

16. For instance, for a DES with 20 components, there are 10% of $(20 \cdot 19) = 38$ links.

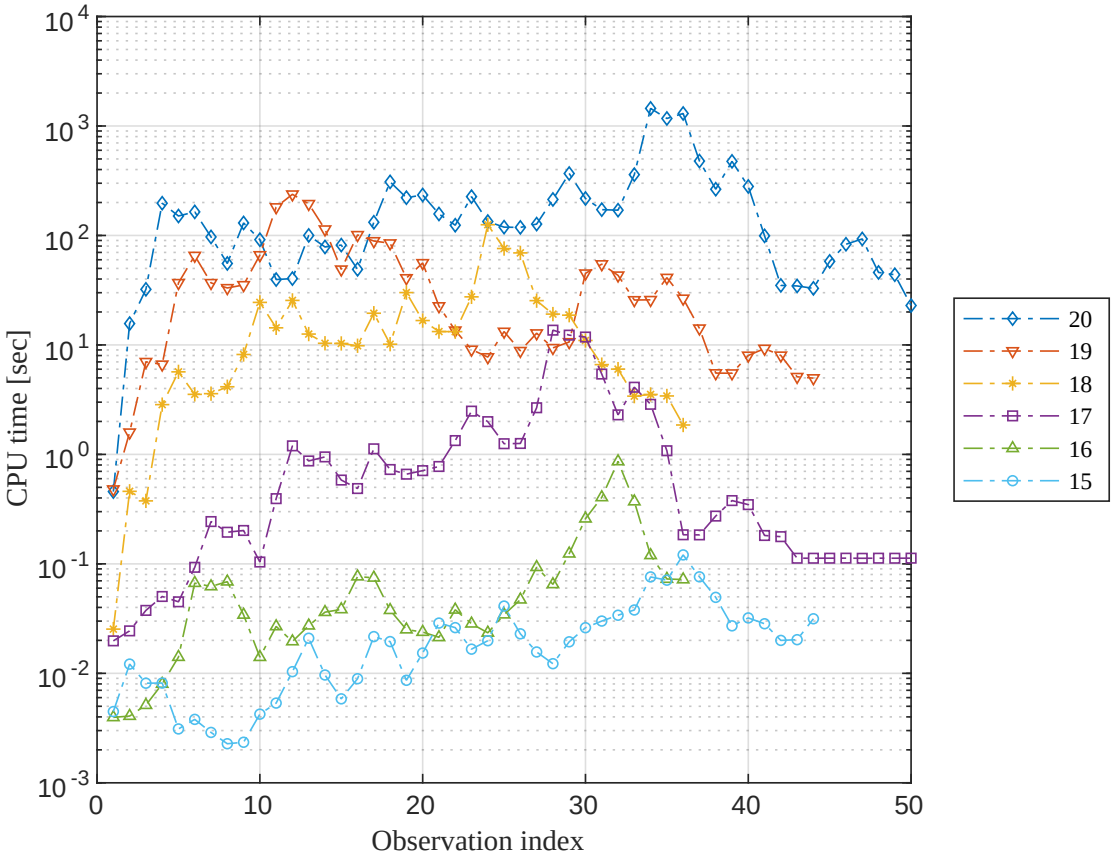


Figure 24: Processing time (in logarithmic scale) of the *lazy* diagnosis engine upon reception of the i-th observation, $i \in [1 \,..\, 50]$, where the number of components ranges from 15 to 20.

fluctuating around an average value. This suggests that, from a certain point onward, the *lazy* engine needs to materialize a constant number of new states. In the long run, the number of new states for each observation is expected to decrease, as a new observable transition is likely to enter a state that was generated already.[17] On the other hand, the computational complexity seems to increase exponentially with the number of components, a property that somehow holds for *greedy* diagnosis also (cf. Figure 22). In fact, the correlation between the number of components and the logarithm of the average processing time (arithmetic average of all the times measured) is approximately linear (angular coefficient $\simeq 0.9$).

17. This remains a conjecture, as the complete explainer could not be generated, owing to the explosion of the number of states involved.

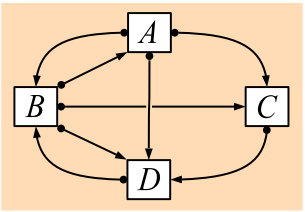


Figure 25: Structure of *icp*, the DES adopted to compare the three diagnosis engines (cf. Figure 26).

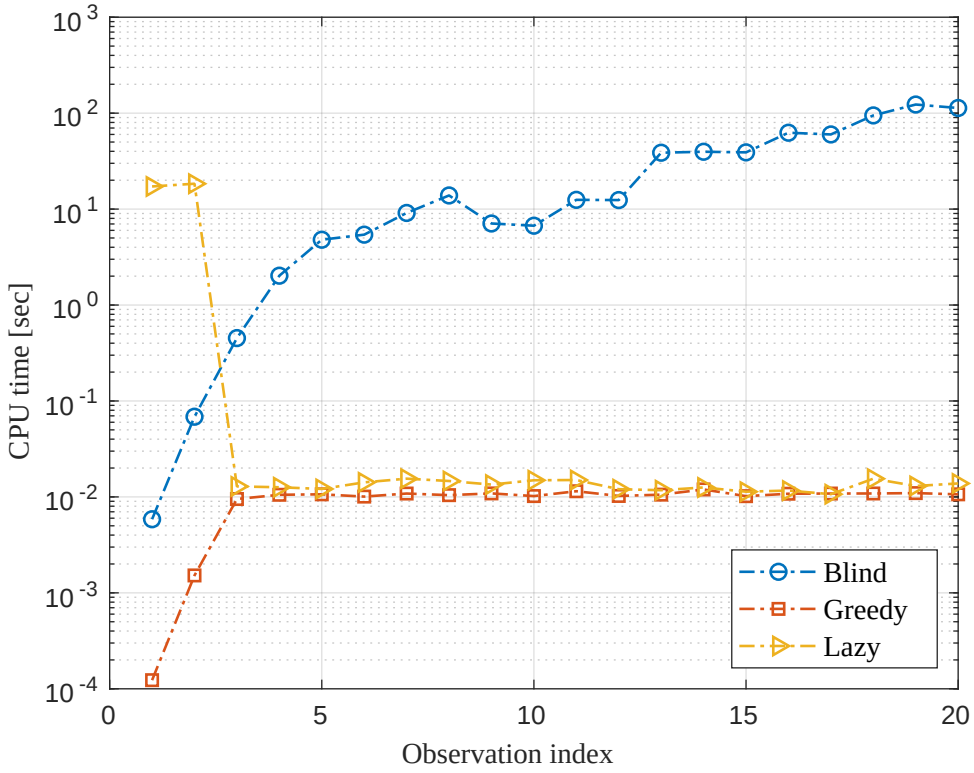


Figure 26: Processing time (in logarithmic scale) upon reception of the i-th observation of a temporal observation for each diagnosis engine applied to DES *icp* (cf. Figure 25).

8.2.5 EXPERIMENT 4: DIAGNOSIS TECHNIQUES COMPARED

In this last experiment, the three different sequence-oriented diagnosis techniques, namely *blind*, *greedy*, and *lazy*, are compared. The DES considered, called *icp*, is outlined in Figure 25.[18] It involves four components and eight links. Components include two to three states and three to six transitions each, with few events involved. Only one transition is observable. These figures cause

18. The specification of the *icp* DES is available online at `github.com/Stefano-BS/Explanatory-Diagnosis`.

the DES space to include a large number of states, which in turn may cause the explainer to include a multitude of large states, a possibly critical situation for the diagnosis task. The generation of the (complete) explainer of *icp* took 34.11 seconds. The diagnosis techniques were tested based on 20 observations in sequence (temporal observation). Results are shown in Figure 26. Unsurprisingly, the best performance is obtained with *greedy* diagnosis (when the explainer is complete). With *lazy* diagnosis, the explainer consists initially of just one state, with no exiting transitions. Remarkably, the *lazy* technique requires two observations to make the explainer complete and, thereafter, it substantially equals the performance of *greedy* diagnosis. By contrast, the performance of *blind* diagnosis, which initially is good owing to its simple implementation, nonetheless deteriorates considerably after a few observations. This suggests that, at best, *blind* diagnosis may be effective for a short temporal observation only. Worse still, a DES with poor observability is bound to make the computation of an $\mathcal{O}$-constrained space impractical, thereby invalidating *blind* diagnosis altogether.

9. Discussion

In the literature, based on the method and the amount of compiled knowledge for tracking the system trajectories that explain a given sequence of observations, two approaches to model-based diagnosis of DESs, represented as networks of communicating automata, were singled out (Basile, 2014):

1. The *compiled diagnoser*, which generates *offline* a concise model of all the possible evolutions of the DES, and then retrieves online the evolutions that explain the observations; it basically consists in the *diagnoser approach* (Sampath et al., 1995, 1996).

2. The *interpreted diagnoser*, generating *online*, in one shot, the evolutions explaining the observations, which was first proposed in the *AS approach* (Baroni, Lamperti, Pogliano, & Zanella, 1998, 1999, 2000; Lamperti & Zanella, 2002, 2004; Cerutti, Lamperti, Scaroni, Zanella, & Zanni, 2007; Lamperti & Zanella, 2011b; Lamperti et al., 2018), and taken over by other works (Pencolé & Cordier, 2005; Grastien, Cordier, & Largouët, 2005).

A further *hybrid* approach, which can be called the *lazy diagnoser approach*, has emerged in the last few years in the domain of ASs, which combines the two approaches above so as to achieve cooperation. Instead of carrying out an unrealistic total knowledge compilation upfront, as the compiled diagnoser approach does, it performs *offline* a *partial* knowledge compilation, and either retrieves *online* the evolutions that explain the temporal observation (in case they are included in the compiled knowledge) or performs model-based reasoning along with just-in-time knowledge compilation (in case they are not). This approach has been investigated for both set-oriented a posteriori diagnosis (Bertoglio, Lamperti, & Zanella, 2019b; Bertoglio, Lamperti, Zanella, & Zhao, 2019d) and set-oriented monitoring-based diagnosis (Bertoglio, Lamperti, & Zanella, 2019c; Bertoglio et al., 2019a). The investigation about the lazy diagnoser approach has been subsequently extended to sequence-oriented *a posteriori* diagnosis (Bertoglio et al., 2020d) and to sequence-oriented *monitoring-based* diagnosis (Bertoglio et al., 2020b), as in this paper.

In a set-oriented diagnosis perspective, the compiled structure is called a *dictionary* (specifically, a *symptom dictionary* for a posteriori diagnosis and a *temporal dictionary* for monitoring-based diagnosis), and is similar to the diagnoser of the diagnoser approach. In the sequence-oriented diagnosis perspective of this paper, the compiled data structure for monitoring-based diagnosis is called an *explainer*.

A dictionary resulting from partial knowledge compilation is said to be *open* since it can be progressively extended. In a set-oriented perspective, the partial compilation of the knowledge is usually intended to build offline a *prefix* of the whole dictionary (Bertoglio et al., 2019b, 2019d, 2019c, 2019a), this being an open dictionary that includes only the states up to a given distance from the initial state of the dictionary (in terms of observable transitions). Later, this prefix can be extended based on significant scenarios, where each scenario is a concise way to specify a collection of evolutions of the DES that are considered critical in the application domain. Incorporating these scenarios in the open dictionary allows the diagnosis engine to work efficiently when the DES performs a trajectory within one such scenario.

The temporal dictionary in the set-oriented AS approach resembles the compiled knowledge structure named *diagnoser* in the diagnoser approach (Sampath et al., 1995, 1996), and the explainer is the sequence-oriented counterpart of the temporal dictionary. Still, compared to the diagnoser approach, the dictionary considers all the trajectories rather than only those ending with an observable transition. Most of all, the method to generate the dictionary is different, as it relies on the operation of FA determinization, being the dictionary a DFA over the alphabet of the observations, the same as the diagnoser of the diagnoser approach.

Several further differences can be singled out between the diagnoser approach (which is set-oriented) and the AS approach, including the current paper. First of all, the diagnoser approach and the AS approach make different modeling assumptions. The diagnoser approach, along with other works that adopt the same DES model, assumes that every faulty component transition is unobservable, on the ground that the diagnosis of faults would be trivial if they were observable. The above justification is objected by the AS approach on the ground that the observable event relevant to a faulty transition can be shared with (several) normal and/or faulty transitions, which is bound to create uncertainty. Hence, in this paper (the same as in the whole AS approach) faulty transitions can be possibly observable.

Moreover, the diagnoser approach (Sampath et al., 1995) does not cope with a possibly infinite number of possibly infinite trajectories that entail a temporal observation. In fact, it assumes that both the language of the transitions of the DES and the language of the observable events of the DES are live, while both such assumptions are missing in the AS approach. Consequently, while according to the diagnoser approach there is no unobservable behavioral cycle and, hence, no cycle of faults, both such cycles are allowed in this paper, where the regular expressions relevant to the occurrence of faults are able to represent an unbounded number of iterations.

As to the symptom dictionary on the one hand, and the diagnoser of the diagnoser approach on the other, both are FAs that recognize the language of all the temporal observations of the DES. The language of temporal observations in the diagnoser approach is prefix closed, however, whereas the language of temporal observations in a posteriori diagnosis is not, as a temporal observation is relevant to a trajectory of the DES ending in a quiescent state. The notion of a quiescent state is specific to a posteriori diagnosis of ASs and does not apply to the diagnoser approach, as the relevant models, which support synchronous communication between components, do not include any links. Since the constraint that a trajectory of the DES ends in a quiescent state is missing when monitoring-based diagnosis comes into play, it may be tempting to speculate that the temporal dictionary and the diagnoser of the diagnoser approach are basically the same notion, which is in fact untrue. To clarify, consider a path $\wp$, both in the diagnoser and in the temporal dictionary, and the temporal observation $\mathcal{O}$ associated with $\wp$. In the diagnoser, $\wp$ represents all the DES trajectories that meet two conditions: (a) their projection on the set of observable events is $\mathcal{O}$, and (b) their

final transition is observable. In the temporal dictionary, $\wp$ represents all the DES trajectories whose projection on the set of observable events is $\mathcal{O}$, where the final component transition can be possibly unobservable. Consequently, given the same temporal observation $\mathcal{O}$, generally speaking, the set of trajectories, from which the output candidates are drawn, differs in the diagnoser approach with respect to the AS approach, including this paper.

To conclude, we recall that the sequence-oriented diagnosis perspective adopted in this paper was introduced by the authors in recent works under the name of *temporal-oriented* diagnosis (Bertoglio et al., 2020c, 2020a, 2020b, 2020d). Specifically, both notions of a (behavioral) scenario (Definition 13) and an observation pattern (Definition 15) can be found in two works (Bertoglio et al., 2020b, 2020d). Two other papers deal with monitoring-based diagnosis (Bertoglio et al., 2020c, 2020b), the former exploiting total knowledge compilation, as in greedy diagnosis, and the latter exploiting also partial knowledge compilation (called *smart knowledge compilation*), as in lazy diagnosis. The notions of a fault space (Definition 8), an explainer (Definition 9), and a (monitoring) trace (Definition 10) can be found in both of them. The notions of a partial explainer (Definition 12) and a scenario abduction (Definition 14) are defined in the latter. However, in either case and unlike in the current paper, every time a new observation is perceived, a *backward pruning* technique is applied to guarantee that all the candidates relevant to any prefix of the temporal observation received so far are consistent with the whole temporal observation received so far. Albeit it can be offered on demand by the software system implemented, this service is ignored in this paper on the ground that it may be not interesting from a practitioner's point of view.

9.1 Hints on Complexity

We first provide some hints about the size of the compiled structures handled by the compiled diagnoser approach (Sampath et al., 1995, 1996) and the approach described in this paper, namely the AS approach for short. Later, we analyze the time complexity of the construction of a single fault space and of the complete explainer. Further remarks about the complexity of the three different techniques implementing sequence-oriented monitoring-based diagnosis of DESs can be found in the Appendix, which adopts the notation introduced in the current section.

In the compiled diagnoser approach, the construction of the diagnoser is based on the previous construction of the whole system model, which essentially corresponds to the Space in this paper (Definition 1). The upper bound of the number of states in the system model of the compiled diagnoser approach is $(n_{\max})^m$, where $n_{\max}$ is the maximum number of states per component and m is the number of components (as in Section 8.2.1). The upper bound of the number of states in the Space is $(n_{\max})^m (p_{\max})^l$, where $p_{\max}$ is the maximum number of configurations per link, namely the maximum number of distinct sequences of events that can be stored in a link[19], and l is the number of links. Since $(n_{\max})^m (p_{\max})^l \leq (max\ \{n_{\max}, p_{\max}\})^{m+l}$, the upper bound of the number of Space states is analogous to the upper bound of the number of diagnoser states in the compiled diagnoser approach.[20] The number of DES states is exponential in both approaches, which is critical for the compiled diagnoser approach since, as mentioned above, it has to build the system model explicitly, which is not required by the AS approach. Let s and t denote the number of states and transitions,

19. According to the specification of a DES presented in this paper, the capacity of a link is implicitly 1, $p_{\max} = e_{\max} + 1$, where $e_{\max}$ is the maximum number of distinct events each of which can be saved in the link; the increment by one takes into account the configuration in which the link is empty.

20. As discussed by Lamperti and Zanella (2013), a link can indeed be regarded as a component that has a synchronous interaction with both the components that feed it and those that extract events from it.

respectively, either in the system model of the compiled diagnoser approach or in the Space of the AS approach. Notice that the upper bound of t is $s(m)t_{\text{c\_out}}$, where $t_{\text{c\_out}}$ is the maximum number of exiting transitions per state in component models.

Coming to the size of the compiled structures, the upper bound of the number of states (fault spaces) in the explainer (Definition 9) is the number s_o of Space states that have an entering observable transition[21]. Roughly, we can estimate s_o as follows. The probability that a Space state is not entered by any observable transition is the product of the probability that each of its entering transition is unobservable. The probability that a Space transition is unobservable is $(1 - (t_o/t_{\text{tot}}))$, where (t_o/t_{tot}) is the observability degree (cf. Section 8.2.1), namely the ratio between the cumulative number of observable transitions in all the component models (t_o) and the total number of transitions in all the component models (t_{tot}). The average number of transitions entering a Space state is (t/s), which is independent of the observability and abnormality of the DES. Hence, the probability that a Space state is not entered by any observable transition is $(1 - (t_o/t_{\text{tot}}))^{t/s}$. Notice that, according to this estimate (and fulfilling our expectations), the probability that a Space state is not entered by any observable transition decreases for increasing values of the observability degree (it becomes zero when the observability degree is 1, that is, when all transitions are observable) as well as for increasing values of the ratio (t/s), assuming that this ratio is no less than 1, as is for any DES exhibiting some cyclic behavior.

Dually, the probability that a Space state is entered by an observable transition is $(1 - (1 - (t_o/t_{\text{tot}}))^{t/s})$, which allows for estimating s_o as $s\left(1 - (1 - (t_o/t_{\text{tot}}))^{t/s}\right)$. Hence, the value of s_o, which is the upper bound of the number of states in the explainer, becomes closer to s (whose upper bound is exponential in the sum of the number of components and links) the higher the value of the observability degree. This estimate formally supports the allegation that the number of states in the explainer increases with the observability degree (Section 8.2.3), while the size of each state (the number of Space states in each fault space) decreases. In fact, the probability that a Space state is entered by an unobservable transition is $(1 - (t_o/t_{\text{tot}})^{t/s})$ where $(t_o/t_{\text{tot}})^{t/s}$ is the probability that all the entering transitions of a state are observable. The upper bound of the number of Space states in a fault space is $(s_u + 1)$, that is, the number of Space states that are entered by an unobservable transition, s_u, incremented by one in order to account for the initial state of the fault space, which maybe is not entered by any unobservable transition. The above rough estimate of this upper bound equals $s\left(1 - (t_o/t_{\text{tot}})^{t/s}\right) + 1$, which is a value that decreases when the value of the observability degree increases.

In the compiled diagnoser approach, each state in the diagnoser includes a (non empty) subset of the states of the system model that are entered by an observable transition whose observable label is the same for all the states in the subset. Within each diagnoser state, each state of the system model is associated with a (possibly empty) set of faults. Hence, the upper bound of the number of states in the diagnoser of the compiled diagnoser approach is

$$\sum_{o \in E_o} \sum_{i=1}^{s_o^o} \binom{s_o^o}{i} \left(2^f\right)^i$$

21. This value has to be incremented by one in case the initial Space state has no entering observable transition, as the explainer always includes the fault space inherent to the initial state of the Space.

where E_o is the set of observable events of the DES, s_o^o is the cardinality of the subset containing all the DES states that have an entering transition labeled by the same observable event o, and f is the cardinality of the set of faults relevant to the DES. Notice that $s_o^o \leq s_o$ (if $|E_o| = 1$, then $s_o^o = s_o$). The above expression is less than $\sum_{o \in E_o} \left(2^f + 1\right)^{s_o^o}$, which, in turn, is less than or equal to $|E_o| \left(2^f + 1\right)^{s_{\text{o\_max}}^o}$, where $s_{\text{o\_max}}^o$ is the maximum value of s_o^o, for the observable event o ranging over all the observable events. By comparing the upper bounds of the number of states in the two compiled structures, the explainer (s_o) and the diagnoser ($|E_o| \left(2^f + 1\right)^{s_{\text{o\_max}}^o}$, with $s_{\text{o\_max}}^o \leq s_o$), we notice that: (1) unlike the diagnoser, the maximum number of states in the explainer does not depend on the number of faults, and (2) the number of states in the explainer (which is an NFA) is likely to be considerably smaller than the number of states in the diagnoser (which is a DFA).

Unfortunately, as confirmed by experimental evidence (cf. Section 8.2.3), the construction of a complete explainer, as expected by GREEDY DIAGNOSIS ENGINE, is all the same out of the question, first because the upper bound of the number of states in the explainer is still exponential in the sum of the number of components and links and, second, because the construction of each state (fault space) is based on the invocation of a generalized version of the state elimination algorithm (Brzozowski & McCluskey, 1963), namely Algorithm 2. In the construction of a complete explainer, the higher cost is paid for the generation of the regular expressions. Figure 23 shows that, by increasing the observability degree (which, as explained above, is bound to increase the number of states in the explainer), the CPU time required for the construction of the complete explainer decreases, as each explainer state includes a smaller number of Space states for which to compute a regular expression.

The original version of the algorithm by Brzozowski and McCluskey computes the regular expression representing the regular language defined by a given FA (both NFAs, including ε-NFA, and DFAs are allowed); the (pessimistic) time complexity of this algorithm, as analyzed by Hopcroft, Motwani & Ullman (2006), is $O\left(n^3 4^n\right)$, where n is the number of states of the FA to be converted. The exponential blow-up when moving from an FA to regular expressions is inherent (Gruber & Holzerl, 2014), that is, independent of the conversion method, as the lower bound of the operation is itself exponential in n. The generalized version of the state elimination algorithm proposed in this paper, which computes multiple regular expressions, one for each state of the unobservable subspace that has to be turned into a fault space, does not alter the complexity of the original version. In fact, the modified version initializes an FA, namely A, this being the unobservable subspace that is rooted in a given Space state, then it adds a new initial state and an ε-transition (from the new initial state to the old one), as well as a pair (new final state, new ε-transition) for each state in the FA (with the exception of the added initial state). The FA obtained, namely A', is then processed by applying the same rules as in the original algorithm, thus progressively reducing the number of states until only the above added (initial and final) states are left. The effort of the modified version, that produces one regular expression for each final state of A', is not harder than that needed by the original version when producing as output the alternative of all such expressions. In fact, to this end, the original algorithm could process an automaton obtained by transforming all the final states in A' into non-final, by adding one final state, and by adding, for each former final state, an ε-transition that leads from this state to the new final one. Since the new automaton A'' has a number of states that equals that of A' incremented by one, where the number of states in A' is $\Theta(n)$, n being the number of states in A, the asymptotic complexity of the two versions of the algorithm is the same.

In the next few lines we delve further into the size of an explainer. Based on the previous analysis, a (pessimistic) upper bound of the total number of Space states within all the fault spaces in the (complete) explainer is $s_o(s_u + 1)$. The upper bound of the number of transitions within each fault space is the number t_u of unobservable transitions in the Space. Hence, a (pessimistic) upper bound of the total number of (unobservable) Space transitions within all the fault spaces in the (complete) explainer is $s_o t_u$. Each fault space has at most $(t - t_u)$ exiting transitions, as the exiting transitions of a fault space are necessarily observable. Hence, a (pessimistic) upper bound for the number of (external) transitions in the explainer is $s_o(t - t_u)$. Altogether, the size of the complete explainer, considered as the number of Space states and Space transitions contained in it, that is, $s_o(s_u + 1)$ and $s_o t$, respectively, is not more than quadratic in the size of the Space.

Algorithm FAULT SPACE is called both by GREEDY DIAGNOSIS ENGINE (to build each fault space of the complete explainer) and LAZY DIAGNOSIS ENGINE (to build each fault space of a partial explainer). This algorithm is given a Space state and has to build the fault space relevant to it. First, it creates the unobservable subspace of the given Space state, then it produces the regular expressions relevant to its states. The upper bound of the number of transitions within a fault space is t_u, and the number of Space states within a fault space[22] cannot exceed $(t_u + 1)$ since each state, possibly excluding the one at the 'entrance', has an entering internal transition; hence, the effort to create the unobservable subspace is $O(t_u)$, while the effort to produce the regular expressions is $O\left((s_u + 1)^3 4^{(s_u+1)}\right)$. The cost of FAULT SPACE is thus $O\left(t_u + (s_u + 1)^3 4^{(s_u+1)}\right)$.

We conclude this section by analyzing the time complexity of the construction of the complete explainer. This consists in at most s_o calls of algorithm FAULT SPACE, which altogether take a time $O\left(s_o(t_u + (s_u + 1)^3 4^{(s_u+1)})\right)$, and in the creation of the transitions between fault spaces, which takes a time $O(s_o(t - t_u))$. Hence, the overall asymptotic complexity is expected to be $O\left(s_o(t + (s_u + 1)^3 4^{(s_u+1)})\right)$.

10. Related Work

This section considers a collection of works in the literature that bear some resemblance with (or may be incorporated in) the techniques presented in this paper. The collection is roughly divided into four sections: papers on set-oriented diagnosis that focus on DESs and/or exploit some sort of compiled knowledge (Section 10.1); an approach amenable to sequence-oriented diagnosis of DESs (Section 10.2); a variety of papers dealing with diagnosis of DESs specified by a temporal logical formula (Section 10.3); and an alternative notion of diagnosis output named *health state* (Section 10.4).

10.1 Diagnosis of DESs and/or Exploitation of Compiled Knowledge

The class of diagnosed ASs has been extended by the notion of a *deep* DES (DDES) (Lamperti et al., 2020). The structure of a DDES is a tree where each node is an augmented AS, called an *active unit* (AU). When the components within an AU collectively perform a sequence of transitions matching a given regular language, an *emerging event* is created and sent to its parent AU, thus

22. Above, the upper bound of the number of Space states within a fault space is estimated as $(s_u + 1)$, whereas it is estimated as $(t_u + 1)$ here. Yet, the two estimates are consistent since $(t_u + 1) \geq (s_u + 1)$. In fact, the number of Space states that have an unobservable entering transition, s_u, is not greater than the number of unobservable transitions in the Space, t_u.

affecting its behavior. Each AU can send emerging events only to its parent AU. An emergent event is generated when a *behavioral pattern* is recognized, a notion that has some analogy with a behavioral scenario for ASs: both are regular languages defined over the alphabet of the component transitions. Their purposes are different, however: patterns for the generation of emergent events aim to support behavior stratification in a hierarchy of AUs, while scenarios aim to extend a data structure embodying a partial knowledge about the behavior of a given (flat) AS. An efficient online a posteriori diagnosis technique for DDESs is proposed, which does not require the computation of the global trajectories of the DDES, as they are inessential to find the global candidates. Instead, it generates the constrained behavior of each AU rather than the constrained behavior of the whole system. The behavior is constrained not only by the temporal observation of the AU, but also by the emergent events generated by its child AUs. Although requiring a single bottom-up traversing of the DDES tree structure, this technique guarantees the soundness and completeness of the candidates computed. Still, a main difference with respect to the current work is that no knowledge compilation is adopted for behavior reconstruction, with the exception of the preprocessing of emergent events.

Knowledge compilation is addressed in several papers coping with model-based diagnosis, such as the work by Console et. al. (2003), where the (tabular) temporal patterns relevant to the observations of a dynamical system are compiled into *temporal decision trees*. These trees, each relevant to a specific sensor, encompass both qualitative and quantitative temporal constraints relevant to the observations corresponding to distinct situations, where the domain of the observations is a finite set of discrete values corresponding to sensor readings or to pieces of information drawn from these measurements. The values observed are assumed to be acquired at discrete times. Once a fault is detected without being isolated, the compiled knowledge is exploited to perform fault isolation so as to carry out a recovery action before a given hard deadline, which is set by accounting for safety and integrity of the physical system. Although a temporal decision tree is a means to represent a temporal pattern, the same as a scenario is in the current approach, its representation, alphabet, semantics, and purpose are quite different. Nodes of a temporal decision tree are endowed with time labels, whereas a scenario is usually represented as a DFA. The alphabet of (the arcs in) a temporal decision tree is the set of observable values, whereas the alphabet of (the arcs in) a scenario is a set of transitions. A temporal decision tree is based on the assumption that observations are performed at regular times, whereas the current approach assumes to register each value change whenever it occurs. A temporal decision tree represents a finite (quantitative) temporal horizon (the deadline within which the recovery action has to be carried out), whereas a scenario is not constrained by any time limit and contains information about the temporal ordering of events only, without any explicit time length. Besides, the purpose of a temporal decision tree is to discriminate a fault, whereas the purpose of a scenario is to extend the diagnostic knowledge within a compiled knowledge structure.

Knowledge compilation is exploited not only for dynamical systems, but also in model-based computation of minimal cardinality candidates of static systems. The system description is compiled into a Decomposable Negation Normal Form (DNNF) by Darwiche (2001) and into a Binary Decision Diagram (BDD) by Torasso and Torta (2006). In either case, minimal cardinality diagnoses are computed in a time that is polynomial in the size of the compiled model, with the latter possibly growing exponentially and becoming a bottleneck. To cope with this explosion, Siddiqi and Huang (2007) suggests to adopt a structural abstraction that is based on *hierarchical diagnosis*: the system can be decomposed into maximal self-contained subsystems, each being regarded as a single component (black box) at a higher abstraction level. If a subsystem is identified as faulty in the top abstraction level, then it can be recursively processed separately, as a subsystem may hierar-

chically contain further subsystems. In the quoted approach, the explosion of the size of compiled knowledge is tackled by means of hierarchical diagnosis, whereas in this paper it is managed by exploiting partial knowledge compilation.

The weak fault models adopted by Siddiqi and Huang (2007) are extended with probabilities by the same authors (2011), where the task is no longer the computation of minimal cardinality candidates, instead it is *sequential diagnosis*. Once the candidates relevant to an observation have been computed, sequential diagnosis isolates the faults by exploiting the evidence provided by additional measurements. The diagnostic cost is the number of the measurements that are needed until all the actual faults are identified. The measurements and the order in which they have to be taken are selected heuristically so as to achieve low costs. The method proposed by Siddiqi and Huang (2011) systematically modifies the structure of a system to reduce the size of its structural abstraction. Attention is focused on the components that are not part of any subsystem and hence cannot be abstracted away in hierarchical diagnosis. The idea is to create a sufficient number of clones of every component so that the original component and each of its clones become part of some subsystems and thus can participate in this (purely structural) abstraction. A single health variable and failure probability is adopted for the entire subsystem, thus scaling up the sequential diagnosis approach to larger systems. The case studies considered by Siddiqi and Huang (2007, 2011) are combinational circuits, hence static systems, although the authors claim that their sequential diagnosis framework applies to other types of systems as well. At any rate, the tasks considered in those papers are different from those in the current paper, the same as the fault models. In particular, no failure probability is processed by the AS approach.

The same task and setting by Siddiqi and Huang (2007) can be found in another work (Metodi, Stern, Kalech, & Codish, 2014), which copes with the computation of minimal cardinality candidates of combinational circuits represented through weak fault models (without any probability) that are expressed by propositional formulae. The task processes a single observation of inputs and outputs of the whole system, that is, no probes relevant to specific components (each of which is assumed to have a single output) are taken into account. The rationale is to apply a general-purpose (SAT) solver to compute the solutions, rather than dedicated diagnostic algorithms. First, some (low-order polynomial) domain-dependent preprocessing techniques are applied to build a constraint model that exploits unique substructures in the system. Second, this constraint model is compiled into a corresponding Conjunctive Normal Form (CNF) formula by a constraint compiler proposed in two works (Metodi & Codish, 2012; Metodi, Codish, & Stuckey, 2013), which is able to simplify the constraints and to generate a succinct CNF encoding with the aim of reducing the runtime of the SAT solver. Last, a structural abstraction, inspired by the idea by Siddiqi and Huang (2007) of identifying components that dominate others, is exploited to decompose the diagnosis problem, so as the SAT solver is only used to find top-level candidates, which are later expanded to all minimal cardinality candidates by a poly-time algorithm. Experimental evidence relevant to two standard benchmarks shows that the runtime of the approach presented by Metodi et.al. (2014) outperforms, often by orders of magnitude, those of the existing approaches to the (exact or approximate) search for either a single minimal cardinality candidate or all minimal cardinality candidates. The comparison browsed, among others, the algorithms presented in several papers (Siddiqi & Huang, 2007; Feldman, Provan, & van Gemund, 2010a; Siddiqi & Huang, 2011). Although performing knowledge compilation, the approach by Metodi et. al. (2014) is quite different from the approach in the current paper. It computes minimal cardinality candidates, while our approach generates all the candidates. It deals with static systems (combinational circuits) whereas

our paper addresses DESs, which are dynamical systems. The knowledge compilation proposed by Metodi et. al. (2014) is meant to produce an encoding that can be processed efficiently by a general-purpose solver, whereas the compiled knowledge produced by our approach is meant to reduce the runtime of dedicated diagnostic algorithms.

In our paper, sequence-oriented diagnosis of DESs refers to the relative order in which faulty transitions occur within a candidate, without any explicit time tags being considered, as the component models are completely untimed. Some contributions in the literature, however, such as the work by Hashtrudi Zad et. al. (2005), add time information to the DES models, where the clock tick is an extra input signal.

Time information is considered also by Feng and Grastien (2020), where a DES is modeled as a set of timed automata (Alur & Dill, 1994), which rely on clocks whose values increase at constant rate. Following the work by Tripakis (2002), explicit quantitative time constraints between event occurrences are considered. Unlike our approach, in the work by Feng and Grastien (2020) faults are always unobservable. Moreover, the task of a posteriori diagnosis is addressed, whereas our paper copes with monitoring-based diagnosis. Finally, in the work by Feng and Grastien (2020) a model checker is exploited in order to solve a diagnosis problem, rather than dedicated algorithms.

Explicit time information is considered by Pencolé et. al. (2021) also, where a DES is modeled as a labeled time Petri net, which is assumed to be safe.[23] Transitions are endowed with priorities. The transition labels represent the events in the system, and only a subset of them are observable. In the interest of realism, it is assumed that no unbounded sequence of transitions can occur in a finite amount of time, whereas all the (possibly unbounded) sequences of transitions are considered in our paper. However, the choice by the AS approach of concisely representing and dealing also with infinite sequences of transitions is not unrealistic; realism is preserved by the given temporal observation, which includes all and only the observable events that have been perceived in a finite implicit time interval. Since a temporal observation is necessarily finite, each relevant trajectory consists of a finite number of transitions. If a candidate is relevant to a trajectory that includes some cyclic subsequences of unobservable transitions, this does not mean that the cycle has been iterated an infinite number of times, it just denotes our ignorance about the actual number of iterations. A regular expression that makes it clear that a cycle of faults may have been followed several times provides *per se* a remarkable piece of diagnostic information.

In the work by Pencolé et. al. (2021), the diagnosis problem, which is turned into a reachability problem over labeled time Petri nets that can be solved by a model checker, aims to identify whether a given combination of unobservable untimed events has occurred. This combination, which is defined as an (untimed) labeled Petri net, is the same as a supervision pattern (Jéron, Marchand, Pinchinat, & Cordier, 2006). However, unlike in the work by Jéron et. al. (2006), the diagnosis output provided by Pencolé et. al. (2021) is threefold: it clarifies whether the pattern is matched by all the DES evolutions that can generate the timed sequence of observations, or only by some of them, or by none of them.

The notion of a fault in our paper is quite simple and yet powerful, namely a tag assigned to a component transition. The diagnosis output is the language over the alphabet of these tags that complies with the temporal observation. Although a fault is usually seen as the cause of a behavioral malfunction, this is not the only admissible point of view. In fact, a fault is possibly any state transition (either observable or not) we are interested in, and the motivations for our interest in a

23. The number of tokens in each place is either 0 or 1, in any reachable marking.

specific state transition may vary. Typically, we may be interested in the occurrence of any state change that provides some hints about how to maintain, repair, or reconfigure the DES, or that is reckoned to be dangerous. We may also be interested in state changes that mark some improvement in a healing system, however. The diagnosis task described in this paper performs (either explicitly or implicitly) a reconstruction of the DES behavior based on a temporal observation, and the output (a fault sequence) is the explanation of what has possibly happened, where this explanation is confined to the events we are interested in (the so-called 'faults').

Some contributions in the literature are meant to generalize the notion of a DES fault to a sequence of state transitions (where this sequence is not necessarily a whole trajectory) that either complies with a given set of specifications or, dually, violates a specification in the given set, as in the work by Jiang and Kumar (2004), where a fault is defined as the violation of any individual LTL-specification among several ones. These generalizations are orthogonal to the concept of sequence-oriented diagnosis, which aims to provide the language of all the sequences of faults that can explain the temporal observation, regardless of the specific fault notion adopted.

In the work by Jéron et. al. (2006), the notion of a DES fault is generalized to a *supervision pattern*, a DFA that can represent the ordered occurrences of multiple faults, the multiple occurrences of the same fault, or any combination of events that is reckoned to be interesting by the experts who are interested in the diagnostic output. It is tempting to speculate that sequence-oriented diagnosis resembles diagnosis with supervision patterns; after all, a pattern enables the detection of a specific language of transitions and, therefore, the detection of a specific language of faulty transitions also. In other words, given a temporal observation in input to the diagnosis task, the supervision pattern approach can find out whether there exists a trajectory that both generates that temporal observation and complies with the given (pattern) language. Notice that there can be several other trajectories that imply the temporal observation while producing sequences of faults that do not belong to the given (pattern) language: yet, the supervision pattern approach does not produce any output about them. The difference with respect to sequence-oriented diagnosis is that the latter is not given any automaton upfront recognizing a language, instead it produces a regular expression representing the language of faults of all the trajectories that imply the given temporal observation. Moreover, the output of the supervision pattern approach clarifies whether the pattern has occurred, yet does not compute the number of its occurrences, nor does it show the relative order of these occurrences or those of individual faults within the trajectories implying the temporal observation. On the other hand, from the point of view of our approach, if a fault is associated with a pattern, this can be part of a fault sequence like all other faults are. This amounts to saying once again that sequence-oriented diagnosis is orthogonal to the classification of faults, being they simple or somehow complex (Jéron et al., 2006; Lamperti & Zanella, 2011a; Lamperti & Zhao, 2014).

The notion of a DES fault is generalized also by Göessler et. al. (2019), in a way that is similar to the work by Jéron et. al. (2006). In fact, the pattern is an automaton, called an *observer*, which accepts only the sequences of state transitions that reach a state where a property is violated (for instance, a safety property like collision avoidance). This means that (only) the irreversible violation of a predefined behavioral property is reckoned as a fault for the DES, whereas in the AS approach faults are not necessarily permanent. Any property can be envisaged; most interestingly, the property is either inherent to classical failures (a device that has broken), or to actions performed by humans, or both.

The quoted paper addresses the task of explaining a perceived temporal observation, called an *observation log*, relevant to a partially observable DES that is modeled as an automaton (with no

unobservable behavioral cycles). The aim is to investigate why the given property has been violated. Two complementary notions of an explanation that have a temporal dimension, in that they are *sequences* of events (events are the triggers of the state transitions), are presented by Göessler et. al. (2019), along with the methods to automatically construct them. The former, called *subsequence-based* explanation, is the set of the shortest violating traces whose observable projection is a subsequence of the given observation log (a trace is a sequence of events). This kind of explanation, in order to retain only the events relevant to the violation, actually privileges the (violating) traces that are observably shorter (while no constraint is put on the number of unobservable events within such traces). In so doing, a (violating) trace provided as a subsequence-based explanation can possibly be inconsistent with the given observation log, that is, one possibly explains what has happened by means of a chain of events that has not happened. In the AS approach, instead, all the generated candidates are relevant to trajectories that produce the given temporal observation.

The latter notion, called *choice-based* explanation, takes into account an automaton representing all and only the violating traces that comply with the observation log. A choice-based explanation highlights (*i*) the *fateful choices* in this automaton, i.e., the state transitions that definitely move the DES closer to the violation of the property, and (*ii*) the *safe alternatives*, if any, that is, the state transitions that, if performed instead of a transition in a trace complying with the observation, bring the DES to a state that is farther from the violation. Also this form of explanation is different with respect to the candidates in the current work, as in the work by Göessler et. al. (2019) attention is focused on identifying the state transitions that have progressively approached the DES to an irreversible state and on the *level of choice* relevant to each state (namely, the minimum number of state transitions that still have to occur to bring the DES to the violation). From the point of view of the model, the quoted paper is different from the work described here as the language of the observable events is live and the cooperation of components manifests itself through synchronous transitions.

10.2 A General Approach to Diagnosis of DESs

We now consider a quite general conceptual framework for diagnosis of DESs, a framework that holds for whichever abstraction level of the diagnosis output, including the fault sequence abstraction embraced in this paper.

Most approaches to model-based diagnosis of DESs, including the one presented in this paper, first reason (offline and/or online) about the trajectories of the DES, then they draw the candidates. The *hypothesis space approach* (Grastien et al., 2011; Grastien, Haslum, & Thiébaux, 2012) reverses this order: first a candidate is assumed as a hypothesis, then it is checked whether such a hypothesis is relevant to a trajectory that is consistent with the given temporal observation. If so, the hypothesis is accepted as a real candidate, otherwise it is discarded.

The hypothesis space approach is based on the exploration of a hypothesis space, which is the domain of all the candidates relevant to a DES, according to a given notion of a diagnosis candidate and independently of the specific temporal observation. Since not all solutions are equally interesting, the approach aims at singling out those that satisfy a specific preference criterion. A *mapping* function virtually associates each evolution (which is called a trajectory in the current paper) with a single hypothesis in the hypothesis space.

Hypotheses are represented by Boolean formulae over atomic properties and consistency tests are performed by a SAT solver. The same hypothesis may correspond to several DES evolutions:

an advantage of the approach is that it suffices finding a single relevant trajectory to accept the hypothesis as a candidate. A drawback is that, if the hypothesis is unfeasible, a useless search is performed.

Different diagnosis algorithms to compute preferred candidates can be envisaged. Distinct algorithms differ in the strategies they adopt for extracting each hypothesis from the search space: preferred first strategies are typically better. Algorithms differ from each other also in the solver used to check whether a hypothesis is a candidate: several general-purpose solvers can be exploited, including planners and SAT solvers. Finally, they differ in the techniques to prune the search space, which depend on the kind of checks that are carried out.

The approach described by Grastien et. al. (2012) to perform a posteriori diagnosis of DESs is theoretically similar to a previous one presented by Grastien et. al. (2011) in that both explore the hypothesis space; however, the former adopts different tests with respect to the latter. If we assume that a candidate is the fault sequence occurring in a trajectory, thus preserving the order of faults in addition to their type and number (like in the current paper), the mapping function virtually associates a trajectory with the relevant sequence of faulty events, in other words, the hypothesis space is the language of fault sequences, which is infinite. This infinitude is coped with by adopting an implicit symbolic representation of the search space.

Unlike the approach presented in this paper, which exploits ad-hoc diagnosis engines, the hypothesis space approach invokes existing efficient general solvers. However, no specific mechanism to deal with monitoring-based diagnosis is proposed by Grastien et. al. (2011, 2012). This means that, in case the hypothesis space approach were adopted for the task of monitoring-based diagnosis, it should be run from scratch any time a new observable event is perceived, by taking as input the temporal observation obtained by appending this event to the temporal observation considered in the previous step. Moreover, the hypothesis space approach aims to compute preferred candidates only, thereby not providing any technique to exhaustively solve the diagnosis problem, whereas our work is meant to find all candidates. A further minor difference with respect to the AS approach is that faults are invariably assumed to be unobservable events.

10.3 Diagnosis of DESs Specified by Temporal Logical Formulae

While all the works quoted so far process an explicit representation of the model of the system to be diagnosed, those that are briefly surveyed in this section deal with a specification of the system through a temporal logical formula.

When a dynamical system has to be designed, a temporal logic formula can represent its most abstract specification. Later, a design is built based on this specification and checked against the specification itself by exploiting a formal verification technique, such as Model Checking (Clarke, Grumberg, & Peled, 1999). In case a model checker is adopted, the designed artifact is described as a Kripke structure, which is basically a nondeterministic transition system. This (finite-state) structure stands for the model. The model checker exhaustively explores the behavioral evolutions of the model, called *traces*. If all the traces satisfy the specification, then the check is successful. If the check fails, the model checker can produce a counterexample, namely a trace that violates the specification. The check performed by a model checker can fail owing to the incorrectness of either the specification or the model. Hence, in order to be able to fix either the former or the latter, the designer has to understand the counterexample deeply.

The work by Beer et. al. (2009) aims to analyze a counterexample trace and explain why it contradicts the specification. Hence, the inputs for the considered task are the specification and the counterexample, while the model is ignored. Specifications are expressed in LTL (Pnueli, 1977); however, the method can be employed with any monotonic temporal logic and is independent of the tool that produced the counterexample. The explanation of a counterexample deals with the question: *what values on the trace cause it to falsify the specification?* The counterexample trace is viewed as a matrix, where cell (i, j) contains the value of variable i at time j. The proposed approach looks for the cell entries that are the causes for the first occurrence of the failure of the specification on the given trace, according to the notion of causality by Halpern and Pearl (2001). An over-approximation algorithm is adopted (as detecting an exact causal set is NP-complete): the explanation is produced in a time that is linear in the size of the formula and in the length of the trace. The approach is implemented as a tool in RuleBase PE, the IBM formal verification platform: the matrix representing the trace is visualized, with red dots highlighting the cells that are the causes for the first occurrence of the violation.

As in the present paper, the work by Beer et. al. (2009) deals with finite-state systems; the diagnosis task considered differs from our paper, however. In the former, a single trace of an unknown system is given, where this trace complies with the unknown system whereas it does not with the given specification of the system itself. The emphasis of Beer et. al. (2009) is on explaining the reasons for this discrepancy between the (given trace of the) unknown system and its requirements, where the explanation is provided in terms of state variable values: if the value of a (red dot) variable at a certain time point could be switched, then the specification would not fail on the trace any longer. In our paper, instead, no such sort of trace is given, and the system model is known: the aim is to find all the sequences of faulty events that occur in the evolutions that comply with both the system and the sequence of observations received so far. Some resemblance with the work by Beer et. al. (2009) can be found in the work by Göessler et. al. (2019), quoted in Section 10.1, as the behavioral property in the latter is similar to the specification in the former.

In the work by Pill and Quaritsch (2013), which extends a work by the same authors (2012), diagnostic reasoning is aimed at assisting the designers who are writing the formal specification of a dynamical system, so as they can assess whether the specified system exhibits the desired behavior and does not allow for any undesired evolution. The quoted paper assumes that the specification is an LTL (Pnueli, 1977) formula in the context of infinite traces: this means that the system under design should evolve indefinitely. The diagnostic task takes as input an LTL formula and a lasso-shaped trace, this being a finite sequence of letters (each letter representing a signal), a sort of regular expression over the alphabet of the I/O signals of the system endowed with a suffix that can be repeated an unbounded number of times. The infinite trace represents (the infinite sequence of observable events relevant to) either a desired or an undesired behavior of the system that is being designed.

A major contribution of the paper is the proposal of an efficient encoding (trough CNF clauses), both for the formula and the trace, so as a SAT solver can check whether the trace is satisfiable. In case a trace relevant to a desired behavior satisfies the formula, everything is as expected, like when a trace relevant to an undesired behavior does not satisfy the formula. Otherwise, there is a discrepancy between the expected and the specified behavior, in other words, the specification is faulty and some diagnostic reasoning has to be performed.

The approach proposed by Pill and Quaritsch (2013) aims to find all the candidate diagnoses, where a candidate is a (subset-minimal) set of (LTL and Boolean) operators whose simultaneous

incorrectness explains the above discrepancy. Operators are indeed the items the designer is working with, so the granularity level of the search space is appropriate. The above encoding needs being extended in order to find the conflicts (Reiter, 1987): candidates can be computed as the minimal hitting sets of such conflicts. Conflicts turn out to be the minimal unsatisfiable cores (Schuppan, 2012) of the LTL specification, hence a SAT solver that can draw these cores is required. The formula encoding is derived directly from locally considering the single operators in the syntax tree of the formula.

Distinct forms of encoding are proposed for adopting either weak or strong fault models. The strong fault model of an operator includes the descriptions of some alternatives of the operator itself, in other words, the designer may have used the operator erroneously, instead of an alternative one. Hence, in case strong fault models are adopted, candidate diagnoses directly suggest repairs (e.g. the replacement of an operator with another one or the flipping of its operands).

The automaton that is implicitly represented by an LTL formula in the work by Pill and Quaritsch (2013) corresponds to the space of the DES in the approach presented in this paper, the same as a trace in the work by Pill and Quaritsch (2013) corresponds to a temporal observation in this paper. However, Pill and Quaritsch consider only automata having infinite evolutions, whereas any DES model taken into account in this paper can have both finite and infinite evolutions. Moreover, a trace in the work by Pill and Quaritsch (2013) is relevant to an infinite temporal observation, that is, (the finite representation of) an infinite temporal observation is taken as input. The diagnosis task considered in this paper, instead, takes as input an observable event at a time, while dealing with unbounded temporal observations (under the form of observation patterns) only to upgrade partial explainers.

The work by Pill and Quaritsch (2013) provides an efficient way to check whether a given infinite trace complies with the given specification of the system behavior, that is, whether an infinite sequence of observable events can be generated by the (implicit) automaton. This check could be done also by exploiting the implicit representation of the space of the DES presented in this paper; however, this is not the task accomplished by our approach. The second contribution by Pill and Quaritsch (2013), namely finding the candidates relevant to the LTL specification, is a different diagnosis task, amounting basically to the diagnosis of a static system (the formula), with respect to the monitoring-based diagnosis of DESs in this paper.

A variant of LTL that implements finite semantics is introduced by Manna and Pnueli (1995) and called FLTL by Bauer et. al. (2010). Finite semantics matches the finite temporal scope of a test case execution for a software program; hence an FLTL formula can represent the underlying knowledge base of a test oracle. A test case is a pair consisting in a finite word over a given alphabet, which specifies the input interactions with the artifact, and an oracle, namely an FLTL formula that expresses our expectations about the resulting behavior when the artifact is given a word as input. When judging the test case execution, both the input word and the corresponding finite word representing the outputs produced by the artifact are considered: the combination of the input word and the output word is called an *execution trace*. Then, it has to be checked whether the expectations are met, that is, whether the execution trace 'implements' the oracle (formula).

In the work by Pill and Wotawa (2018), a SAT encoding that can be used for implementing an automated FLTL oracle is introduced. This oracle not only provides a verdict (passed or failed), but, in case the execution trace violates the FLTL formula, it also shows how the individual sub-formulae evolve along the trace.

In a subsequent work, Pill and Wotawa (2019) address the problem of explaining why an automated FLTL oracle judged a test case to be failing; in a model-based diagnosis perspective, they are focused on deriving the faulty parts in the formula, those that are relevant to the violation. The aim is to support the software designers/programmers by providing information that can be helpful to come up with the desired repair in the debugging process. In case the test failed, the SAT encoding of the FLTL formula (used for implementing the oracle) is extended in a way similar to the work by Pill and Quaritsch (2013), that is, by augmenting it with health state variables for all the individual operators in the formula itself. Model-based diagnosis is employed to isolate those sub-formulae that were violated. Diagnoses can be computed as the minimal hitting sets of the minimal conflicts (Reiter, 1987), the minimal conflicts being characterized as chains from some sub-formula to the root of the parse tree of the whole FLTL formula. The aim is not to localize the faulty parts in the software system, it is instead to derive the parts that failed in the oracle, i.e., the fault sources in the formula. Hence, the model-based diagnosis approach does not account for the model of the software system, it rather considers the 'model' of the oracle. A candidate diagnosis suggests that "if this and that part of the formula would have been different, then the test case execution could have been judged to have passed". In other words, a candidate diagnosis provides some knowledge about how the observed behavior violated the FLTL property.

The automaton that is implicitly represented by an FLTL formula by Pill and Wotawa (2018, 2019) corresponds to the space of the DES in our paper. However, only automata having finite evolutions are considered in those works, whereas, as remarked above, any automaton (representing a DES model) taken into account in this paper can have both finite and infinite evolutions.

The work by Pill and Wotawa (2018) provides an efficient way to check whether a given finite trace complies with the given specification of the system behavior (this specification is the oracle), which is not the task accomplished by our approach. The work by Pill and Wotawa (2019) is aimed at finding out the faulty parts (typically, the faulty operators) in the specification, given a failing trace, where a trace consists in all the recorded observable events, whereas the task of monitoring-based diagnosis addressed in this paper takes as input an observable event at a time.

10.4 Health State

In the literature, there exists a further notion of a diagnosis output besides the ones discussed in this paper, which is called an *health state* (Stern, Kalech, Rogov, & Feldman, 2017). This requires that the system model includes the prior probability of individual components to be faulty, so as the posterior probability that each candidate is the real candidate given the observation can be estimated. Indeed, several proposals in the literature suggest to output, in addition to the set of candidates, a probability distribution over such candidates, or to compute the most likely candidates only. The claim by Stern et. al. (2017), instead, is that viewing the most probable candidates may be misleading as it ignores the information stored in all the other candidates, whereas providing a human operator with the posterior probabilities relevant to single components is more meaningful: this is the reason for the proposal of a health state.

Under the assumption of priors known, a health state (that is, the output of the diagnosis task) is a function that either assigns to each component its posterior probability to be faulty given the observation, if the fault model is weak, or to each pair (component, behavior mode) the posterior probability that, given the observation, the said behavior mode is the real one of the said component,

if the fault model is strong (and hence it specifies different behavior modes of each component for different faults affecting the component itself).

Although the assumption of priors known has not been made for ASs, there is no problem (at least conceptually) in extending the models of AS components to include prior failure probabilities. In the work by Stern et. al. (2017), the health state of a system is calculated based on an adaptive number of set-oriented candidates, as the challenge is to produce an accurate health state without computing all the candidates. We remark here that whether the health state can be drawn from (an adaptive number of) sequence-oriented candidates could be investigated in the future. Exploiting sequence-oriented candidates could be beneficial in estimating the likelihood that a component is affected by intermittent faults.

11. Conclusion

In our views, standard approaches to model-based diagnosis of DESs suffer from two major drawbacks: *narrow explainability* and *computational complexity*, which are discussed below.

11.1 Narrow Explainability

When we argue that set-oriented candidates relevant to dynamical systems, DESs in particular, suffer from narrow explainability, we refer to the interpretability of the collection of candidates by a human expert. Although a set of faults is a natural representation for a diagnosis candidate of static systems, it may become unnatural for dynamical systems, DESs included, owing to an additional *temporal* dimension coming into play, which is not embedded in candidates. The approach proposed in this paper provides more detailed explanations than the traditional set-oriented DES candidates produced by abductive diagnosis, inasmuch it adopts a sequence-oriented perspective: each candidate is a *fault sequence*, that is, the possibly unbounded list (ordered multiset) of faults relevant to a trajectory that entails the given temporal observation. In the sequence-oriented perspective, the temporal dimension is the chronological order of fault occurrences, which is a first step towards a better explanation, although there still may be some trajectories, affected by the sequence of faults in the candidate, that do not produce the given temporal observation. Our claim is that sequence-oriented candidates of DESs can help a practitioner understand what has happened in the system modeled by the DES more than set-oriented candidates can do, and that the temporal dimension may be essential for a precise assessment of the system behavior, especially in safety critical domains.

The candidates produced by our approach are sound and complete. In previous work about set-oriented diagnosis of ASs, all the distinct set of faults inherent to distinct trajectories entailing the given observation are generated; this means that minimality is not enforced (that is, a candidate is possibly a subset of another), the same as no other preference criterion is enforced. Likewise, in sequence-oriented diagnosis of ASs in this paper, all the distinct sequences of faults inherent to distinct trajectories entailing the given observation are generated; this means that minimality is not enforced, that is, the resulting regular expression can be the alternative of several regular expressions where one is possibly a prefix of another, or is possibly subsumed by another (for instance, ab is subsumed by c^*ab).

The three diagnosis engines described in this paper have been implemented and provided online (cf. Section 8). A variety of simplification rules have been applied in order to mitigate the redundancy in the resulting regular expressions. However, one may argue that regular expressions are not easy to understand for diagnosticians that are not computer scientists. Replying to this comment

would require a study about the cognitive perception of a regular expression by a user who has never heard about regular expressions before, and that has been briefly taught their syntax and semantics. The study could be comparative, in case alternative ways to represent regular expressions could be envisaged (e.g. by means of graphs, or through animations). In the literature, a branch of research about explanations is relevant to the way explanations are presented. In fact, as illustrated by Miller (2019), XAI lays in the area of human-agent interaction, which in turn is the intersection of AI, social sciences, and human-computer interaction. These studies are beyond the scope of our paper.

In the XAI literature, the work by Ribeiro et. al. (2016) gives two different (yet related) definitions of trust that can be generalized to any intelligent system. The former definition is inherent to each specific output (the question is whether a user trusts this output enough to take some action based on it), the latter is inherent to the system that has generated the output (the question is whether the user trusts the system to behave in reasonable ways if deployed). The former notion is fundamental if one plans to make a decision based on the output, the latter is important when we have to choose whether to deploy an intelligent agent. The quoted paper proposes providing explanations for individual outputs as a solution to the "trusting an output" problem, and selecting multiple such outputs (and explanations) as a solution to the "trusting the system" problem. The above two notions of trust are quite interesting also for diagnostic systems: we have to differentiate the trust in an individual output from the trust in the diagnostic system that has been run to produce it. Since a model-based diagnosis system is a general engine that can process any model belonging to a certain domain (e.g. a DES model), the latter notion becomes that of trust in the specific model. In the realm of model-based diagnosis of DESs we assume that the model is complete: in fact, the model may be incomplete or unreliable, however. We believe that sequence-oriented diagnosis can also help practitioners understand and evaluate DES models. Building proper techniques to suggest which diagnostic problem instances and relevant candidates to inspect for assessing the trust in a given DES model is an interesting topic for future research.

11.2 Computational Complexity

The second major drawback of model-based diagnosis of DESs is computational complexity, owing to the exponential explosion of the number of states embedded in the discrete space of the DES. In this respect, both the *compiled diagnoser* and the *interpreted diagnoser* approaches to diagnosis of DESs (cf. Section 9) suffer, to varying degrees, from the same computational issues. A sequence-oriented instantiation of the AS diagnosis engine with no knowledge compilation (as in the interpreted approach) and with total knowledge compilation (as in the compiled approach) have been presented in this paper under the guise of *blind* and *greedy* diagnosis, respectively. Evidence from the (perhaps unsurprising) experimental results (cf. Section 8.2) suggests that both approaches are inadequate for DESs including more than a few components. The compiled knowledge structure adopted by greedy diagnosis, the *explainer*, has a size that is smaller than that of the diagnoser, the structure of the original compiled diagnoser approach (Sampath et al., 1995, 1996), which supports the online computation of set-oriented candidates only. However, a smaller size of an explainer comes with a disadvantage: the explainer is an NFA, whereas the diagnoser is a DFA. Hence, in greedy diagnosis, the time required by the online search within the explainer may be longer than in the compiled diagnoser approach. Moreover, building a complete explainer is out of the question for real size DESs. In the construction of a complete explainer, the generation of the regular expres-

sions is very expensive since, in fact, there is a trade-off between the richness of the information embedded in the diagnosis output and the time required for its computation.

A possible mitigation of the complexity problem comes from *lazy* diagnosis, which performs only partial knowledge compilation, thus obtaining a *partial explainer*. Lazy diagnosis is in fact a *hybridization* of the interpreted (resp. *blind*) and compiled (resp. *greedy*) diagnoser approaches. The experimental activity recorded in this paper indicates a possible viability of lazy diagnosis, both for the construction of the partial explainer and for the online processing time in order to compute the candidates. A partial explainer can be progressively extended, the upgrade being relevant either just to a single observable event (the latest one perceived, in case it is not encompassed yet by the current partial explainer) or to the observation pattern corresponding to a behavioral scenario. In the latter case, the size of the upgrade can be large, because of two main reasons: the observation pattern may correspond to several further trajectories in addition to the ones defined by the behavioral scenario, and the upgrade algorithm generates all the fault spaces inherent to all the (space) states that can be reached through every prefix of the temporal observations defined by the pattern, even in case, starting from such (space) states, the observation suffixes cannot be followed. A tighter control on the size of the upgrade (i.e., on the number of fault spaces to be added) can be achieved by using, instead of the observation pattern, the scenario abduction, so as to build (if needed) only the fault spaces relevant to the (space) states included in it that are entered by an observable transition. The upgrade of an explainer based on a scenario abduction can be supported by a future version of the diagnosis engine developed.

The execution time needed to build each fault space in the (partial) explainer can possibly be reduced by using some heuristic strategy. Given the same automaton, the state elimination algorithm (Brzozowski & McCluskey, 1963), and, hence, its variant adopted in the approach proposed in this paper, may compute distinct (equivalent) regular expressions, which have different sizes, depending on the ordering in which the states are removed. The elimination ordering can be chosen heuristically, based on the features of the given automaton, in order to reduce the execution time and the size of the resulting regular expression(s). Empirical comparisons of several heuristics are reported in the literature, including the works by Gruber et al. (2009) and Moreira et. al. (2010): the greedy heuristic by Delgado and Morais (2004) is shown to perform best in most cases.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China (grant number 61972360).

Appendix A. Additional Hints on Complexity

A.1 BLIND DIAGNOSIS ENGINE

Whenever called, the BLIND DIAGNOSIS ENGINE algorithm produces a new cluster starting from the last cluster reached in the previous call. Each transition from the current cluster to the new cluster generates the newly-received observation o. Very pessimistically, the time needed to update the instance of the observation-constrained space upon reception of a new observation is $O(t)$: in order to add the new cluster, we have at most to consider all the transitions in the Space, the observable ones to generate the states at the 'entrance' of the cluster, and the unobservable ones to generate the remaining states.

As to the size of the cluster, its internal transitions are at most all the unobservable transitions in the DES Space, namely t_u. The upper bound of the number of Space states in a cluster is the sum of the upper bound of (i) the number of 'entrance' states of the cluster, s_o^o, and *(ii)* the number of Space states that are distinct from the 'entrance' states and are entered by an unobservable transition, where this number is not greater than s_u. As all the states in a cluster have the same index, they cannot exceed the number of states in the Space. Hence the upper bound of the number of Space states in a cluster is $min\{s_o^o + s_u, s\}$, which is $O(s)$.

The largest size of the new observation-constrained space corresponds to the case when no pruning has ever been performed. In this case, the upper bound of the total number of states in the new observation-constrained space is given by the product of the upper bound of the number of states in each cluster, which is $O(s)$, by the number of clusters, this being the length of the formerly perceived temporal observation incremented by one, that is, $(|\mathcal{O}| + 1)$. All (and only) the states in the final cluster are final. The case with no pruning is the worst case for the conversion of the observation-constrained space into a regular expression. The conversion is performed by algorithm CANDIDATES, which is indeed the algorithm by Brzozowski and McCluskey (1963). Hence, the complexity of this conversion is $O\left(s^3\left(|\mathcal{O}|+1\right)^3 4^{s(|\mathcal{O}|+1)}\right)$. This function has to be added to $O(t)$, which accounts for the update of the instance of the observation-constrained space, to obtain the cost of BLIND DIAGNOSIS ENGINE.

If some pruning is performed when updating the current observation-constrained space upon reception of a new observation, the number of states in the new observation-constrained space is not the maximum one, which considerably shrinks the effort of the conversion into a regular expression. However, the cascade of pruning steps propagating backward requires a cost that, very pessimistically, is upper bounded by $t|\mathcal{O}|$, which means processing the (upper bound of the number of) transitions in all the previous clusters.

A.2 GREEDY DIAGNOSIS ENGINE

Algorithm GREEDY DIAGNOSIS ENGINE takes as input a (complete) explainer.[24] The other input parameters are the trace (Definition 10) relevant to the previous observable events in the temporal observation[25], and a new observation o. The algorithm updates the trace based on o, and generates the diagnosis relevant to the temporal observation perceived so far.

Each node in the trace cannot include more than s_o^o fault spaces, where, as known, $s_o^o \leq s_{\text{o\_max}}^o \leq s_o$. Let $t_{\text{o\_max}}^{\text{out}}$ be the maximum number of transitions with observable label o exiting a fault space for o ranging over every observable label. Notice that $t_{\text{o\_max}}^{\text{out}} \leq (t - t_u)$, where $(t - t_u)$ is the number of observable Space transitions. Hence, the upper bound of the number of transitions exiting the current node in the trace to reach the new one is $s_{\text{o\_max}}^o t_{\text{o\_max}}^{\text{out}}$, that is, the product of (the upper bound of) the number of fault spaces in the node by (the upper bound of) the number of transitions, marked by the same observable label, exiting a fault space. This is the cost of line 1 in Algorithm 3. The upper bound of the cost of the line 2 is given by the product of the number of transitions entering the new node of the trace and that of the number of transitions exiting it, that is, $\left(s_{\text{o\_max}}^o t_{\text{o\_max}}^{\text{out}}\right)^2$. The cost of line 2, namely $O\left(\left(s_{\text{o\_max}}^o t_{\text{o\_max}}^{\text{out}}\right)^2\right)$, is the asymptotic cost of the algorithm, as the third (and last) line takes a constant time.

24. The complexity relevant to the offline construction of the explainer is analyzed in Section 9.1.
25. This initially empty trace has been built in the previous calls of algorithm GREEDY DIAGNOSIS ENGINE itself.

A.3 LAZY DIAGNOSIS ENGINE

Lazy diagnosis handles a partial explainer. The effort to generate a partial explainer upfront consisting of s_e fault spaces is that relevant to s_e calls of FAULT SPACE, whose (single-run) complexity is analyzed in Section 9.1, incremented by the cost of building the transitions between such fault spaces, which is $O(s_e(t - t_u))$.

Algorithm LAZY DIAGNOSIS ENGINE takes as input a new observation o and, based on it, extends both the given partial explainer, if needed, and the trace; finally, it generates the diagnosis relevant to the temporal observation perceived so far.

The upper bound of the number of fault spaces that are added to the partial explainer is $s^o_{\text{o\_max}}$, that is (cf. Section 9.1), the maximum number of Space states that have an entering transition whose associated observable label is o. As already remarked, the effort to build each fault space (algorithm FAULT SPACE) is $O\left(t_u + (s_u+1)^3 4^{(s_u+1)}\right)$. Hence, the cost to build (at most) $s^o_{\text{o\_max}}$ fault spaces and an entering transition for each of them is $O\left(s^o_{\text{o\_max}}\left(t_u + (s_u+1)^3 4^{(s_u+1)}\right)\right)$. The cost of lines 9-11 of Algorithm 5 (they are the same lines as in algorithm GREEDY DIAGNOSIS ENGINE, whose complexity has already been analyzed) is $O\left((s^o_{\text{o\_max}} t^{\text{out}}_{\text{o\_max}})^2\right)$, hence the time complexity relevant to the whole algorithm is $O\left(s^o_{\text{o\_max}}\left(t_u + (s_u+1)^3 4^{(s_u+1)}\right)\right) + O\left(\left(s^o_{\text{o\_max}} t^{\text{out}}_{\text{o\_max}}\right)^2\right)$, which gives $O\left(s^o_{\text{o\_max}}\left(t_u + s^o_{\text{o\_max}}\left(t^{\text{out}}_{\text{o\_max}}\right)^2 + (s_u+1)^3 4^{(s_u+1)}\right)\right)$.

If necessary, algorithm LAZY DIAGNOSIS ENGINE upgrades a partial explainer based on the latest perceived observation. However, an upgrade can also be achieved based on an observation pattern corresponding to a behavioral scenario, where this observation pattern has been drawn from the scenario abduction. In the following, we will consider the cost of this upgrade. To this end, we analyze the complexity of the generation of a scenario abduction. Given a scenario $\mathcal{S} = (\mathbb{T}, \mathcal{L})$, let S be the number of states of the DFA A recognizing the language $\mathcal{L}$. Assume to virtually remove from the DES Space: (1) all the instances of the transitions that belong to $\mathbb{T}$ and have no instance in A, and (2) all the states that, after this removal, have become unreachable starting from the initial Space state. We denote with Space' the space obtained that way, and s' and t' the number of its states and transitions, respectively. The upper bound of the number of states resulting from the (virtual) composition of Space' with A is (Ss'),[26] while the upper bound of the number of transitions is (St').[27] Therefore, the (very pessimistic) time complexity of the generation of the scenario abduction is $O(St')$.

We now analyze the complexity of the generation of the observation pattern relevant to a given scenario. Let s_a and t_a be the number of states and transitions in the scenario abduction, respectively, where $s_a = O(Ss')$ and $t_a = O(St')$ (see above). The replacement of each symbol marking a transition in the scenario abduction with the relevant (possibly empty) observable label, thus obtaining an NFA, takes a time $\Theta(t_a)$. The determinization of the NFA and the minimization of the resulting DFA, which is the observation pattern, takes a time $O(2^{s_a})$.

Finally, we consider algorithm PARTIAL EXPLAINER UPGRADE, which upgrades a partial explainer (having s_e fault spaces, where $s_e > 0$) based on an observation pattern that has been drawn from a scenario abduction. The lower bound of the number of new fault spaces to be added to the

26. This upper bound is obtained by assuming that in the resulting automaton there is a state for each distinct pair of states, one in Space' and the other in A.

27. This upper bound is obtained by assuming that the exiting transitions of each distinct pair, namely (Space' state, state of DFA A), are the same as the exiting transitions of the considered Space' state.

partial explainer equals the number of distinct Space states in the abduction[28] that are either the initial Space state or a Space state that is entered by an observable transition and that is not the 'entrance' state of any fault space in the partial explainer. The upper bound of the number of new fault spaces to be added to the partial explainer is s_n, this being defined as $min\left\{s_o - s_e, s^o_{\text{o\_max}}\ell\right\}$, where ℓ is the number of distinct observable labels in the observation pattern. The term $(s_o - s_e)$ is the difference between the upper bound of the number of fault spaces in the complete explainer and the number of fault spaces that are currently present in the partial explainer. The term $\left(s^o_{\text{o\_max}}\ell\right)$ is an upper bound for the number of fault spaces relevant to all and only the Space states that are entered by transitions marked by observable labels that belong to the observation pattern. Hence, the effort required to upgrade the partial explainer is $O\left(s_n\left(t_u + (s_u+1)^3 4^{(s_u+1)}\right)\right) + O\left((s_e + s_n)\ell t^{\text{out}}_{\text{o\_max}}\right)$, where the first addend is inherent to the construction of the new fault spaces, while the second one is relevant to transition processing and the construction of the new transitions that enter these fault spaces.

References

Alur, R., & Dill, D. (1994). A theory of timed automata. *Theoretical Computer Science*, *126*(2), 183–235.

Baroni, P., Lamperti, G., Pogliano, P., & Zanella, M. (1998). Diagnosis of active systems. In *Thirteenth European Conference on Artificial Intelligence (ECAI 1998)*, pp. 274–278, Brighton, United Kingdom.

Baroni, P., Lamperti, G., Pogliano, P., & Zanella, M. (1999). Diagnosis of large active systems. *Artificial Intelligence*, *110*(1), 135–183.

Baroni, P., Lamperti, G., Pogliano, P., & Zanella, M. (2000). Diagnosis of a class of distributed discrete-event systems. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, *30*(6), 731–752.

Basile, F. (2014). Overview of fault diagnosis methods based on Petri net models. In *Proceedings of the 2014 European Control Conference, ECC 2014*, pp. 2636–2642.

Bauer, A., Leucker, M., & Schallhart, C. (2010). Comparing LTL semantics for runtime verification. *Journal of Logic and Computation*, *20*(3), 651–674.

Beer, I., Ben-David, S., Chockler, H., Orni, A., & Trefler, R. (2009). Explaining counterexamples using causality. In Bouajjani, A., & Maler, O. (Eds.), *21st International Conference on Computer-Aided Verification (CAV 2009)*, Vol. 5643 of *Lecture Notes in Computer Science*, pp. 94–108. Springer-Verlag, Berlin, Heidelberg.

Benveniste, A., Fabre, E., Haar, S., & Jard, C. (2003). Diagnosis of asynchronous discrete-event systems: A net unfolding approach. *IEEE Transactions on Automatic Control*, *48*, 714–727.

Bertoglio, N., Lamperti, G., & Zanella, M. (2019a). Intelligent diagnosis of discrete-event systems with preprocessing of critical scenarios. In Czarnowski, I., Howlett, R., & Jain, L. (Eds.), *Intelligent Decision Technologies 2019*, Vol. 142 of *Smart Innovation, Systems and Technologies*, pp. 109–121. Springer, Singapore.

28. Recall that each abduction state is a pair (Space state, recognizer DFA state).

Bertoglio, N., Lamperti, G., & Zanella, M. (2019b). A posteriori diagnosis of discrete-event systems with symptom dictionary and scenarios. In Wotawa, F., Friedrich, G., Pill, I., Koitz-Hristov, R., & Ali, M. (Eds.), *Advances and Trends in Artificial Intelligence. From Theory to Practice. IEA/AIE 2019*, Vol. 11606 of *Lecture Notes in Computer Science*, pp. 325–333. Springer International Publishing, Cham.

Bertoglio, N., Lamperti, G., & Zanella, M. (2019c). Temporal diagnosis of discrete-event systems with dual knowledge compilation. In Holzinger, A., Kieseberg, P., Weippl, E., & Tjoa, A. M. (Eds.), *Machine Learning and Knowledge Extraction*, Vol. 11713 of *Lecture Notes in Computer Science*, pp. 333–352. Springer, Berlin.

Bertoglio, N., Lamperti, G., Zanella, M., & Zhao, X. (2019d). Twin-engined diagnosis of discrete-event systems. *Engineering Reports*, *1*, 1–20.

Bertoglio, N., Lamperti, G., Zanella, M., & Zhao, X. (2020a). Diagnosis of temporal faults in discrete-event systems. In Giacomo, G. D., Catala, A., Dilkina, B., Milano, M., Barro, S., Bugarín, A., & Lang, J. (Eds.), *24th European Conference on Artificial Intelligence (ECAI 2020)*, Vol. 325 of *Frontiers in Artificial Intelligence and Applications*, pp. 632–639. IOS Press, Amsterdam.

Bertoglio, N., Lamperti, G., Zanella, M., & Zhao, X. (2020b). Explanatory diagnosis of discrete-event systems with temporal information and smart knowledge-compilation. In Calvanese, D., Erdem, E., & Thielsher, M. (Eds.), *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020)*, pp. 130–140. IJCAI Organization.

Bertoglio, N., Lamperti, G., Zanella, M., & Zhao, X. (2020c). Explanatory monitoring of discrete-event systems. In Czarnowski, I., Howlett, R., & Jain, L. (Eds.), *Intelligent Decision Technologies 2020*, Vol. 193 of *Smart Innovation, Systems and Technologies*, pp. 63–77. Springer, Singapore.

Bertoglio, N., Lamperti, G., Zanella, M., & Zhao, X. (2020d). Temporal-fault diagnosis for critical-decision making in discrete-event systems. In Cristani, M., Toro, C., Zanni-Merk, C., Howlett, R., & Jain, L. (Eds.), *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 24th International Conference KES2020*, Vol. 176 of *Procedia Computer Science*, pp. 521–530. Elsevier.

Boselli, R., Cesarini, M., Mercorio, F., & Mezzanzanica, M. (2014). Planning meets data cleansing. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS 2014)*, pp. 439–443, Portsmouth, New Hampshire, USA. Association for the Advancement of Artificial Intelligence.

Box, G., & Muller, M. (1958). A note on the generation of random normal deviates. *The Annals of Mathematical Statistics*, *29*(2), 610–611.

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

Brzozowski, J., & McCluskey, E. (1963). Signal flow graph techniques for sequential circuit state diagrams. *IEEE Transactions on Electronic Computers*, *EC-12*(2), 67–76.

Cabasino, M. P., Giua, A., & Seatzu, C. (2010). Fault detection for discrete event systems using Petri nets with unobservable transitions. *Automatica*, *46*, 1531–1539.

Cassandras, C., & Lafortune, S. (2008). *Introduction to Discrete Event Systems* (second edition). Springer, New York.

Cerutti, S., Lamperti, G., Scaroni, M., Zanella, M., & Zanni, D. (2007). A diagnostic environment for automaton networks. *Software: Practice and Experience*, *37*(4), 365–415. DOI: 10.1002/spe.773.

Clarke, E., Grumberg, O., & Peled, D. (1999). *Model Checking*. MIT Press, Cambridge, MA, USA.

Cong, X., Fanti, M., Mangini, A., & Li, Z. (2018). Decentralized diagnosis by Petri nets and integer linear programming. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *48*(10), 1689–1700.

Console, L., Picardi, C., & Dupré, D. T. (2003). Temporal decision trees: Model-based diagnosis of dynamic systems on-board. *Journal of Artificial Intelligence Research*, *19*, 469–512.

Courcoubetis, C., Vardi, M., Wolper, P., & Yannakakis, M. (1991). Memory efficient algorithms for the verification of temporal properties. In Clarke, E., & Kurshanr, R. (Eds.), *Second International Conference on Computer-Aided Verification (CAV 1990)*, Vol. 531 of *Lecture Notes in Computer Science*, pp. 233–242. Springer-Verlag, Berlin, Heidelberg.

Daniele, M., Giunchiglia, F., & Vardi, M. (1999). Improved automata generation for linear temporal logic. In Halbwachs, N., & Peled, D. (Eds.), *11th International Conference on Computer-Aided Verification (CAV 1999)*, Vol. 1633 of *Lecture Notes in Computer Science*, pp. 249–260. Springer-Verlag, Berlin, Heidelberg.

Darwiche, A. (2001). Decomposable negation normal form. *Journal of the ACM*, *48*(4), 608–647.

de Kleer, J., & Williams, B. (1987). Diagnosing multiple faults. *Artificial Intelligence*, *32*(1), 97–130.

Delgado, M., & Morais, J. (2004). Approximation to the smallest regular expression for a given regular language. In Domaratzki, M., Okhotin, A., Salomaa, K., & Yu, S. (Eds.), *9th Conference on Implementation and Application of Automata (CIAA 2004), Kingston, Ontario, Canada, July 2004*, Vol. 3317 of *Lecture Notes in Computer Science*, pp. 312–314. Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-540-30500-2˙31.

Emerson, E. (1990). Temporal and modal logic. In van Leeuwen, J. (Ed.), *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, chap. 16, pp. 995–1072. MIT Press, Cambridge, MA, USA.

Feldman, A., Provan, G., & van Gemund, A. (2010a). Approximate model-based diagnosis using greedy stochastic search. *Journal of Artificial Intelligence Research*, *38*, 371–413.

Feldman, A., Provan, G., & van Gemund, A. (2010b). A model-based active testing approach to sequential diagnosis. *Journal of Artificial Intelligence Research*, *39*, 301–334.

Feng, Z., & Grastien, A. (2020). Model based diagnosis of timed automata with model checkers. In *31st International Workshop on Principles of Diagnosis (DX-20)*, Online.

Gerth, R., Peled, D., Vardi, M. Y., & Wolper, P. (1996). Simple on-the-fly automatic verification of linear temporal logic. In Dembiński, P., & Średniawa, M. (Eds.), *Proceedings of the Fifteenth IFIP International Symposium on Protocol Specification, Testing and Verification, Warsaw, Poland, June 1995*, pp. 3–18. Springer US, Boston, MA.

Göessler, G., Mari, T., Pencolé, Y., & Travé-Massuyès, L. (2019). Towards causal explanations of property violations in discrete event systems. In *30th International Workshop on Principles of Diagnosis (DX-19)*, Klagenfurt, Austria.

Grastien, A., Cordier, M., & Largouët, C. (2005). Incremental diagnosis of discrete-event systems. In *Sixteenth International Workshop on Principles of Diagnosis (DX 2005)*, pp. 119–124, Monterey, CA.

Grastien, A., Haslum, P., & Thiébaux, S. (2011). Exhaustive diagnosis of discrete event systems through exploration of the hypothesis space. In *22nd International Workshop on Principles of Diagnosis (DX 2011)*, pp. 60–67, Murnau, Germany.

Grastien, A., Haslum, P., & Thiébaux, S. (2012). Conflict-based diagnosis of discrete event systems: theory and practice. In *Thirteenth International Conference on Knowledge Representation and Reasoning (KR 2012)*, pp. 489–499, Rome, Italy. Association for the Advancement of Artificial Intelligence.

Gruber, H., & Holzerl, M. (2014). From finite automata to regular expressions and back — a summary on descriptional complexity. In Ésik, Z., & Fülöp, Z. (Eds.), *Proceedings of the Fourteenth International Conference on Automata and Formal Languages (AFL 2014), Szeged, Hungary, May 2014*, pp. 25–48. Open Publishing Association.

Gruber, H., Holzerl, M., & Tautschnig, M. (2009). Short regular expressions from finite automata: empirical results. In Maneth, S. (Ed.), *Proceedings of the Fourteenth International Conference on Implementation and Application of Automata (CIAA 2009), Sydney, Australia, July 2009*, Vol. 5642 of *Lecture Notes in Computer Science*, pp. 188–197. Springer.

Halpern, J., & Pearl, J. (2001). Causes and explanations: A structural-model approach: Part 1: Causes. In Breese, J. S., & Koller, D. (Eds.), *17th Conference on Uncertainty in Artificial Intelligence (UAI 2001), University of Washington, Seattle, Washington, USA, August 2001*, pp. 194–202, San Francisco, CA, USA. Morgan Kaufmann Publishers.

Hamscher, W., Console, L., & de Kleer, J. (Eds.). (1992). *Readings in Model-Based Diagnosis*. Morgan Kaufmann, San Mateo, CA.

Hashtrudi Zad, S., Kwong, R., & Wonham, W. (2005). Fault diagnosis in discrete-event systems: incorporating timing information. *IEEE Transactions on Automatic Control*, *50*(7), 1010–1015.

Hopcroft, J., Motwani, R., & Ullman, J. (2006). *Introduction to Automata Theory, Languages, and Computation* (third edition). Addison-Wesley, Reading, MA.

Jéron, T., Marchand, H., Pinchinat, S., & Cordier, M. (2006). Supervision patterns in discrete event systems diagnosis. In *Workshop on Discrete Event Systems (WODES 2006)*, pp. 262–268, Ann Arbor, MI. IEEE Computer Society.

Jiang, S., & Kumar, R. (2004). Failure diagnosis of discrete-event systems with linear-time temporal logic specifications. *IEEE Transactions on Automatic Control*, *49*(6), 934–945.

Jiang, S., Kumar, R., & Garcia, H. (2003). Diagnosis of repeated/intermittent failures in discrete event systems. *IEEE Transactions on Robotics and Automaton*, *19*(2), 310–323.

Jiroveanu, G., Boel, R., & Bordbar, B. (2008). On-line monitoring of large Petri net models under partial observation. *Journal of Discrete Event Dynamic Systems*, *18*, 323–354.

Kesten, Y., Manna, Z., McGuire, H., & Pnueli, A. (1993). A decision algorithm for full propositional temporal logic. In Courcoubetis, C. (Ed.), *International Conference on Computer Aided Verification (CAV 1993), Elounda, Greece, June 1993*, Vol. 697 of *Lecture Notes in Computer Science*, pp. 97–109, Berlin, Heidelberg. Springer.

Lamperti, G., & Zanella, M. (2002). Diagnosis of discrete-event systems from uncertain temporal observations. *Artificial Intelligence*, *137*(1–2), 91–163.

Lamperti, G., & Zanella, M. (2004). A bridged diagnostic method for the monitoring of polymorphic discrete-event systems. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, *34*(5), 2222–2244.

Lamperti, G., & Zanella, M. (2011a). Context-sensitive diagnosis of discrete-event systems. In Walsh, T. (Ed.), *Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI 2011)*, Vol. 2, pp. 969–975, Barcelona, Spain. AAAI Press.

Lamperti, G., & Zanella, M. (2011b). Monitoring of active systems with stratified uncertain observations. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, *41*(2), 356–369.

Lamperti, G., & Zanella, M. (2013). Preliminaries on complexity of diagnosis of discrete-event systems. In *24th International Workshop on Principles of Diagnosis (DX 2013)*, pp. 192–197, Jerusalem, Israel.

Lamperti, G., Zanella, M., & Zhao, X. (2018). *Introduction to Diagnosis of Active Systems*. Springer, Cham.

Lamperti, G., Zanella, M., & Zhao, X. (2020). Diagnosis of deep discrete-event systems. *Journal of Artificial Intelligence Research*, *69*, 1473–1532.

Lamperti, G., & Zhao, X. (2014). Diagnosis of active systems by semantic patterns. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *44*(8), 1028–1043.

Li, B., Khlif-Bouassida, M., & Toguyéni, A. (2019). Reduction rules for diagnosability analysis of complex systems modeled by labeled Petri nets. *IEEE Transactions on Automation Science and Engineering*.

Lundberg, S., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, December 2017*, pp. 4768–4777, Red Hook, NY, USA. Curran Associates Inc.

Lunze, J. (2000). Diagnosis of quantized systems based on a timed discrete-event model. *IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans*, *30*(3), 322–335.

Manna, Z., & Pnueli, A. (1995). *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York.

Marques-Silva, J., & Ignatiev, A. (2022). Delivering trustworthy AI through formal XAI. In *Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI-22)*, pp. 12342–12350.

McIlraith, S. (1998). Explanatory diagnosis: conjecturing actions to explain observations. In *Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR 1998)*, pp. 167–177, Trento, I. Morgan Kaufmann, S. Francisco, CA.

Metodi, A., & Codish, M. (2012). Compiling finite domain constraints do SAT with BEE. *Theory and Practice of Logic Programming*, *12*(4-5), 465–483.

Metodi, A., Codish, M., & Stuckey, P. (2013). Boolean equi-propagation for concise and efficient SAT encodings of combinatorial problems. *Journal of Artificial Intelligence Research*, *46*, 303–341.

Metodi, A., Stern, R., Kalech, M., & Codish, M. (2014). A novel SAT-based approach to model based diagnosis. *Journal of Artificial Intelligence Research*, *51*, 377–411.

Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence Journal*, *267*, 1–38.

Moreira, N., Nabais, D., & Reis, R. (2010). State elimination ordering strategies: some experimental results. In McQuillan, I., & Pighizzinir, G. (Eds.), *12th International Workshop on Descriptional Complexity of Formal Systems (DCFS 2010), Saskatoon, Canada, August 2010*, Vol. 31, pp. 139–148. Open Publishing Association. DOI: 10.4204/EPTCS.31.16.

Pencolé, Y., & Cordier, M. (2005). A formal framework for the decentralized diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, *164*(1–2), 121–170.

Pencolé, Y., Steinbauer, G., Mühlbacher, C., & Travé-Massuyès, L. (2018). Diagnosing discrete event systems using nominal models only. In Zanella, M., Pill, I., & Cimatti, A. (Eds.), *28th International Workshop on Principles of Diagnosis (DX'17)*, Vol. 4, pp. 169–183. Kalpa Publications in Computing.

Pencolé, Y., Subias, A., & Coquand, C. (2021). A model checking method to solve the event pattern diagnosis problem in safe labeled time Petri nets. In *32nd International Workshop on Principles of Diagnosis (DX-21)*, Hamburg, Germany.

Pill, I., & Quaritsch, T. (2012). An LTL SAT encoding for behavioral diagnosis. In *Twenty-Third International Workshop on Principles of Diagnosis (DX 2012)*, pp. 67–74.

Pill, I., & Quaritsch, T. (2013). Behavioral diagnosis of LTL specifications at operator level. In *Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pp. 1053–1059. AAAI Press.

Pill, I., & Wotawa, F. (2018). Automated generation of (F)LTL oracles for testing and debugging. *Journal of Systems and Software*, *139(C)*, 124–141.

Pill, I., & Wotawa, F. (2019). Extending automated FLTL test oracles with diagnostic support. In *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pp. 354–361.

Pnueli, A. (1977). The temporal logic of programs. In *8th Annual Symposium on Foundations of Computer Science*, SFCS '77, pp. 46–57, Washington, DC, USA. IEEE Computer Society.

Rabin, M., & Scott, D. (1959). Finite automata and their decision problems. *IBM Journal of Research and Development*, *3*(2), 114–125.

Ran, N., Su, H., Giua, A., & Seatzu, C. (2018). Codiagnosability analysis of bounded Petri nets. *IEEE Transactions on Automatic Control*, *63*(4), 1192–1199.

Reggia, J., Nau, D., & Wang, P. (1983). Diagnostic expert systems based on a set covering model. *International Journal of Man-Machine Studies*, *19*(5), 437–460. DOI: 10.1016/S0020-7373(83)80065-0.

Reggia, J., Nau, D., & Wang, P. (1985). A formal model of diagnostic inference. i. problem formulation and decomposition. *Information Sciences*, *37*(1-3), 227–256. DOI: 10.1016/0020-0255(85)90015-5.

Reiter, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, *32*(1), 57–95.

Ribeiro, M., Singh, S., & Guestrin, C. (2016). "Why should I trust you?" explaining the predictions of any classifier. In *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016), San Francisco, CA, USA, August 2016*, pp. 1135–1144, New York, NY, USA. Association for Computing Machinery.

Sakarovitch, J. (2009). *Elements of Automata Theory*. Cambridge University Press.

Sampath, M., Lafortune, S., & Teneketzis, D. (1998). Active diagnosis of discrete-event systems. *IEEE Transactions on Automatic Control*, *43*(7), 908–929.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., & Teneketzis, D. (1995). Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, *40*(9), 1555–1575.

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

Schuppan, V. (2012). Towards a notion of unsatisfiable and unrealizable cores for LTL. *Sci. Comput. Program.*, *77*(7–8), 908–939.

Shortliffe, E. (1976). *Computer-Based Medical Consultations: MYCIN*. American Elsevier, New York, NY.

Siddiqi, S., & Huang, J. (2007). Hierarchical diagnosis of multiple faults. In *20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pp. 581–586, Hyderabad, India.

Siddiqi, S., & Huang, J. (2011). Sequential diagnosis by abstraction. *Journal of Artificial Intelligence Research*, *41*, 329–365.

Stern, R., Kalech, M., Rogov, S., & Feldman, A. (2017). How many diagnoses do we need?. *Artificial Intelligence*, *248*, 26–45.

Struss, P. (1997). Fundamentals of model-based diagnosis of dynamic systems. In *Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 1997)*, pp. 480–485, Nagoya, Japan.

Torasso, P., & Torta, G. (2006). Model-based diagnosis through OBDD compilation: A complexity analysis. In Stock, O., & Schaerf, M. (Eds.), *Reasoning, Action and Interaction in AI Theories and Systems*, Vol. 4155 of *Lecture Notes in Computer Science*, pp. 287–305. Springer, Berlin, Heidelberg.

Trerotola, S. (2022). Sviluppo e sperimentazione di algoritmi per la diagnosi di guasti temporali in sistemi ad eventi discreti. Master's thesis, Department of Information Engineering, University of Brescia, Brescia, Italy.

Tripakis, S. (2002). Fault diagnosis for timed automata. In Damm, W., & Olderog, E.-R. (Eds.), *Formal Techniques in Real-Time and Fault-Tolerant Systems*, Vol. 2469 of *Lecture Notes in Computer Science*, pp. 205–221. Springer, Berlin, Heidelberg.

Watson, B. (1995). A taxonomy of finite automata minimization algorithms. Computing Science Note 93/44.

Yin, X., & Lafortune, S. (2017). On the decidability and complexity of diagnosability for labeled Petri nets. *IEEE Transactions on Automatic Control*, *62*(11), 5931–5938.

Yoo, T.-S., & Garcia, H. (2009). Event counting of partially-observed discrete-event systems with uniformly and nonuniformly bounded diagnosis delays. *Discrete Event Dynamic Systems*, *19*(2), 167–187. DOI: 10.1007/s10626-008-0056-1.