Logical Methods in Computer Science
Vol. 11(4:??)2015, pp. 1–33
www.lmcs-online.org

Submitted Jan. 13, 2014
Published Oct. ??, 2015

FORMAL DESIGN OF ASYNCHRONOUS FAULT DETECTION AND IDENTIFICATION COMPONENTS USING TEMPORAL EPISTEMIC LOGIC

MARCO BOZZANO, ALESSANDRO CIMATTI, MARCO GARIO, AND STEFANO TONETTA

Fondazione Bruno Kessler, Trento, Italy
e-mail address: {bozzano, cimatti, gario, tonettas}@fbk.eu

Abstract. Autonomous critical systems, such as satellites and space rovers, must be able to detect the occurrence of faults in order to ensure correct operation. This task is carried out by Fault Detection and Identification (FDI) components, that are embedded in those systems and are in charge of detecting faults in an automated and timely manner by reading data from sensors and triggering predefined alarms.

The design of effective FDI components is an extremely hard problem, also due to the lack of a complete theoretical foundation, and of precise specification and validation techniques.

In this paper, we present the first formal approach to the design of FDI components for discrete event systems, both in a synchronous and asynchronous setting. We propose a logical language for the specification of FDI requirements that accounts for a wide class of practical cases, and includes novel aspects such as maximality and trace-diagnosability. The language is equipped with a clear semantics based on temporal epistemic logic, and is proved to enjoy suitable properties. We discuss how to validate the requirements and how to verify that a given FDI component satisfies them. We propose an algorithm for the synthesis of correct-by-construction FDI components, and report on the applicability of the design approach on an industrial case-study coming from aerospace.

1. Introduction

The operation of complex critical systems (e.g., trains, satellites, cars) increasingly relies on the ability to detect when and which faults occur during operation. This function, called Fault Detection and Identification (FDI), provides information that is vital to drive the containment of faults and their recovery. This is especially true for fail-operational systems, where the occurrence of faults should not compromise the ability to carry on critical functions, as opposed to fail-safe systems, where faults are typically handled by going to a safe state. FDI is often carried out by dedicated modules, called FDI components, running

2012 ACM CCS: [**Theory of computation**]: Logic—Logic and verification — Verification by model checking; [**Computing methodologies**]: Artificial intelligence—Knowledge representation and reasoning—Temporal reasoning; Artificial intelligence—Reasoning about belief and knowledge / Causal reasoning and diagnostics; [**Hardware**]: Robustness—Fault tolerance / Hardware reliability.

Key words and phrases: Fault Detection and Identification; Diagnoser Synthesis; Model Checking; Temporal Epistemic Logic.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.2168/LMCS-11(4:??)2015

in parallel with the system. An FDI component, hereafter also referred to as a diagnoser, processes sequences of observations, made available by predefined sensors, and is required to trigger a set of predefined alarms in a timely and accurate manner. The alarms are then used by recovery modules to guarantee the survival of the system without requiring external control. Faults are often not directly observable. Their occurrence can only be inferred by observing the effects that they have on the observable parts of the system. Moreover, faults may have complex dynamics, and may interact with each other in complex ways.

For these reasons, the design of FDI components is a very challenging task, and also a practical problem, as witnessed by multiple Invitations To Tender issued by the European Space Agency [Eur10, Eur11, Eur13]. The current methodologies lack a comprehensive theoretical foundation, and do not provide clear and effective specification and validation techniques and tools. Most approaches asses the quality of an FDI component based on simulation and quantitative analysis [FKN$^+$10], that do not start from a specification of the behavior the the FDI needs to satisfy. This leads to a uniform treatment of all faults, while in general some faults are more important then others, and in many cases we are not interested in the specific fault characteristics but only to know that the fault occurred in a given part of the system (isolation). As a consequence, the design often results in very conservative assumptions, so that the overall system features sub-optimal behaviors, and it is not trusted during critical phases.

The goal of this paper is to propose a formal foundation to support the design of FDI components. We provide a way to specify FDI components, and cover the following problems: (i) validation of an FDI component specification, (ii) verification of a given FDI component with respect to a given specification, and (iii) automated synthesis of an FDI component from a given specification.

The specification of an FDI component is tackled by introducing a *pattern-based* language. Intuitively, an FDI component is specified by stating the observable signals (the inputs of the FDI component), the desired alarms (in terms of the unobservable state), and by defining the relation between the two. The language supports various forms of delay (exact, finite, bounded) between the occurrence of faults and the raising of the corresponding alarm. The patterns are given a formal semantics expressed in terms of epistemic temporal logic [HV89], where the *knowledge* operator is used to express the certainty of a condition, based on the available observations. The formalization encodes properties such as *alarm correctness* and *alarm completeness*. Correctness states that whenever an alarm is raised by the FDI component, then its associated triggering condition did occur; completeness states that if an alarm is not raised, then either the associated condition did not occur, or it would have been impossible to detect it, given the available observations. Moreover, we precisely characterize two aspects that are important for the specification of FDI requirements. The first one is the *diagnosability* of the plant, i.e., whether the sensors convey enough information to detect the required conditions. We explain how to deal with non-diagnosable plants by introducing a more fine-grained concept of *trace diagnosability*, where diagnosability is localized to individual traces. Most of the state of the art focuses on the fact that the system is diagnosable for any execution. However, in practice, this is rarely the case, since usually the plant is diagnosable in many situations but not in all of them. The classic example is the one of a burnt light-bulb, of which we cannot say anything until we try to turn it on. In this case, we would like to build a diagnoser that can raise the alarm whenever there is no ambiguity on whether the light bulb is burnt. Therefore, we introduce the concept

of trace diagnosability, intuitively accepting the fact that the plant might not always be diagnosable.

The second important concept that we introduce is *maximality*. A diagnoser is maximal if it is able to raise an alarm as soon as and whenever possible. This, in particular, means that in all traces that are diagnosable, a maximal diagnoser needs to raise the alarm.

The approach provides a full account of synchronous and asynchronous perfect-recall semantics for the epistemic operator. We show that the specification language correctly captures the formal semantics and we clearly define the relation between diagnosability, maximality and correctness.

Within our setting, the validation of a diagnoser specification is reduced to validity checking in temporal epistemic logic, while the verification of a given diagnoser is mapped to model checking for a temporal epistemic logic. As for synthesis, we propose an algorithm that is proved to generate correct-by-construction diagnosers.

From the practical standpoint, the applicability of the design approach has been demonstrated on two projects funded by the European Space Agency [AUT, FAM]. The paper actually provides the conceptual foundation underlying a design tool-set [ANY$^+$12, BBC$^+$14a, BBC$^+$14b], which has been applied to the specification, verification and synthesis of an FDI component for a satellite.

Finally, please note the deep difference between the design of FDI components and most diagnosis [dKK04] approaches. In most settings, diagnosis systems can benefit from powerful computing platforms. Partial diagnoses are typically acceptable, and can be complemented by further (post-mortem) inspections. This is typical of approaches that rely on logical reasoning engines (e.g., SAT solvers [GARK07]). Other approaches [HD05, SSL$^+$95, Sch04] rely on knowledge compilation to reduce the on-line complexity. An FDI component, on the contrary, runs on-board (as part of the on-line control strategy), and is subject to restrictions of various nature, such as timing and computation power. FDI design thus requires a deeper theory, which accounts for the issues of delay in raising the alarms, trace diagnosability, and maximality. Moreover, it becomes crucial to be able to verify and certify the effectiveness of the system, since it might not be possible to change it after deployment.

This paper is structured as follows. Section 2 provides some introductory background and introduces our running example. Section 3 formalizes the notion of FDI. Section 4 presents the specification language. In Section 5, we discuss how to validate the requirements, and how to verify an FDI component with respect to the requirements. In Section 6, we present an algorithm for the synthesis of correct-by-construction FDI components. The results of evaluating our approach in an industrial setting are presented in Section 7. Section 8 compares our work with previous related works. In Section 9, we draw some conclusions and outline the directions for future work.

2. Background

2.1. **Labeled Transition Systems.** In order to model the plant and the FDI, we use a symbolic representation of *Labeled Transition Systems* (LTS). Control locations and data are represented by variables, while sets of states and transitions are represented by formulas, and transitions are labeled with explicit events.

Given a set of variables X and a (finite) domain $\mathcal{U}$ of values, an assignment to X is a mapping from the set X to the set $\mathcal{U}$. We use $\Sigma(X)$ to denote the set of assignments to X.

Given an assignment $a \in \Sigma(X)$ and $X_1 \subseteq X$, we use $a_{|X_1}$ to denote the projection of a over X_1. We use $\mathcal{F}(X)$ to denote the set of propositional formulas over X.

Definition 2.1 (LTS)**.** A *Labeled Transition System* is a tuple $S = \langle V, E, I, \mathcal{T} \rangle$, where:

- V is the set of state variables;
- E is the set of events;
- $I \in \mathcal{F}(V)$ is a formula over V defining the initial states;
- $\mathcal{T} : E \to \mathcal{F}(V \cup V')$ maps an event $e \in E$ to a formula over V and V' defining the transition relation for e (with V' being the next version of the state variables).

A *state* s is an assignment to the state variables V (i.e., $s \in \Sigma(V)$). We denote by s' the corresponding assignment to V'. A transition labeled with e is a pair of states $\langle s, s' \rangle$ such that $s, s' \models \mathcal{T}(e)$. A *trace* of S is a sequence $\sigma = s_0, e_0, s_1, e_1, s_2, \ldots$ alternating states and event such that s_0 satisfies I and, for each $k \geq 0$, $\langle s_k, s_{k+1} \rangle$ satisfies $\mathcal{T}(e_k)$. Note that we consider infinite traces only, and w.l.o.g. we assume the system to be dead-lock free. Given $\sigma = s_0, e_0, s_1, e_1, s_2, \ldots$ and an integer $k \geq 0$, we denote by σ^k the finite prefix $s_0, e_0, \ldots, s_k$ of σ containing the first $k+1$ states. We denote by $\sigma[k]$ the $k+1$-th state s_k. We say that s is *reachable* in S iff there exists a trace σ of S such that $s = \sigma[k]$ for some $k \geq 0$.

We say that S is deterministic iff:

(i) there is one initial state (i.e., there exists a state s such that $s \models I$ and, for all t, if $t \models I$, then $s = t$);
(ii) for every reachable state s, for every event e, there is one successor (i.e., there exists s' such that $\langle s, s' \rangle \models \mathcal{T}(e)$ and, for all t', if $\langle s, t' \rangle \models \mathcal{T}(e)$, then $s' = t'$).

Definition 2.2 (Synchronous Product)**.** Let

$$S^1 = \langle V^1, E^1, I^1, \mathcal{T}^1 \rangle \text{ and } \; S^2 = \langle V^2, E^2, I^2, \mathcal{T}^2 \rangle$$

be two transition systems with $E^1 = E^2 = E$. We define the *synchronous product* $S^1 \times S^2$ as the transition system $\langle V^1 \cup V^2, E, I^1 \wedge I^2, \mathcal{T} \rangle$ where, for every $e \in E$, $\mathcal{T}(e) = \mathcal{T}^1(e) \wedge \mathcal{T}^2(e)$. Every state s of $S^1 \times S^2$ can be considered as the product $s^1 \times s^2$ such that $s^1 = s_{|V^1}$ is a state of S^1 and $s^2 = s_{|V^2}$ is a state of S^2. Similarly, every trace σ of $S^1 \times S^2$ can be considered as the product $\sigma^1 \times \sigma^2$ where σ^1 is a trace of S^1 and σ^2 is a trace of S^2.

Definition 2.3 (Asynchronous Product)**.** Let

$$S^1 = \langle V^1, E^1, I^1, \mathcal{T}^1 \rangle \text{ and } \; S^2 = \langle V^2, E^2, I^2, \mathcal{T}^2 \rangle$$

be two transition systems. We define the *asynchronous product* $S^1 \otimes S^2$ as the transition system $\langle V^1 \cup V^2, E^1 \cup E^2, I^1 \wedge I^2, \mathcal{T} \rangle$ where:

- for every $e \in E^1 \setminus E^2$, $\mathcal{T}(e) = \mathcal{T}^1(e) \wedge \mathit{frame}(V^2 \setminus V^1)$.
- for every $e \in E^2 \setminus E^1$, $\mathcal{T}(e) = \mathcal{T}^2(e) \wedge \mathit{frame}(V^1 \setminus V^2)$.
- for every $e \in E^1 \cap E^2$, $\mathcal{T}(e) = \mathcal{T}^1(e) \wedge \mathcal{T}^2(e)$.

where $\mathit{frame}(X)$ stands for $\bigwedge_{x \in X} x' = x$ and is used to represent the fact that while one transition system moves on a local event, the other transition system does not change its local state variables. Every state s of $S^1 \otimes S^2$ can be considered as the product $s^1 \otimes s^2$ such that $s^1 = s_{|V^1}$ is a state of S^1 and $s^2 = s_{|V^2}$ is a state of S^2. If either S^1 or S^2 is deterministic, also every trace σ of $S^1 \otimes S^2$ can be considered as the product $\sigma^1 \otimes \sigma^2$ where σ^1 is a trace of S^1 and σ^2 is a trace of S^2 (more in general, the product of two traces produces a set of traces due to different possible interleavings).

In general, composing two systems can reduce the behaviors of each system and introduce deadlocks. However, given two systems that do not share any state variable (e.g., the diagnoser and the plant), if one of the systems is deterministic (the diagnoser) then it cannot alter the behavior of the second (the plant).

Notice that the synchronous product coincides with the asynchronous case when the two sets of events coincide.

2.2. **Linear Temporal Logic.** We now present a Linear Temporal Logic extended with past operators [Pnu77, LMS02, LPZ85], in the following simply referred to as LTL. A formula in LTL over variables V and events E is defined as

$$\beta ::= p \mid e \mid \beta \wedge \beta \mid \neg\beta \mid O\beta \mid Y\beta \mid \beta S\beta \mid G\beta \mid F\beta \mid X\beta \mid \beta U\beta$$

where p is a predicate over $\mathcal{F}(V)$ and $e \in E$. Intuitively, p are the propositions over the state of the LTS, while e represents an event.

Given a trace $\sigma = s_0, e_0, s_1, e_1, s_2, \ldots$, the semantics of LTL is defined as follows:

- $\sigma, i \models p$ iff $s_i \models p$
- $\sigma, i \models e$ iff $e_i = e$
- $\sigma, i \models \beta_1 \wedge \beta_2$ iff $\sigma, i \models \beta_1$ and $\sigma, i \models \beta_2$
- $\sigma, i \models \neg\beta$ iff $\sigma, i \not\models \beta$
- Once: $\sigma, i \models O\beta$ iff $\exists j \leq i.\ \sigma, j \models \beta$
- Yesterday: $\sigma, i \models Y\beta$ iff $i > 0$ and $\sigma, i-1 \models \beta$
- Since: $\sigma, i \models \beta_1 S \beta_2$ iff there exists $j \leq i$ such that $\sigma, j \models \beta_2$ and for all k, $j < k \leq i$, $\sigma, k \models \beta_1$
- Finally: $\sigma, i \models F\beta$ iff $\exists j \geq i.\ \sigma, j \models \beta$
- Globally: $\sigma, i \models G\beta$ iff $\forall j \geq i.\ \sigma, j \models \beta$
- Next: $\sigma, i \models X\beta$ iff $\sigma, i+1 \models \beta$
- Until: $\sigma, i \models \beta_1 U \beta_2$ iff there exists $j \geq i$ such that $\sigma, j \models \beta_2$ and for all k, $i \leq k < j$, $\sigma, k \models \beta_1$.

Given an LTS $S = \langle V, E, I, \mathcal{T} \rangle$, $S \models \beta$ iff for every trace σ of S, $\sigma, 0 \models \beta$.

Notice that $Y\beta$ is always false in the initial state, and that we use a reflexive semantics for the operators U, F, G, S and O. We use the abbreviations $Y^n\beta = YY^{n-1}\beta$ (with $Y^0\beta = \beta$), $O^{\leq n}\beta = \beta \vee Y\beta \vee \cdots \vee Y^n\beta$ and $F^{\leq n}\beta = \beta \vee X\beta \vee \cdots \vee X^n\beta$.

2.3. **Partial Observability.** A *partially observable* LTS is an LTS $S = \langle V, E, I, \mathcal{T} \rangle$ extended with a set $E_o \subseteq E$ of observable events.

We consider here only observations on events. In practice, observation on states are common and relevant. However, dealing with them in the asynchronous setting makes the formalism less clear. Therefore, we limit ourselves to observations on events and whenever observations on state variables are needed, such as sensor readings, we incorporate them in the events as done in [SSL+96].

The observable part of the prefix σ^k of a trace σ is defined recursively as follows: $obs(\sigma^0) = \epsilon$ (empty sequence); if $e \in E_o$, then $obs(\sigma^k, e, s) = obs(\sigma^k), e$; if $e \notin E_o$, then $obs(\sigma^k, e, s) = obs(\sigma^k)$.

Definition 2.4 (Observation Point)**.** We say that i is an *observation point* for σ, denoted by $ObsPoint(\sigma, i)$, iff the last event of σ^i is observable, i.e., iff $\sigma^i = \sigma', e, s$ for some σ', e, s and $e \in E_o$.

The notion of two traces being observationally equivalent requires that the two traces end both or neither in an observation point. This captures the idea that a trace ending in an observation point can be distinguished from the same trace extended with local unobservable steps. In other terms, an observer can distinguish the instant in which it is observing and an instant right after.

Definition 2.5 (Observational Equivalence)**.** We say that $((\sigma_1, i), (\sigma_2, j)) \in \mathit{ObsEq}$ if and only if:
- $\mathit{ObsPoint}(\sigma_1, i)$ iff $\mathit{ObsPoint}(\sigma_2, j)$, and
- $\mathit{obs}(\sigma_1^i) = \mathit{obs}(\sigma_2^j)$.

2.4. **Temporal Epistemic Logic.** Epistemic logic has been used to describe and reason about knowledge of agents and processes. There are several ways of extending epistemic logic with temporal operators. We use the logic KL_1 [HV89], extended with past operators. A formula in KL_1 is defined as

$$\beta ::= p \mid e \mid \beta \wedge \beta \mid \neg\beta \mid O\beta \mid Y\beta \mid \beta S \beta \mid F\beta \mid X\beta \mid \beta U \beta \mid G\beta \mid K\beta$$

KL_1 can be seen as extension of LTL with past operators, with the addition of the epistemic operator K. The intuitive semantics of $K\beta$ is that the reasoner *knows* that β holds in a state of a trace σ, by using only the observable information. This means that $K\beta$ holds iff β holds in all situations that are observationally equivalent. Therefore, while in LTL the interpretation of a formula is local to a single trace, in KL_1 the semantics of the K operator quantifies over the set of indistinguishable traces. Given a trace σ_1 of a partially observable LTS, the semantics of K is formally defined as:

$$\sigma_1, i \models K\beta \text{ iff } \forall \sigma_2, \forall j.\ \text{ if } ((\sigma_1, i), (\sigma_2, j)) \in \mathit{ObsEq} \text{ then } \sigma_2, j \models \beta.$$

$K\beta$ holds at time i in a trace σ_1 iff β holds in all traces that are observationally equivalent to σ_1 up to time i. Note that, due to the asynchronous nature of the observations, two traces of different length might lead to the same observable trace. This definition implicitly forces *perfect-recall* in the semantics of the epistemic operator, since we define the epistemic equivalence between traces and not between states.

In many situations, we are interested in considering formulas only at observation points. We do so by introducing the following abbreviation.

Definition 2.6 (Observed)**.** If E_o is the set of observable events, given a formula ϕ, we use $\llcorner\phi\lrcorner$ (read "Observed ϕ") as abbreviation for $\phi \wedge Y \bigvee_{e \in E_o} e$.

2.5. **Running Example.** The *Battery Sensor System* (BSS) (Figure 1) will be our running example. The BSS provides a redundant reading of the sensors to a device. Internal batteries provide backup in case of failure of the external power supply. The safety of the system depends on both of the sensors providing a correct reading. The system can work in three different operational modes: *Primary*, *Secondary 1* and *Secondary 2*. In Primary mode, each sensor is powered by the corresponding battery. In the Secondary modes, instead, both sensors are powered by the same battery; e.g., during Secondary 1, both Sensor 1 and Sensor 2 are powered by Battery 1. The Secondary modes are used to keep the system operational in case of faults. However, in the secondary modes, the battery in use will discharge faster.

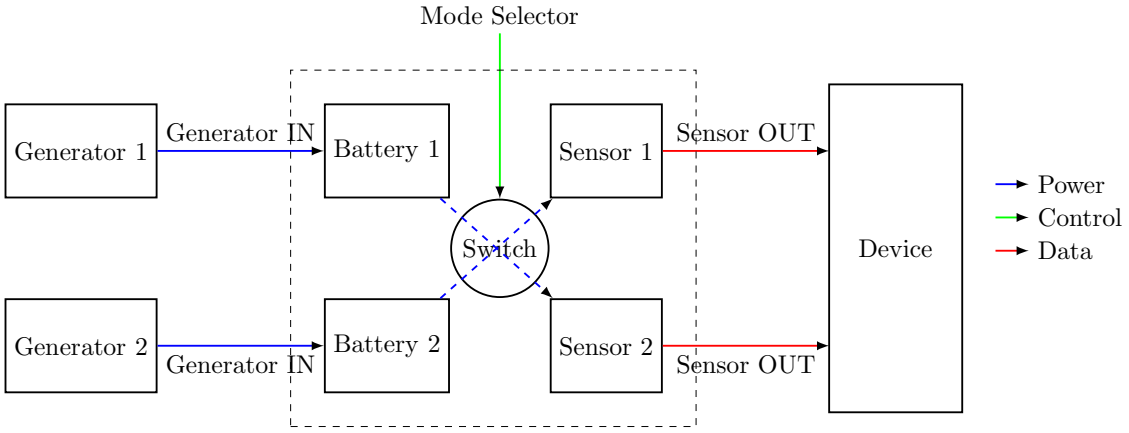


FIGURE 1. Running Example (Battery Sensor System)

We consider two possible recovery actions: i) Switch Mode, or ii) Replace the Battery-Sensor Block (the dotted block in Figure 1). In order to decide which recovery to apply, we are going to define a set of requirements connecting the faults to alarms. The faults and observable information of the system are shown in Figure 2.

This example is particularly interesting because we can define two sources of delay: the batteries, and the device resilience to wrong inputs. The batteries provide a buffer for supplying power to the sensors. The size of this buffer is determined by the capacity of the battery, the initial charge, and the discharge rate. For the device, we assume that two valid sensor readings are required for optimal behavior, however, we can work in degraded mode with only one valid reading for a limited amount of time. The device will stop working if both sensors are providing invalid readings, or if one sensor has been providing an invalid reading for too long.

Both a synchronous and asynchronous version of this model are possible. In the asynchronous model, we have an event for each possible combination of observations (e.g., "Mode Primary & Battery 1 Low"). In the synchronous model, we also have an additional observable event (*tick*) that represents the passing of time in the absence of any observable event. This event forces the synchronization of the plant with the diagnoser. The key difference between the synchronous and asynchronous setting is the amount of information that we can infer in this particular case. For example, if we know the initial charge level of a battery, and we know its discharge rate (given by the operational mode), then at each point in time we can infer the current charge of the battery. By comparing our expectation with the available information, we can detect when something is not behaving as expected. Unfortunately, there are practical settings in which the assumption of synchronicity is not realistic. Therefore, our approach accounts for both the synchronous and asynchronous models.

Observables	**Possible Values**
Mode	Primary, Secondary 1, Secondary 2
Battery Level {1, 2}	High, Mid, Low
Sensors Delta	Zero, Non-Zero ($\|S1.Out - S2.Out\| = 0$)
Device Status	On, Off

Component	**Faults**
Generator	Off ($G1_{Off}, G2_{Off}$)
Battery	Leak ($B1_{Leak}, B2_{Leak}$)
Sensor	Wrong Output ($S1_{WO}, S2_{WO}$)

FIGURE 2. Observables and Faults Summary

To provide a better understanding of how the running example behaves, we provide the LTS of each of the components. Figure 3 shows the LTS of the generator and switch. We assume that the only way the generator can turn off is if a fault event occurs, thus the model of the generator is rather simple. Also the switch features a rather simple model, where the labels *toS1* and *toS2* are defined as:

- toS1: Mode=Secondary1 ∧ Battery1.Double ∧ Battery2.Offline
- toS1: Mode=Secondary2 ∧ Battery1.Offline ∧ Battery1.Double

thus they drive the change in operational mode of the batteries.

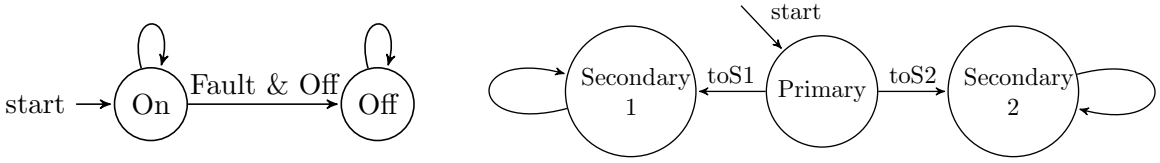


FIGURE 3. Generator (Left) and Switch (Right) LTS

Figure 4 shows two slightly more complex components: the sensor and the device. The sensor periodically outputs a good or a bad reading depending on the state it is in. Notice that the transition from a good to a bad state can occur either because of a fault (Wrong Output in Figure 2) or because the battery connected to the sensor has no charge ($Batt.c = 0$), notice, in particular, that both events are not observable. The device instead has two main transitions. The *stay* is defined as $S1.Value = S2.Value \wedge Delta = Zero$, while *degrade* represents a discrepancy in the reading from the sensor that will eventually lead to the device stopping: $(S1.Value \neq S2.Value) \wedge Delta =$*Non-Zero*. The values of the sensors are not observable, but their difference is observable via the $Delta$ variable. Intuitively, the device has an intermediate state that works as a buffer, before reaching the final *Off* state.

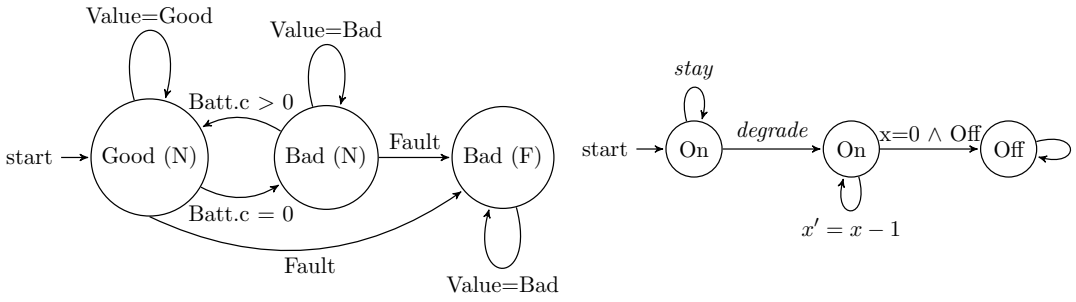


FIGURE 4. Sensor (Left) and Device (Right) LTS

The most complex component, the battery, is presented in Figure 5. Vertical transitions indicate a change in operational mode of the battery. The left half of the LTS indicates that the generator is working and feeding the battery (thus charging it) while the right half shows that the battery is not charging. Additionally, the two central columns describe the faulty behavior of the battery. This information is represented also in each state. Each

state has an additional self-loop (not in the picture) denoting the update of the charge of the battery, following the update rule:

$$charge' = (charge + recharge - (load + leak)) \bmod C$$

where C is the capacity, and the other variables depend on the state:

(1) Charging: $recharge = 1$, Not Charging: $recharge = 0$
(2) Primary: $load = 1$, Offline: $load = 0$, Double: $load = 2$
(3) Nominal: $leak = 0$, Faulty: $leak = 2$

Thus the charge of the battery can change from $+1$ (Nominal, Offline, Charging) to -4 (Faulty, Double, Not Charging), while staying within the bound $[0, Capacity]$.

Every time the update of the charge causes the charge to pass a threshold, the transition raises the observable event: Low, Mid, $High$. These events indicate when the charge of the battery is above 20%, 50% and 80%. All other transitions are not observable. These transitions have been omitted from the figure to make it more readable.

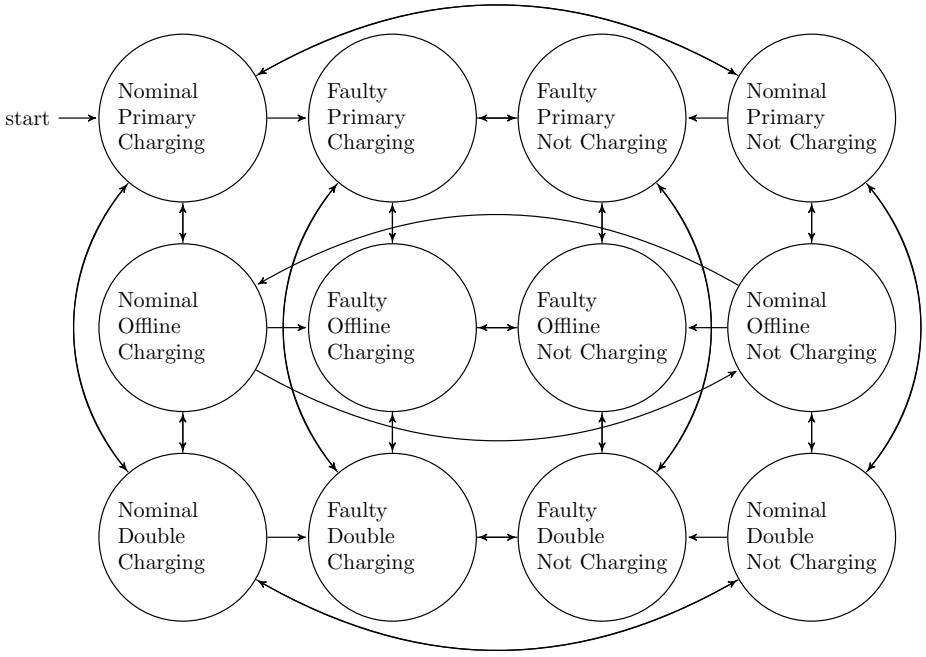


FIGURE 5. Battery LTS

3. Formal Characterization

3.1. **Diagnoser.** In our general setting, a plant is connected to components for Fault Detection and Isolation, and for Fault Recovery, as depicted in Figure 6. The role of FDI is to collect and analyze the observable information from the plant, and to turn on suitable alarms associated with (typically unobservable) relevant conditions. The Fault Recovery component is intended to apply suitable reconfiguration actions based on the alarms in

input. Recovery is beyond the scope of this work; we consider a *system* composed of the plant and the FDI component.

An FDI component (also called diagnoser in the following) is a machine D that synchronizes with observable traces of the plant P. D has a set $\mathcal{A}$ of alarms that are activated in response to the monitoring of P. Different mechanisms to connect a diagnoser to a plant are possible. In the synchronous case, the plant is assumed to convey to the diagnoser information at a fixed rate (including state sampling and values for event ports). This model is adopted, for example, in [BCGT14, CPC03]. In this paper we focus on the more general model of asynchronous case, where the diagnoser reacts to the observable events in the plant [1].

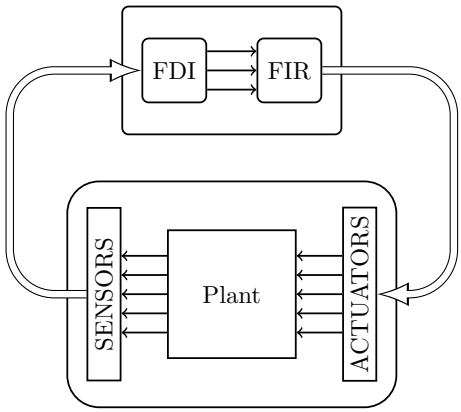


FIGURE 6. Integration of the FDIR and Plant

Definition 3.1 (Diagnoser)**.** Given a set $\mathcal{A}$ of alarms and a partially observable plant $P = \langle V^P, E^P, I^P, \mathcal{T}^P, E^P_o \rangle$, a diagnoser is a deterministic LTS $D(\mathcal{A}, P) = \langle V^D, E^D, I^D, \mathcal{T}^D \rangle$ such that $E^P_o = E^D$, $V^P \cap V^D = \emptyset$ and $\mathcal{A} \subseteq V^D$.

When clear from the context, we use D to indicate $D(\mathcal{A}, P)$. We assume that the events of the diagnoser coincide with the observable events of the plant. This means that the diagnoser does not have internal transitions: every transition of the diagnoser is associated with an observable transition of the plant. We say that the alarm A is triggered when A is true after the diagnoser synchronized with the plant (i.e., when $\llcorner A \lrcorner$ is true).

Since the synchronous case is a particular case of the asynchronous composition, in the rest of the paper we assume that the plant and diagnoser are composed asynchronously: i.e., $D \otimes P$. Only observable events are used to perform synchronization.

The choice of using a deterministic diagnoser is driven by the following result, that makes it easier to understand how the diagnoser will react to the plant:

Definition 3.2 (Diagnoser Matching trace)**.** Given a diagnoser D of P and a trace σ_P of P, the diagnoser trace matching σ_P, denoted by $D(\sigma_P)$, is the trace σ of D such that $\sigma \otimes \sigma_P$ is a trace of $D \otimes P$.

Note that the notion of diagnoser matching trace is well defined because, since D is deterministic, there exists one and only one trace in D matching σ_P.

[1] The relation between the synchronous and the asynchronous combination is discussed in Section 8.1.

3.2. **Detection, Identification, and Diagnosis Conditions.** The first element for the specification of the FDI requirements is given by the conditions that must be monitored. Here, we distinguish between detection and identification, which are the two extreme cases of the diagnosis problem; the first deals with knowing whether a fault occurred in the system, while the second tries to identify the characteristics of the fault. Between these two cases there can be intermediate ones: we might want to restrict the detection to a particular sub-system, or identification among two similar faults might not be of interest.

The *detection* task is the problem of understanding when (at least) one of the components has failed. The *identification* task tries to understand exactly which fault occurred.

In the BSS every component can fail. Therefore the detection problem boils down to knowing that at least one of the generators, batteries or sensors is experiencing a fault. For identification, instead, we are interested in knowing whether a specific fault, (e.g., $G1_{Off}$) occurred. There are also intermediate situations (sometimes called *isolation*), in which we are not interested in distinguishing whether $G1_{Off}$ or $B1_{Leak}$ occurred, as long as we know that there is a problem in the power-supply chain.

FDI components are generally used to recognize faults. However, there is no reason to restrict our interest to faults. Recovery procedures might differ depending on the current state of the plant, therefore, it might be important to consider other unobservable information of the system. For example, we might want to estimate the charge level of a battery, or its discharge rate.

We call the condition of the plant to be monitored *diagnosis condition*, denoted by β. We assume that for any point in time along a trace execution of the plant (and therefore also of the system), β is either true or false based on what happened before that time point. Therefore, β can be an atomic condition (including faults), a sequence of atomic conditions, or Boolean combination thereof. If β is a fault, the fault must be identified; if β is a disjunction of faults, instead, it suffices to perform the detection, without identifying the exact fault.

Diagnosis condition	**Definition**
$\beta_{Generator1}$, $\beta_{Generator2}$	$G1_{Off}$, $G2_{Off}$
$\beta_{Battery1}$, $\beta_{Battery2}$	$B1_{Leak}$, $B2_{Leak}$
β_{PSU1}, β_{PSU2}	$G1_{Off} \vee B1_{Leak}$, $G2_{Off} \vee B2_{Leak}$
$\beta_{Batteries}$	$B1_{Leak} \vee B2_{Leak}$
$\beta_{Sensor1}$, $\beta_{Sensor2}$	$S1_{WO}$, $S2_{WO}$
$\beta_{Sensors}$	$S1_{WO} \vee S2_{WO}$
β_{BS}	$(S1_{WO} \vee S2_{WO}) \vee (B1_{Leak} \wedge B2_{Leak})$
β_{Seq}	$(B1_{Charge} < B2_{Charge}) \wedge O(B1_{Charge} \geq B2_{Charge})$
$\beta_{Charging}$	$Y((B1_{Charge} \leq 0) \wedge Y(B1_{Charge} > 0)$
$\beta_{Depleted}$	$(B1_{Charge} = 0) \vee (B2_{Charge} = 0)$

FIGURE 7. Diagnosis conditions for the BSS

Figure 7 shows several examples of diagnosis conditions for the BSS. Notice how we might be in complex situations such as knowing if the Battery-Sensor block is working (β_{BS}) or knowing some information on the evolution of the system (β_{Seq}, $\beta_{Charging}$). We use LTL operators to define those diagnosis conditions, but in general, we require that a diagnosis condition can be evaluated on a point in a trace by only looking at the trace prefix.

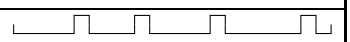
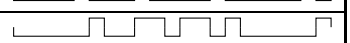
β	
$\textsc{ExactDel}(A, \beta, 2)$	
$\textsc{BoundDel}(A, \beta, 4)$	
$\textsc{FiniteDel}(A, \beta)$	

FIGURE 8. Examples of alarm responses to the diagnosis condition β.

3.3. **Alarm Conditions.** The second element of the specification of FDI requirements is the relation between a diagnosis condition and the raising of an alarm. This also leads to the definition of when the FDI is correct and complete with regard to a set of alarms.

An *alarm condition* is composed of two parts: the diagnosis condition and the delay. The delay relates the time between the occurrence of the diagnosis condition and the corresponding alarm. Although it might be acceptable that the occurrence of a fault can go undetected for a certain amount of time, it is important to specify clearly how long this interval can be. An alarm condition is a property of the system composed by the plant and the diagnoser, since it relates a condition of the plant with an alarm of the diagnoser. Thus, when we say that a diagnoser D of P satisfies an alarm condition, we mean that the traces of the system $D \otimes P$ satisfy it.

Interaction with industrial experts led us to identify three patterns of *alarm conditions*, which we denote by $\textsc{ExactDel}(A, \beta, d)$, $\textsc{BoundDel}(A, \beta, d)$, and $\textsc{FiniteDel}(A, \beta)$:

1. $\textsc{ExactDel}(A, \beta, d)$ specifies that whenever β is true, A must be triggered exactly d steps later and A can be triggered only if d steps earlier β was true; formally, for any trace σ of the system, if β is true along σ at the time point i, then $\llcorner A \lrcorner$ is true in $\sigma[i + d]$ (Completeness); if $\llcorner A \lrcorner$ is true in $\sigma[i]$, then β must be true in $\sigma[i - d]$ (Correctness).
2. $\textsc{BoundDel}(A, \beta, d)$ specifies that whenever β is true, A must be triggered within the next d steps and A can be triggered only if β was true within the previous d steps; formally, for any trace σ of the system, if β is true along σ at the time point i then $\llcorner A \lrcorner$ is true in $\sigma[j]$, for some $i \leq j \leq i + d$ (Completeness); if $\llcorner A \lrcorner$ is true in $\sigma[i]$, then β must be true in $\sigma[j']$ for some $i - d \leq j' \leq i$ (Correctness).
3. $\textsc{FiniteDel}(A, \beta)$ specifies that whenever β is true, A must be triggered in a later step and A can be triggered only if β was true in some previous step; formally, for any trace σ of the system, if β is true along σ at the time point i then $\llcorner A \lrcorner$ is true in $\sigma[j]$ for some $j \geq i$ (Completeness); if $\llcorner A \lrcorner$ is true in $\sigma[i]$, then β must be true along σ in some time point between 0 and i (Correctness).

Figure 8 provides an example of admissible responses for the various alarms to the occurrences of the same diagnosis condition β; note how in the case of $\textsc{BoundDel}(A, \beta, 4)$ the alarm can be triggered at any point as long as it is within the next 4 time-steps. Since A is a state variable and the diagnoser changes it only in response to synchronizations with the plant, every rising and falling edge of the alarm in the figure corresponds to an observation point.

Figure 9 contains a simple specification for our running example. There are two types of PSU (Power Supply Unit) alarms (that can be similarly defined for PSU 2). The first one defines multiple alarms, each having a different delay i. Let us assume that each battery has a capacity C of 10, and that this provides us with a delay of at most 10 time-units.

Pattern	**Description**
$\textsc{ExactDel}(PSU1_{Exact_i}, \beta_{PSU1}, i)$	Detect if the PSU 1 (Generator 1 + Battery 1) is broken, in order to switch to secondary mode
$\textsc{BoundDel}(PSU1_{Bound}, \beta_{PSU1}, C)$	Detect if the PSU (Generator 1 + Battery 1) was broken within the bound, in order to switch to secondary mode
$\textsc{BoundDel}(BS, \beta_{BS}, DC)$	Detect if the whole Battery-Sensor block is working incorrectly, in order to replace it
$\textsc{FiniteDel}(Discharged, \beta_{Depleted})$	Detect if any of the battery was ever completely discharged

FIGURE 9. Example Specification for the BSS

We can instantiate 10 alarms one for each $i \in [0, 10]$. Ideally, we want to detect the exact moment in which the PSU stop working. However, this might not be possible due to non-diagnosability. Therefore, we define a weaker version of the alarm ($PSU1_{Bound}$), in which we say that within the time-bound provided by the battery capacity (C) we want to know if the PSU stop working. In Section 5.1 we will prove that one alarm condition is weaker than the other. For most alarms, we specify what recovery can be applied to address the problem. In this way, our process of defining the alarms of interest is driven by the recovery procedures available. If there is no automated recovery for a given situation, time-bounds might not be relevant anymore. Therefore, we use alarms to collect information on the historical state of the system (e.g., *Discharged* alarm); notice, in fact, that FINITEDEL alarm have a permanent behavior, i.e., they can never be turned off.

3.4. **Diagnosability.** Given an alarm condition, we need to know whether it is possible to build a diagnoser for it. In fact, there is no reason in having a specification that cannot be realized. This property is called *diagnosability* and was introduced in [SSL+95].

In this section, we define the concept of diagnosability for the different types of alarm conditions. We proceed by first giving the definition of diagnosability in the traditional way (à la Sampath) in terms of observationally equivalent traces w.r.t. the diagnosis condition. Then, we prove that a plant P is diagnosable iff there exists a diagnoser that satisfies the specification.

Definition 3.3. Given a plant P and a diagnosis condition β, we say that $\textsc{ExactDel}(A, \beta, d)$ is diagnosable in P iff for all σ_1, i s.t. $\sigma_1, i \models \beta$ then $ObsPoint(\sigma_1, i+d)$ and for all σ_2, j, if $ObsEq((\sigma_1, i+d), (\sigma_2, j+d))$, then $\sigma_2, j \models \beta$.

Therefore, an exact-delay alarm condition is not diagnosable in P iff either there is no synchronization after d steps (note that this is not possible in the synchronous case) or there exists a pair of traces σ_1 and σ_2 such that for some $i, j \geq 0$, $\sigma_1, i \models \beta$, $ObsEq((\sigma_1, i+d), (\sigma_2, j+d))$, and $\sigma_2, j \not\models \beta$. We call such a pair a *critical pair*.

Definition 3.4. Given a plant P and a diagnosis condition β, we say that $\textsc{BoundDel}(A, \beta, d)$ is diagnosable in P iff forall σ_1, i s.t. $\sigma_1, i \models \beta$ there exists k s.t. $i \leq k \leq i + d$, $ObsPoint(\sigma_1, k)$ and for all σ_2, l, if $ObsEq((\sigma_1, k), (\sigma_2, l))$, then there exists j s.t. $l - d \leq j \leq l$ and $\sigma_2, j \models \beta$.

Intuitively, k, l denote points that are observationally equivalent and i, j denote the states where the condition occurred, and their relation is such that i and j do not occur more than d steps away from each other.

This definition takes into account occurrences of β that happened before i. Indeed, we need to check occurrences up to d states before and after i. Consider the two traces $\sigma_1 = apbqc$ and $\sigma_2 = aqbpc$, where a, b, c are observable events, and $\beta = p$. We can see that we can justify p in σ_1 by looking at the occurrence of p in σ_2 that is in the future. However, we cannot justify the p in σ_2 by just looking in the future, but we need to look in the past.

Definition 3.5. Given a plant P and a diagnosis condition β, we say that $\textsc{FiniteDel}(A, \beta)$ is diagnosable in P iff for all σ_1, i s.t. $\sigma_1, i \models \beta$ then there exist $k \geq i$ s.t. $ObsPoint(\sigma_1, k)$ and for all σ_2, l if $ObsEq((\sigma_1, k), (\sigma_2, l))$ then there exists $j \leq l$ $\sigma_2, j \models \beta$.

Definition 3.4 is a generalization of Sampath's definition of diagnosability:

Definition 3.6. (Diagnosability [SSL+95]) Given a plant P and a diagnosis condition β, we say that β is diagnosable in P iff there exists d s.t. for all σ, i, σ_2, l, $k \geq i + d$ if $\sigma_1, i \models \beta$ and $obs(\sigma_2^l) = obs(\sigma_1^k)$ then there exists $j \leq l$ s.t. $\sigma_2, j \models \beta$.

In [SSL+95] (specifically in Section II.A), Sampath et al. also assume that there are no cycles of unobservable events. This means that there is a d_u s.t. for all σ, i s.t. $\sigma, i \models \beta$ then there exists k s.t. $0 \leq k \leq d_u$ and $ObsPoint(\sigma, i + k)$.

Theorem 3.7. *Let P be a plant such that there is no cycle of unobservable events, and let p be a propositional formula, then p is diagnosable (as defined in 3.6) in P iff there exists d such that $\textsc{BoundDel}(A, Op, d)$ is diagnosable in P.*

Proof.

⇒) Assume that p is diagnosable in P. Consider a trace σ_1 such that for some $i \geq 0$, $\sigma_1, i \models Op$. Then, for some $0 \leq i' \leq i$, $\sigma_1, i' \models p$. By assumption, we know that there is a d s.t. for all $k \geq i' + d$ and any trace σ_2 and point l such that $obs(\sigma_2^l) = obs(\sigma_1^k)$ then $\sigma_2, j' \models p$ for some j', $j' \leq l$. Then $\sigma_2, j \models Op$ for all $j \geq j'$. Since this holds for any k and l, it holds also for the k and l that are observation points for σ_1 and σ_2. Let $d' = d + n_u$. Then there exists $k' < d'$ such that $ObsPoint(\sigma_1, i + k')$ and for all trace σ_2 and point l such that $ObsEq((\sigma_1, k'), (\sigma_2, l))$ then $\sigma_2, j' \models p$ for some j', $j' \leq l$. We can conclude that $\textsc{BoundDel}(A, Op, d')$ is diagnosable in P.

⇐) Assume that $\textsc{BoundDel}(A, Op, d)$ is diagnosable in P. Consider a trace σ_1 such that for some $i \geq 0$ $\sigma_1, i \models p$. Then $\sigma_1, i \models Op$. By assumption, there exists k, $i \leq k \leq i + d$ such that $ObsPoint(\sigma_1, k)$ and, for any trace σ_2 and point l such that $ObsEq((\sigma_1, k), (\sigma_2, l))$ then $\sigma_2, j \models Op$ for some $l - d \leq j \leq l$. Let us consider σ_2 and l such that $obs(\sigma_2^l) = obs(\sigma_1^k)$. Then for some $l' \leq l$ we have that $ObsPoint(\sigma_2, l')$ and therefore $ObsEq((\sigma_1, k), (\sigma_2, l'))$. Then $\sigma_2, j \models Op$ for some $l - d \leq j \leq l$. Thus $\sigma_2, j' \models p$ for some $j' \leq j$ and P is diagnosable. □

The following theorem shows that if a component satisfies the diagnoser specification then the monitored plant must be diagnosable for that specification. In Section 6 on synthesis we will show also the converse, i.e., if the specification is diagnosable then a diagnoser exists.

Theorem 3.8. *Let D be a diagnoser for P. If D satisfies an alarm condition then the alarm condition is diagnosable in P.*

Proof. By contradiction, suppose $\textsc{ExactDel}(A, \beta, d)$ is not diagnosable in P. Then either there exists a trace σ_1 with $\sigma_1, i \models \beta$ for some i such that $\mathit{ObsPoint}(\sigma_1, j)$ is false for all $j \geq i$ or there exists a critical pair. In the first case, A is not triggered and the diagnoser is not complete. Suppose there exists a critical pair of traces σ_1 and σ_2, i.e., for some $i, j \geq 0$ $\sigma_1, i \models \beta$, $\mathit{ObsPoint}(\sigma_1, i+d)$, $\mathit{ObsEq}((\sigma_1, i+d), (\sigma_2, j+d))$, and $\sigma_2, j \not\models \beta$. Since D is deterministic, $D(\sigma_1)$ and $D(\sigma_2)$ have a common prefix compatible with $\mathit{obs}(\sigma_1^{i+d}) = \mathit{obs}(\sigma_2^{j+d})$. If the diagnoser is complete then A is triggered in $D(\sigma_1) \otimes \sigma_1$ at position $i+d$, and so also in $D(\sigma_2) \otimes \sigma_2$ at position $j+d$, but in this way the diagnoser is not correct, which is a contradiction. If the diagnoser is correct, then A is not triggered in $D(\sigma_2) \otimes \sigma_2$ at position $j+d$, but so neither in $D(\sigma_1) \otimes \sigma_1$ at position $i+d$, but in this way the diagnoser is not complete, which is a contradiction.

Similarly, for $\textsc{FiniteDel}(A, \beta)$ and $\textsc{BoundDel}(A, \beta, d)$. □

The definition above of diagnosability might be stronger than necessary, since diagnosability is defined as a global property of the plant. Imagine the situation in which there is a critical pair and after removing this critical pair from the possible executions of the system, our system becomes diagnosable. This suggests that the system was "almost" diagnosable, and an ideal diagnoser would be able to perform a correct diagnosis in all the cases except one (i.e., the one represented by the critical pair). To capture this idea, we redefine the problem of diagnosability from a global property expressed on the plant, to a local property expressed on points of single traces.

Definition 3.9. Given a plant P, a diagnosis condition β and a trace σ_1 such that for some $i \geq 0$ $\sigma_1, i \models \beta$, we say that $\textsc{ExactDel}(A, \beta, d)$ is *trace diagnosable in* $\langle \sigma_1, i \rangle$ iff $\mathit{ObsPoint}(\sigma_1, i+d)$ and for any trace σ_2, for all $j \geq 0$ such that $\mathit{ObsEq}((\sigma_1, i+d), (\sigma_2, j+d))$, $\sigma_2, j \models \beta$.

Definition 3.10. Given a plant P, a diagnosis condition β, and a trace σ_1 such that for some $i \geq 0$ $\sigma_1, i \models \beta$, we say that $\textsc{BoundDel}(A, \beta, d)$ is *trace diagnosable in* $\langle \sigma_1, i \rangle$ iff there exists k s.t. $i \leq k \leq i+d$, $\mathit{ObsPoint}(\sigma_1, k)$, and for any σ_2, l if $\mathit{ObsEq}((\sigma_1, k), (\sigma_2, l))$, then there exists j s.t. $l - d \leq k \leq l$ and $\sigma_2, j \models \beta$.

Definition 3.11. Given a plant P, a diagnosis condition β, and a trace σ_1 such that for some $i \geq 0$, $\sigma_1, i \models \beta$, we say that $\textsc{FiniteDel}(A, \beta)$ is *trace diagnosable in* $\langle \sigma_1, i \rangle$ iff there exists $k \geq i$ s.t. $\mathit{ObsPoint}(\sigma_1, k)$ and for all σ_2, l if $\mathit{ObsEq}((\sigma_1, k), (\sigma_2, l))$, then there exists $j \leq l$ and $\sigma_2, j \models \beta$.

A specification that is trace diagnosable in a plant along all points of all traces is diagnosable in the classical sense, and we say it is *system* diagnosable. The concept of trace diagnosability does not impose any specific behavior to the diagnoser. However, it is an important concept that allows us to better characterize and understand the specification and the system.

3.5. **Maximality.** As shown in Figure 8, bounded- and finite-delay alarms are correct if they are raised within the valid bound. However, there are several possible variations of the same alarm in which the alarm is active in different instants or for different periods. We address this problem by introducing the concept of *maximality*. Intuitively, a maximal diagnoser is required to raise the alarms as soon as possible and as long as possible (without violating the correctness condition).

Definition 3.12. D is a maximal diagnoser for an alarm condition with alarm A in P iff for every trace σ_P of P, $D(\sigma_P)$ contains the maximum number of observable points i such that $D(\sigma_P), i \models A$; that is, if $D(\sigma_P), i \not\models A$, then there does not exist another correct diagnoser D' of P such that $D'(\sigma_P), i \models A$.

4. Formal Specification

In this section, we present the Alarm Specification Language with Epistemic operators (ASL_K). This language allows designers to define requirements on the FDI alarms including aspects such as delays, diagnosability and maximality.

Diagnosis conditions and alarm conditions are formalized using LTL with past operators. The definitions of trace diagnosability and maximality, however, cannot be captured by using a formalization based on LTL. To capture these two concepts, we rely on temporal epistemic logic. The intuition is that this logic enables us to reason on set of observationally equivalent traces instead that on single traces (like in LTL). We show how this logic can be used to specify diagnosability, define requirements for non-diagnosable cases and express the concept of maximality.

4.1. Diagnosis and Alarm Conditions as LTL Properties.

Let $\mathcal{P}$ be a set of propositions representing either faults, events or elementary conditions for the diagnosis. The set $\mathcal{D}_\mathcal{P}$ of *diagnosis conditions* over $\mathcal{P}$ is any formula β built with the following rule:

$$\beta ::= p \mid \beta \wedge \beta \mid \neg\beta \mid O\beta \mid Y\beta$$

with $p \in \mathcal{P}$.

We provide the LTL characterization of the *Alarm Specification Language* (ASL) in Figure 10. On the left column we provide the name of the alarm condition (as defined in the previous section), and on the right column we provide the associated LTL formalization encoding the concepts of correctness and completeness. *Correctness*, the first conjunct, intuitively says that whenever the diagnoser raises an alarm, then the fault must have occurred. *Completeness*, the second conjunct, intuitively encodes that whenever the fault occurs, the alarm will be raised. In the following, for simplicity, we abuse notation and indicate with φ both the alarm condition and the associated LTL; for an alarm condition φ, we denote by A_φ the associated alarm variable A, and with $\tau(\varphi)$ the following formulas:

$\tau(\varphi) = Y^d\beta$ for $\varphi = \textsc{ExactDel}(A, \beta, d)$;
$\tau(\varphi) = O^{\leq d}\beta$ for $\varphi = \textsc{BoundDel}(A, \beta, d)$;
$\tau(\varphi) = O\beta$ for $\varphi = \textsc{FiniteDel}(A, \beta)$.

When clear from the context, we use just A and τ instead of A_φ and $\tau(\varphi)$, respectively.

Alarm Condition	**LTL Formulation**
$\textsc{ExactDel}(A, \beta, d)$	$G(\llcorner A \lrcorner \rightarrow Y^d\beta) \wedge G(\beta \rightarrow X^d \llcorner A \lrcorner)$
$\textsc{BoundDel}(A, \beta, d)$	$G(\llcorner A \lrcorner \rightarrow O^{\leq d}\beta) \wedge G(\beta \rightarrow F^{\leq d} \llcorner A \lrcorner)$
$\textsc{FiniteDel}(A, \beta)$	$G(\llcorner A \lrcorner \rightarrow O\beta) \wedge G(\beta \rightarrow F \llcorner A \lrcorner)$

Figure 10. Alarm conditions as LTL (ASL): *Correctness* and *Completeness*

Alarm Condition	Diagnosability	Maximality
$\textsc{ExactDel}(A, \beta, d)$	$G(\beta \to X^d \llcorner K Y^d \beta \lrcorner)$	$G(\llcorner K Y^d \beta \lrcorner \to \llcorner A \lrcorner)$
$\textsc{BoundDel}(A, \beta, d)$	$G(\beta \to F^{\leq d} \llcorner K O^{\leq d} \beta \lrcorner)$	$G(\llcorner K O^{\leq d} \beta \lrcorner \to \llcorner A \lrcorner)$
$\textsc{FiniteDel}(A, \beta)$	$G(\beta \to F \llcorner K O \beta \lrcorner)$	$G(\llcorner K O \beta \lrcorner \to \llcorner A \lrcorner)$

FIGURE 11. *Diagnosability* and *Maximality*.

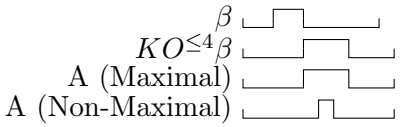


FIGURE 12. Example of Maximal and Non-Maximal traces

4.2. **Diagnosability as Epistemic Property.** We can write the diagnosability test for the different alarm conditions directly as epistemic properties. The general formulation is presented on the left column of Figure 11. In order to test for system diagnosability, we will check whether the formula holds for all traces of the system; while to check for trace diagnosability we will check whether the formula holds for single points in a trace. For example, the diagnosability test for $\textsc{ExactDel}(A, \beta, d)$ says that it is always the case that whenever β occurs, exactly d steps afterwards, the diagnoser *knows* β occurred d steps earlier. Since K is defined on observationally equivalent traces, the only way to falsify the formula would be to have a trace in which β occurs, and another one (observationally equivalent at least for the next d steps) in which β did not occur; but this is in contradiction with the definition of diagnosability (Definition 3.3).

4.3. **Maximality as Epistemic Property.** The property of maximality says that the diagnoser will raise the alarm as soon as it is possible to know the diagnosis condition, and the alarm will stay up as long as possible. The property $\llcorner K\tau \lrcorner \to \llcorner A \lrcorner$ encodes this behavior:

Theorem 4.1. *D is maximal for φ in P iff $D \otimes P \models G(\llcorner K\tau \lrcorner \to \llcorner A \lrcorner)$.*

Proof. $\Rightarrow$) Suppose D is maximal and by contradiction $D \otimes P \not\models G(\llcorner K\tau \lrcorner \to \llcorner A \lrcorner)$. Thus, there exists a trace σ_P of P and $i \geq 0$ such that $D(\sigma_P) \times \sigma_P, i \models (\llcorner K\tau \lrcorner \wedge \neg \llcorner A \lrcorner)$ (where $D(\sigma_P)$ is the diagnoser trace matching σ_P as defined in Definition 3.2). By Definition 2.6 of $\llcorner \cdot \lrcorner$, i is an observation point. Let i be the j-th observation point of σ_P. Consider D' obtained by $D(\sigma_p)$ converting the trace into a transition system using a sink state so that D' is deterministic and setting $\llcorner A \lrcorner$ to true only in the state $D(\sigma_P)[j]$ (thus triggering A in j and setting it to false at the next observation point). For every trace σ'_P of P matching with $D'(\sigma_P)$, $obs(\sigma'_P) = obs(\sigma_P)$, and thus $\sigma'_P, i \models \tau$ (since $D(\sigma_P) \times \sigma_P, i \models (\llcorner K\tau \lrcorner)$. Therefore $D' \models G(\llcorner A \lrcorner \to \tau)$ contradicting the hypothesis.

$\Leftarrow$) Suppose $D \otimes P \models G(\llcorner K\tau \lrcorner \to \llcorner A \lrcorner)$ and by contradiction D is not maximal for φ in P. Then there exists a trace σ_P of P such that $D(\sigma_P), i \not\models \llcorner A \lrcorner$ and there exists another diagnoser D' of P such that $D'(\sigma_P), i \models \llcorner A \lrcorner$ and $D' \otimes P \models G(\llcorner A \lrcorner \to \tau)$. Then, for some j, $D(\sigma_P) \otimes \sigma_P, j \not\models \llcorner A \lrcorner$, $D'(\sigma_P) \otimes \sigma_P, j \models \llcorner A \lrcorner$, and so $D(\sigma_P) \otimes \sigma_P, j \not\models \llcorner K\tau \lrcorner$ and $\sigma_P, j \models \tau$. Then there exists another trace σ'_P of P and j' such that $ObsEq((\sigma'_P, j'), (\sigma_P, j))$

and $\sigma'_P, j' \not\models \tau$. Since D' is deterministic, $D'(\sigma'_P)$ and $D'(\sigma_P)$ are equal up to position i, and so $D' \otimes P \not\models G(\llcorner A \lrcorner \rightarrow \tau)$ contradicting the hypothesis. □

Whenever the diagnoser knows that τ is satisfied, it will raise the alarm. An example of maximal and non-maximal alarm is given in Figure 12. Note that according to our definition, the set of maximal alarms is a subset of the non-maximal ones.

A property related to Maximality is the capability of the diagnoser to justify the raising of the alarm. This property is guaranteed by construction by any correct diagnoser, as shown in the following theorem.

Theorem 4.2. *Given a diagnoser D and a plant P, for each alarm A of D, with temporal condition τ, if D is correct for A it holds that:*

$$D \otimes P \models G(\llcorner A \lrcorner \rightarrow \llcorner K\tau \lrcorner)$$

Thus, whenever the diagnoser raises an alarm, it knows that the diagnosis condition has occurred.

Proof. We assume by contradiction that the $G(\llcorner A \lrcorner \rightarrow \llcorner K\tau \lrcorner)$ is not satisfied. Therefore, there exist σ and i such that $D(\sigma) \otimes \sigma, i \models \llcorner A \lrcorner \wedge \neg \llcorner K\tau \lrcorner$ (where $D(\sigma_P)$ is the diagnoser trace matching σ_P as defined in Definition 3.2), which is equivalent to $\llcorner A \lrcorner \wedge \neg K\tau$ (by Definition 2.6 of $\llcorner \cdot \lrcorner$). Thus, $\sigma, i \models \tau$ by correctness of D. In order for the $\neg K\tau$ to hold, we need another trace σ' and j s.t. $ObsEq((\sigma, i), (\sigma', j))$ and $\sigma', j \models \neg\tau$. By definition, the diagnoser is deterministic, thus we know that for σ, σ' at points i, j we will have the same value of A. Therefore, $D(\sigma') \otimes \sigma', j \models \llcorner A \lrcorner \wedge \neg\tau$ so that D is not correct, thus reaching a contradiction. □

4.4. **ASL_K Specifications.** The formalization of ASL_K (Figure 13) is obtained by extending ASL (Figure 10) with the concepts of maximality and diagnosability, defined as epistemic properties. When *maximality* is required we add a third conjunct following Theorem 4.1. When $Diag = Trace$ instead, we precondition the completeness to the trace diagnosability (as defined in Figure 11); this means that the diagnoser will raise an alarm whenever the diagnosis condition is satisfied and the diagnoser is able to know it.

Several simplifications are possible. For example, in the case $Diag = Trace$, we do not need to verify the completeness due to the following result:

Theorem 4.3. *Given a diagnoser D for a plant P and a trace diagnosable alarm condition φ, if D is maximal for φ, then D is complete.*

Proof. (ExactDel) For all σ, i if $\sigma, i \models (\beta \rightarrow X^d \llcorner K Y^d \beta \lrcorner)$, then by using the maximality assumption, we know that $\sigma, i \models (\beta \rightarrow X^d \llcorner A \lrcorner)$; thus, $\sigma, i \models (\beta \rightarrow X^d \llcorner K Y^d \beta \lrcorner) \rightarrow (\beta \rightarrow X^d \llcorner A \lrcorner)$. Similarly we can prove BoundDel and FiniteDel. □

As a corollary of Theorem 4.3, the same can be applied also for system diagnosable alarm conditions if P is diagnosable, since system diagnosability implies trace diagnosability:

Theorem 4.4. *Given an alarm condition for the system diagnosable case, and a diagnoser D for a plant P, if D is maximal for φ and φ is diagnosable in P then D is complete.*

Proof. The theorem follows directly from Theorem 4.3 and the fact that if D is complete for a trace diagnosable alarm condition that is system diagnosable, then D is also complete for the corresponding system diagnosable alarm condition. □

	Template	$Maximality = False$	$Maximality = True$
$Diag = System$	ExactDel	$G(\llcorner A \lrcorner \rightarrow Y^d \beta) \wedge G(\beta \rightarrow X^d \llcorner A \lrcorner)$	$G(\llcorner A \lrcorner \rightarrow Y^d \beta) \wedge G(\beta \rightarrow X^d \llcorner A \lrcorner) \wedge$ $G(\llcorner KY^d \beta \lrcorner \rightarrow \llcorner A \lrcorner)$
	BoundDel	$G(\llcorner A \lrcorner \rightarrow O^{\leq d} \beta) \wedge G(\beta \rightarrow F^{\leq d} \llcorner A \lrcorner)$	$G(\llcorner A \lrcorner \rightarrow O^{\leq d} \beta) \wedge G(\beta \rightarrow F^{\leq d} \llcorner A \lrcorner) \wedge$ $G(\llcorner KO^{\leq d} \beta \lrcorner \rightarrow \llcorner A \lrcorner)$
	FiniteDel	$G(\llcorner A \lrcorner \rightarrow O \beta) \wedge G(\beta \rightarrow F \llcorner A \lrcorner)$	$G(\llcorner A \lrcorner \rightarrow O \beta) \wedge G(\beta \rightarrow F \llcorner A \lrcorner) \wedge$ $G(\llcorner KO \beta \lrcorner \rightarrow \llcorner A \lrcorner)$
$Diag = Trace$	ExactDel	$G(\llcorner A \lrcorner \rightarrow Y^d \beta) \wedge$ $G((\beta \rightarrow X^d \llcorner KY^d \beta \lrcorner) \rightarrow (\beta \rightarrow X^d \llcorner A \lrcorner))$	$G(\llcorner A \lrcorner \rightarrow Y^d \beta) \wedge$ $G((\beta \rightarrow X^d \llcorner KY^d \beta \lrcorner) \rightarrow (\beta \rightarrow X^d \llcorner A \lrcorner)) \wedge$ $G(\llcorner KY^d \beta \lrcorner \rightarrow \llcorner A \lrcorner)$
	BoundDel	$G(\llcorner A \lrcorner \rightarrow O^{\leq d} \beta) \wedge$ $G((\beta \rightarrow F^{\leq d} \llcorner KO^{\leq d} \beta \lrcorner) \rightarrow (\beta \rightarrow F^{\leq d} \llcorner A \lrcorner))$	$G(\llcorner A \lrcorner \rightarrow O^{\leq d} \beta) \wedge$ $G((\beta \rightarrow F^{\leq d} \llcorner KO^{\leq d} \beta \lrcorner) \rightarrow (\beta \rightarrow F^{\leq d} \llcorner A \lrcorner)) \wedge$ $G(\llcorner KO^{\leq d} \beta \lrcorner \rightarrow \llcorner A \lrcorner)$
	FiniteDel	$G(\llcorner A \lrcorner \rightarrow O \beta) \wedge$ $G((\beta \rightarrow F \llcorner KO \beta \lrcorner) \rightarrow (\beta \rightarrow F \llcorner A \lrcorner))$	$G(\llcorner A \lrcorner \rightarrow O \beta) \wedge$ $G((\beta \rightarrow F \llcorner KO \beta \lrcorner) \rightarrow (\beta \rightarrow F \llcorner A \lrcorner)) \wedge$ $G(\llcorner KO \beta \lrcorner \rightarrow \llcorner A \lrcorner)$

FIGURE 13. ASL_K specification patterns among the four dimensions: $Diagnosability$, $Maximality$, $Completeness$ and $Correctness$.

This Theorem is interesting because it tells us that if a specification that was required to be system diagnosable is indeed system diagnosable, then we can just check whether the diagnoser is maximal and avoid performing the completeness test.

	Template	$Maximality = False$	$Maximality = True$
$Diag = System$	ExactDel	$G(\llcorner A \lrcorner \rightarrow Y^d \beta) \wedge G(\beta \rightarrow X^d \llcorner A \lrcorner)$	$G(\llcorner A \lrcorner \rightarrow Y^d \beta) \wedge G(\beta \rightarrow X^d \llcorner A \lrcorner)$ $G(\llcorner KY^d \beta \lrcorner \rightarrow A)$
	BoundDel	$G(\llcorner A \lrcorner \rightarrow O^{\leq d} \beta) \wedge G(\beta \rightarrow F^{\leq d} \llcorner A \lrcorner)$	$G(\llcorner A \lrcorner \rightarrow O^{\leq d} \beta) \wedge G(\beta \rightarrow F^{\leq d} \llcorner A \lrcorner) \wedge$ $G(\llcorner KO^{\leq d} \beta \lrcorner \rightarrow A)$
	FiniteDel	$G(\llcorner A \lrcorner \rightarrow O \beta) \wedge G(\beta \rightarrow F \llcorner A \lrcorner)$	$G(\llcorner A \lrcorner \rightarrow O \beta) \wedge G(\beta \rightarrow F \llcorner A \lrcorner) \wedge$ $G(\llcorner KO \beta \lrcorner \rightarrow A)$
$Diag = Trace$	ExactDel	$G(\llcorner A \lrcorner \rightarrow Y^d \beta) \wedge$ $G(\llcorner KY^d \beta \lrcorner \rightarrow A)$	$G(\llcorner A \lrcorner \rightarrow Y^d \beta) \wedge$ $G(\llcorner KY^d \beta \lrcorner \rightarrow A)$
	BoundDel	$G(\llcorner A \lrcorner \rightarrow O^{\leq d} \beta) \wedge$ $G((\beta \wedge F^{\leq d} \llcorner KO^{\leq d} \beta \lrcorner) \rightarrow F^{\leq d} \llcorner A \lrcorner)$	$G(\llcorner A \lrcorner \rightarrow O^{\leq d} \beta) \wedge$ $G(\llcorner KO^{\leq d} \beta \lrcorner \rightarrow A)$
	FiniteDel	$G(\llcorner A \lrcorner \rightarrow O \beta) \wedge$ $G((\beta \wedge F \llcorner KO \beta \lrcorner) \rightarrow F \llcorner A \lrcorner)$	$G(\llcorner A \lrcorner \rightarrow O \beta) \wedge$ $G(\llcorner KO \beta \lrcorner \rightarrow A)$

FIGURE 14. ASL_K with simplified patterns for $Diag = Trace$

Theorem 4.5. *For all trace diagnosable and non-maximal* ExactDel *specifications, completeness can be replaced by maximality. Formally, for all* σ, $\sigma \models G((\beta \rightarrow X^d \llcorner KY^d \beta \lrcorner) \rightarrow (\beta \rightarrow X^d \llcorner A \lrcorner))$ *iff* $\sigma \models G(\llcorner KY^d \beta \lrcorner \rightarrow \llcorner A \lrcorner)$

Proof.

$$\begin{aligned}
&\sigma, i \models ((\beta \rightarrow X^d \llcorner KY^d \beta \lrcorner) \rightarrow (\beta \rightarrow X^d \llcorner A \lrcorner)) && \text{iff}\\
&\sigma, i \models ((\beta \wedge X^d \llcorner KY^d \beta \lrcorner) \rightarrow X^d \llcorner A \lrcorner) && \text{iff}\\
\sigma, i+d &\models ((Y^d \beta \wedge \llcorner KY^d \beta \lrcorner) \rightarrow \llcorner A \lrcorner) && \text{iff}\\
\sigma, i+d &\models ((\llcorner Y^d \beta \wedge KY^d \beta \lrcorner) \rightarrow \llcorner A \lrcorner) && \text{iff}\\
\sigma, i+d &\models (\llcorner KY^d \beta \lrcorner \rightarrow \llcorner A \lrcorner)
\end{aligned}$$

Therefore, we can conclude that for all i, $\sigma, i \models ((\beta \rightarrow X^d \llcorner KY^d \beta \lrcorner) \rightarrow (\beta \rightarrow X^d \llcorner A \lrcorner))$ iff for all $j \geq d$, $\sigma, j \models (\llcorner KY^d \beta \lrcorner \rightarrow \llcorner A \lrcorner)$. We conclude noting that for $j < d$, $Y^d \beta$ is false and therefore $\sigma, j \models (\llcorner KY^d \beta \lrcorner \rightarrow \llcorner A \lrcorner)$. □

After applying the simplifications specified in Theorem 4.3 and Theorem 4.5 and the equivalence $\llcorner \phi \lrcorner \rightarrow \llcorner \psi \lrcorner \equiv \llcorner \phi \lrcorner \rightarrow \psi$, we obtain the table in Figure 14, where the patterns in the lower half ($Diag = Trace$) have been simplified.

An ASL_K specification is built by instantiating the patterns defined in Figure 13. For example, we would write $\text{ExactDel}_K(A, \beta, d, Trace, True)$ for an exact-delay alarm A for β with delay d, that satisfies the trace diagnosability property and is maximal. An introductory example on the usage of ASL_K for the specification of a diagnoser is provided in [BCGT13]. Figure 15 shows how we extend the specification for the BSS by introducing requirements on the diagnosability and maximality of alarms. In particular, all the alarms that we defined are not system diagnosable. Therefore, we need to weaken the requirements and make them trace-diagnosable. The patterns are then converted into temporal epistemic formulae as shown in Figure 16.

$\text{ExactDel}_K(PSU1_{Exact_i}, \beta_{PSU1}, i, Trace, True)$
$\text{BoundDel}_K(PSU1_{Bound}, \beta_{PSU1}, C, Trace, True)$
$\text{BoundDel}_K(BS, \beta_{BS}, DC, Trace, True)$
$\text{FiniteDel}_K(Discharged, \beta_{Depleted}, Trace, False)$
$\text{FiniteDel}_K(B1Leak, \beta_{Battery1}, System, True)$

Figure 15. ASL_K Specification for the BSS

Alarm	**Formula**
$PSU1_{Exact_i}$	$G(\llcorner PSU1_{Exact_i} \lrcorner \rightarrow Y^i \beta_{PSU1}) \wedge G(\llcorner KY^i \beta_{PSU1} \lrcorner \rightarrow \llcorner PSU1_{Exact_i} \lrcorner)$
$PSU1_{Bound}$	$G(\llcorner PSU1_{Bound} \lrcorner \rightarrow O^{\leq C} \beta_{PSU1}) \wedge G(\llcorner KO^{\leq C} \beta_{PSU1} \lrcorner \rightarrow \llcorner PSU1_{Bound} \lrcorner)$
BS	$G(\llcorner BS \lrcorner \rightarrow O^{\leq DC} \beta_{BS}) \wedge G(\llcorner KO^{\leq DC} \beta_{BS} \lrcorner \rightarrow \llcorner BS \lrcorner)$
$Discharged$	$G(\llcorner Discharged \lrcorner \rightarrow O \beta_{Deplated}) \wedge G((\beta_{Deplated} \wedge F \llcorner KO \beta_{Deplated} \lrcorner) \rightarrow F \llcorner Discharged \lrcorner)$
$B1Leak$	$G(\llcorner B1Leak \lrcorner \rightarrow O \beta_{Battery1}) \wedge G(\beta_{Battery1} \rightarrow F \llcorner B1Leak \lrcorner) \wedge G(\llcorner KO \beta_{Battery1} \lrcorner \rightarrow \llcorner B1Leak \lrcorner)$

Figure 16. KL_1 translation of ASL_K patterns for the BSS

In the BSS, if we assume at most one fault, then the sensor faults are neither system nor trace diagnosable since we are only able to observe the difference in output of the sensors, and therefore we can never be sure of which sensor is experiencing the fault. Restricting the model to two faults, instead, makes it possible to detect when both sensors are faulty, since the device stops working. The Battery Leak is trace diagnosable but not system diagnosable. This means that in general, we cannot detect the battery leak, but there is at least one execution in which we can. In particular, this is the execution in which the mode becomes Secondary 2 when Battery 1 was charged, and we can see the battery discharging, thus detecting the fault. Note that to detect this fault, we need to recall the fact that previously the battery was charged, and therefore a simple diagnoser without memory would not be able to detect this fault.

5. Validation and Verification of ASL_K Specifications

Thanks to the formal characterization of ASL_K, it is possible to apply formal methods for the validation and verification of a set of FDI requirements. In *validation* we verify that the requirements capture the interesting behaviors and exclude the spurious ones, before proceeding with the design of the diagnoser. In *verification*, we check that a candidate diagnoser fulfills a set of requirements.

5.1. **Validation.** Given a specification $\mathcal{A}$ for our diagnoser, we want to make sure that it captures the designer expectations. Known techniques for requirements validation (e.g.,[CRST12]) include checking their *consistency*, and their *realizability*, i.e., whether they can be implemented on a given plant. Moreover, often we want to show that there exists some condition under which the alarm might be triggered (*possibility*), and some other conditions that require the alarm to be triggered (*necessity*).

By construction, an ASL_K specification is always *consistent*, i.e., there are no internal contradictions. This is due to the fact that alarm specifications do not interact with each other, and each alarm specification can always be satisfied by a diagnosable plant. Moreover, in Section 6, we will prove that we can always synthesize a diagnoser satisfying $\mathcal{A}$, with the only assumption that if $\mathcal{A}$ contains some system diagnosable alarm condition, then that condition is diagnosable in the plant. Thus, the check for *realizability* reduces to checking that the plant is diagnosable for the system diagnosable conditions in $\mathcal{A}$. The diagnosability check can be performed via epistemic model-checking (Section 4.2) or it can be reduced to an LTL model-checking problem using the twin-plant construction [CPC03].

An alarm that is always (or never) triggered is not useful. Therefore, we need to check under which conditions the alarm can and cannot be triggered. Moreover, there might be some assumptions on the environment of the diagnoser (including details on the plant) that might have an impact on the the alarms. For example, if we have a single fault assumption for our system, an alarm that implicitly depends on the occurrence of two faults will never be triggered. Similarly, our assumptions on the environment might provide some link between the behavior of different components, or dynamics of faults and thus characterize the relation between different alarms.

We consider a set of environmental assumptions E expressed as LTL properties. This set can be empty, or include detailed information on the behavior of the environment and plant, since throughout the different phases of the development process, we have access to better versions of the plant model, and therefore the analysis can be refined.

When checking *possibility* we want that the alarms can be eventually activated, but also that they are not always active. This means that for a given alarm condition $\varphi \in \mathcal{A}$, we are interested in verifying that there is a trace $\sigma \in E$ and a trace $\sigma' \in E$ s.t. $\sigma \models F\llcorner A_\varphi \lrcorner$ and $\sigma' \models F\neg\llcorner A_\varphi \lrcorner$. This can be done by checking the unsatisfiability of $(E \wedge \varphi) \rightarrow G\neg\llcorner A_\varphi \lrcorner$ and $(E \wedge \varphi) \rightarrow G\llcorner A_\varphi \lrcorner$.

Checking *necessity* provides us a way to understand whether there is some correlation between alarms. This, in turns, makes it possible to simplify the model, or to guarantee some redundancy requirement. To check whether $A_{\varphi'}$ is a more general alarm than A_φ (subsumption) we check whether $(E \wedge \varphi \wedge \varphi') \rightarrow G(\llcorner A_\varphi \lrcorner \rightarrow \llcorner A_{\varphi'} \lrcorner)$ is valid. An example of subsumption of alarms is given by the definition of maximality: any non-maximal alarm subsumes its corresponding maximal version. Finally, we can verify that two alarms are mutually exclusive by checking the validity of $(E \wedge \varphi \wedge \varphi') \rightarrow G\neg(\llcorner A_\varphi \lrcorner \wedge \llcorner A_{\varphi'} \lrcorner)$.

To clarify the concepts presented in this section, we apply a necessity check on our running example. In the Battery-Sensor, we have two alarms specified on $PSU1$ (Figure 15): $PSU1_{Exact_i}$ and $PSU1_{Bound}$. Let's take $i = C = 2$, thus obtaining:

- $\textsc{ExactDel}_K(PSU1_{Exact_2}, \beta_{PSU1}, 2, Trace, True)$
- $\textsc{BoundDel}_K(PSU1_{Bound}, \beta_{PSU1}, 2, Trace, True)$

we want to show that $PSU1_{Exact_i}$ is more specific than (is subsumed by) $PSU1_{Bound}$. This means that for any plant and diagnoser, the following holds:

$$D \otimes P \models (\varphi_{PSU1_{Exact_2}} \wedge \varphi'_{PSU1_{Bound}}) \rightarrow G(\llcorner PSU1_{Exact_2} \lrcorner \rightarrow \llcorner PSU1_{Bound} \lrcorner)$$

By renaming with $PE = PSU1_{Exact_2}$ and $PB = PSU_{Bound}$ (for brevity) and expanding the definitions of $\varphi_{PSU1_{Exact_2}} \wedge \varphi'_{PSU1_{Bound}}$ we have that

$$\begin{aligned} D \otimes P \models (G(\llcorner PE \lrcorner \rightarrow Y^2\beta) \wedge G(\llcorner KY^2\beta \lrcorner \rightarrow \llcorner PE \lrcorner) \wedge \\ G(\llcorner PB \lrcorner \rightarrow O^{\leq 2}\beta) \wedge G(\llcorner KO^{\leq 2}\beta \lrcorner \rightarrow \llcorner PB \lrcorner)) \\ \rightarrow G(\llcorner PE \lrcorner \rightarrow \llcorner PB \lrcorner) \end{aligned}$$

We can apply Theorem 4.2, and therefore write:

$$\begin{aligned} D \otimes P \models (G(\llcorner PE \lrcorner \rightarrow Y^2\beta) \wedge G(\llcorner KY^2\beta \lrcorner \rightarrow \llcorner PE \lrcorner) \wedge \\ G(\llcorner PB \lrcorner \rightarrow O^{\leq 2}\beta) \wedge G(\llcorner KO^{\leq 2}\beta \lrcorner \rightarrow \llcorner PB \lrcorner) \wedge \\ G(\llcorner PE \lrcorner \rightarrow \llcorner KY^2\beta \lrcorner) \wedge G(\llcorner PB \lrcorner \rightarrow \llcorner KO^{\leq 2}\beta \lrcorner)) \\ \rightarrow G(\llcorner PE \lrcorner \rightarrow \llcorner PB \lrcorner) \end{aligned}$$

To prove that the above formula is valid (and therefore it is satisfied by any plant and diagnoser), we prove that its negation is *unsatisfiable*:

$$\begin{aligned} (G(\llcorner PE \lrcorner \rightarrow Y^2\beta) \wedge G(\llcorner KY^2\beta \lrcorner \rightarrow \llcorner PE \lrcorner) \wedge \\ G(\llcorner PB \lrcorner \rightarrow O^{\leq 2}\beta) \wedge G(\llcorner KO^{\leq 2}\beta \lrcorner \rightarrow \llcorner PB \lrcorner) \wedge \\ G(\llcorner PE \lrcorner \rightarrow \llcorner KY^2\beta \lrcorner) \wedge G(\llcorner PB \lrcorner \rightarrow \llcorner KO^{\leq 2}\beta \lrcorner)) \\ \wedge \neg G(\llcorner PE \lrcorner \rightarrow \llcorner PB \lrcorner) \end{aligned}$$

The first part of this formula is composed by conjuncts in the form $G\psi$. This means that a counter examples is a trace for which each state satisfies ψ. Moreover, we need one of these states to satisfy $(PE \wedge \neg\llcorner PB \lrcorner)$. Therefore, to prove the unsatisfiable of the above formula,

we can just prove that no state exists that satisfies:

$$\begin{aligned}(\llcorner PE\lrcorner \to Y^2\beta) \wedge (\llcorner KY^2\beta\lrcorner \to \llcorner PE\lrcorner) \wedge \\ (\llcorner PB\lrcorner \to O^{\leq 2}\beta) \wedge (\llcorner KO^{\leq 2}\beta\lrcorner \to \llcorner PB\lrcorner) \wedge \\ (\llcorner PE\lrcorner \to \llcorner KY^2\beta\lrcorner) \wedge (\llcorner PB\lrcorner \to \llcorner KO^{\leq 2}\beta\lrcorner)) \\ \wedge \llcorner PE\lrcorner \wedge \neg\llcorner PB\lrcorner\end{aligned}$$

We show this by a contradiction since:

$$\begin{aligned}& \cdots \wedge \llcorner PE\lrcorner \wedge \neg\llcorner PB\lrcorner \\ \textbf{\textit{ObsPoint}}\ \textbf{Def.}\quad & \cdots \wedge \llcorner\top\lrcorner \wedge PE \wedge \neg PB \\ \textbf{Theorem 4.2 on PE}\quad & \cdots \wedge \llcorner\top\lrcorner \wedge PE \wedge \neg PB \wedge KYY\beta \\ \textbf{Maximality of PB}\quad & \cdots \wedge \llcorner\top\lrcorner \wedge PE \wedge \neg PB \wedge KYY\beta \wedge \neg KO^{\leq 2}\beta \\ \dagger\textbf{Def. of } \neg K\quad & \cdots \wedge \llcorner\top\lrcorner \wedge PE \wedge \neg PB \wedge KYY\beta \wedge \neg O^{\leq 2}\beta \\ \textbf{Def. of } O^{\leq n}\quad & \cdots \wedge \llcorner\top\lrcorner \wedge PE \wedge \neg PB \wedge KYY\beta \wedge \neg(\beta \vee Y\beta \vee YY\beta) \\ \textbf{K Axiom } (K\phi \to \phi)\quad & \cdots \wedge \llcorner\top\lrcorner \wedge PE \wedge \neg PB \wedge YY\beta \wedge \neg\beta \wedge \neg Y\beta \wedge \neg YY\beta\end{aligned}$$

Thus reaching a contradiction between $YY\beta$ and $\neg YY\beta$. In the step marked with † we need to show that two observationally equivalent traces exists s.t. one satisfies $O^{\leq 2}\beta$ and the other $\neg O^{\leq 2}\beta$; therefore, we only need to show that one of the two (namely $\neg O^{\leq 2}\beta$) does not exist.

5.2. **Verification.** The verification of a system w.r.t. a specification can be performed via model-checking techniques using the semantics of the alarm conditions:

Definition 5.1. Let D be a diagnoser for alarms $\mathcal{A}$ and plant P. We say that D *satisfies a set* $\mathcal{A}$ *of* ASL_K *specifications* iff for each φ in $\mathcal{A}_{\mathcal{P}}$ there exists an alarm $A_\varphi \in \mathcal{A}$ and $D \otimes P \models \varphi$.

To perform this verification steps, we need in general a model checker for KL_1 with asynchronous/synchronous perfect recall such as MCK [GM04]. However, if the specification falls in the pure LTL fragment (ASL) we can verify it with an LTL model-checker such as nuXmv [CCD+14] thus benefiting from the efficiency of the tools in this area.

Moreover, a diagnoser is required to be deterministic. This is important, on one hand, for implementability, on the other hand, to ensure that the composition of the plant with the diagnoser does not reduce the behaviors of the plant. In order to verify that a given diagnoser $D = \langle V, E, I, \mathcal{T}\rangle$ is deterministic, we check the following conditions:

- I must be satisfiable,
- $I \wedge I[V_c/V] \to V = V_c$ must be valid,
- for all $e \in E$, $\forall V \exists V'.\mathcal{T}(e)$ must be valid (note that this corresponds to the validity of the pre-image of $\top$),
- for all $e \in E$, $\mathcal{T}(e) \wedge \mathcal{T}(e)[V_c/V'] \to V' = V_c$ must be valid.

Therefore, we can solve the problem with a finite set of satisfiability checks and pre-image computations.

6. Synthesis of a Diagnoser from an ASL_K Specification

In this section, we discuss how to synthesize a diagnoser that satisfies a given specification $\mathcal{A}$. We considers the most expressive case of ASL_K (maximal/trace diagnosable), which also satisfies all the other cases.

The idea is to generate an automaton that encodes the set of possible states in which the plant could be after each observations. The result is achieved by generating the power-set of the states of the plant, also called *belief states*, and defining a suitable transition relation among the elements of this set, only taking into account observable information. Each belief state of the automaton is then annotated with the alarms that are satisfied in all the states of the belief state. The resulting automaton is the Diagnoser.

The approach resembles the constructions by Sampath [SSL+96] and Schumann [Sch04], with the following main differences. First, we consider LTL Past expression as diagnosis condition, and not only fault events as done in previous works. Second, instead of providing a set of possible diagnoses, we provide alarms. In order to raise the alarm, we need to be certain that the alarm condition is satisfied for all possible diagnoses. This gives raise to a 3-valued alarm system: we *know* that the fault occurred; *know* that the fault did *not* occur; or we are *uncertain*. Moreover, the approach works for the asynchronous case. Although the use of a power-set construction in the setting of temporal epistemic logic is not novel (e.g. [Dim09] for synchronous CTLK model-checking), the main contribution of this section is to show the formal properties of the diagnoser, and in particular that it satisfies the specification. In a way, this algorithm is a strong indicator of a deep connection between the topics of temporal epistemic logic reasoning and FDI design.

6.1. **Synthesis algorithm.** Given a partially observable plant $P = \langle V^P, E^P, I^P, \mathcal{T}^P, E_0^P \rangle$, let S be the set of states of P. The *belief automaton* is defined as $\mathcal{B}(P) = \langle B, E, b_0, R \rangle$ where $B = 2^S$, $E = E_o^P$, $b_0 \in B$ and $R : (B \times E) \to B$. B represents the set of sets of states, also called *belief states*. Given a belief state b, we use b^* to represent the set of states that are reachable from b by only using events in $E^P \setminus E_o^P$ (non observable events), and call it the u-transitive closure. Formally, b^* is the least set s.t. $b \subseteq b^*$ and if there exist $e \in E^P \setminus E_o^P$ and $s' \in b^*$ such that $\langle s', s \rangle \in \mathcal{T}^P(e)$ then $s \in b^*$. b_0 is the initial belief state and contains the states that satisfy the initial condition I^P (i.e., $b_0 = \{s \mid s \models I^P\}$).

Given a belief state b and an observable event $e \in E_o^P$, we define the successor belief state b' as:

$$R(b, e) = b' = \{s' \mid \exists s \in b^*.\ \langle s, s' \rangle \models T^P(e)\}$$

that is the set of states that are compatible with the observable event e in a state of the u-transitive closure of b. Intuitively, we first compute the u-transitive closure of b to account for all non-observable transitions, and then we consider all the different states that can be reached from b^* with an occurrence of the event e.

The diagnoser is obtained by annotating each state of the belief automaton with the corresponding alarms. We annotate with A_φ all the states b that satisfy the temporal property $\tau(\varphi)$. As explained later on, any temporal $\tau(\varphi)$ can be handled by introducing suitable propositional formulas. Therefore we consider the simplest case in which $\tau(\varphi)$ is a propositional formula and formally say that the annotation a_b of the belief state b is the assignment to A_φ such that $a_b(A_\varphi)$ is true iff for all $s \in b$, $s \models \tau(\varphi)$. We perform the same annotation for $A_{\neg\varphi}$. The diagnoser obtained by this algorithm induces three alarms, related to the knowledge of the diagnoser. In particular, the diagnoser can be sure that

```
function BELIEF_AUTOMATON(I, T, E, E_o)
    visited ← {}
    edges ← {}
    stack ← [I]
    while not stack.is_empty() do
        b ← stack.pop()
        b* ← u_trans_closure(b, T, E)
        for all o ∈ get_observable_events(b*, T, Eo) do
            target_belief ← reachable_w_obs(b*, o, T)
            edges.add((b, o, target_belief))
            if target_belief ∉ visited then
                visited.add(target_belief)
                stack.push(target_belief)
            end if
        end for
    end while
    return Automaton(visited, edges)
end function
```

FIGURE 17. Pseudo-code of the Belief Automaton construction phase

a condition occurred (A_φ) can be sure that a condition did not occur ($A_{\neg\varphi}$) or can be uncertain on whether the condition occurred ($\neg A_\varphi \wedge \neg A_{\neg\varphi}$) – notice that, by construction, it is not possible for both A_φ and $A_{\neg\varphi}$ to be true at the same time. In this way, at any point in time we are able to understand whether we are on a trace that is not diagnosable (and thus there is uncertainty) or whether the diagnoser knows that the condition did not occur. This can thus provide additional insight on the behavior of the system.

Figure 17 provides a pseudo-code of the main function of the synthesis task: the construction of the belief automaton. Starting from the set of initial states, we perform an explicit visit until we have explored all belief states. For each belief state we first compute its u-transitive closure ($u\_trans\_closure$) w.r.t. the non-observable events E, obtaining b^*. We then compute the possible observable events available from b^*, and iterate over each event o_i obtaining the set of states $target\_belief$ such that $T(b\_star, o_i, target\_belief)$ is satisfied ($reachable\_w\_obs$). We can now add a transition to our automaton linking the belief state b to the belief state $target\_belief$ through the event o_i. Once we have completed this phase, we have an automaton with labeled transitions. The automaton resulting from this function can then be annotated by visiting each state and testing whether the state entails (or not) the alarm specification.

6.2. **Running Example.** We show the first step of the algorithm on a simplified version of the battery component of our running example (Figure 5). We ignore the events related to threshold passing of the battery (*Mid*, *Low*, *High*) and only consider the observable event *Off*, signaled when the charge reaches zero, and the ones due to mode changes. To keep the representation compact, we indicate each state with three symbols. For example, we use (NPC) to indicate the state "Nominal, Primary, Charging" and ($NP\overline{C}$) to indicate the state "Nominal, Primary, Not Charging". Similarly we use F, O, and D to indicate

Faulty, Offline and Double. We recall that in the original model, the mode transitions are observable but all other transitions are not.

In the first step (Figure 18), we take the set of initial states. This is the set of states (NPC) for any value of the $charge \in [0, C]$. The u-transitive closure needs to take into account all non-observable transitions. Therefore, we need to consider going from Nominal to Faulty, from Charging to Not Charging, and their combination.

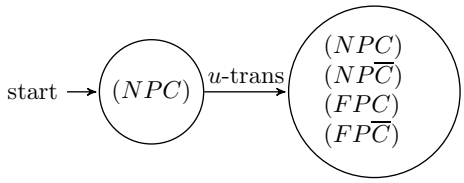


FIGURE 18. Expanding the initial belief state of the battery LTS.

These are all the states that are reachable before an observable event can occur. We now take each observable event and compute the set of states that are reachable with one of the observable events (Figure 19): the battery being discharge (*Off*), and the change of mode (*Offline*, *Double*). Note that one of the belief states is smaller than the others.

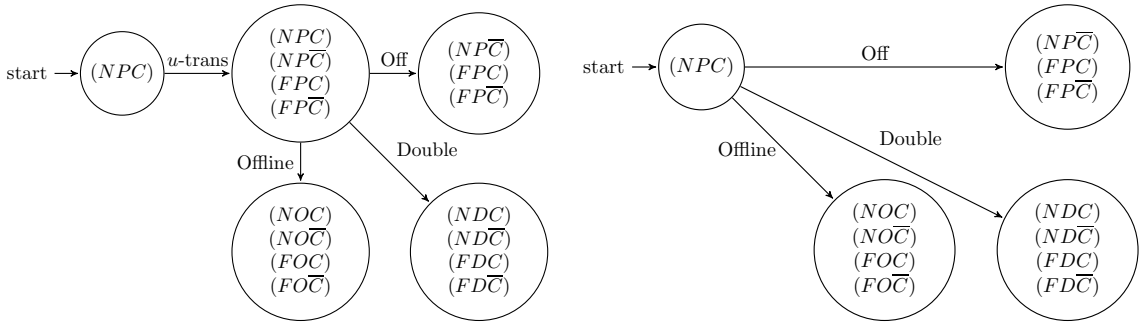


FIGURE 19. Expanding the belief state via observable transitions

This is due to the fact that in our model, the discharging of the battery cannot occur if the battery is nominal, charging and in primary mode (NPC). Thus, the fact that we receive the Off event allows us to exclude that state. The state obtained by computing the transitive closure is not part of our final automaton, and is provided in the figure only to simplify the understanding.

We repeat these two steps until all belief states have been explored. We then proceed to the labeling phase, in which we label each state with the corresponding alarm. For example, by considering the alarms $\textsc{ExactDel}(A_{NC}, Nominal \wedge Charging, 0)$ and $\textsc{ExactDel}(A_N, Nominal, 0)$, we obtain the diagnoser partially represented in Figure 20. Notice how, in the initial state we can raise the alarm A_{NC}, and this alarm can only be changed by an observable transition.

6.3. **Formal Properties of the Synthesized diagnoser.** We now show that the generated transition system is a diagnoser and that it is correct, complete and maximal. Lets assume that φ is an exact delay specification, with delay zero. Any other alarm conditions

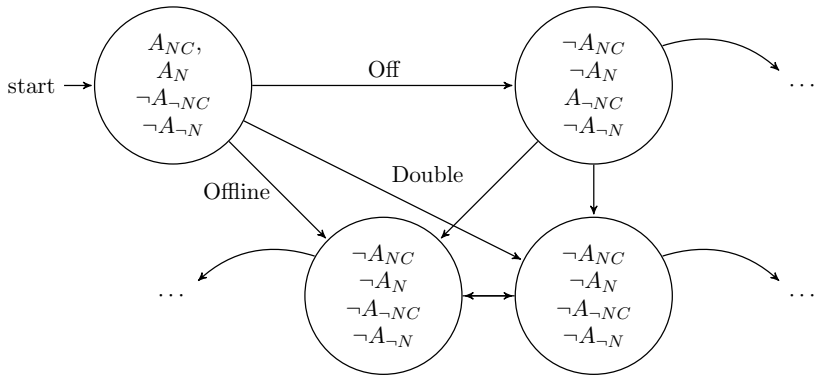


FIGURE 20. Annotation of the belief states

can be reduced to this case. We build a new plant P' by adding a monitor variable $\overline{\tau}$ to P s.t., $P' = P \times (G(\tau(\varphi) \leftrightarrow \overline{\tau}))$, where we abuse notation to indicate the synchronous composition of the plant with an automaton that encodes the monitor variable. By rewriting the alarm condition as $\varphi' = \textsc{ExactDel}(A_\varphi, \overline{\tau}, 0)$, we obtain that $D \otimes P \models \varphi$ iff $D \otimes P' \models \varphi'$. Thus, it is sufficient to show the following results only for the zero delay case. We define D_φ as the diagnoser for φ. $D_\varphi = \langle V^{D_\varphi}, E^{D_\varphi}, I^{D_\varphi}, \mathcal{T}^{D_\varphi} \rangle$ is a symbolic representation of $\mathcal{B}(P)$ with $A_\varphi \subseteq V^{D_\varphi}$, $E_o^{D_\varphi} = E_o^P$ and such that every state b of D_φ represents a state in B (with abuse of notation we do not distinguish between the two since the assignment to A_φ is determined by b).

Theorem 6.1. *D_φ is deterministic.*

Proof. The result follows directly from the definition of the belief automaton, which is deterministic (one initial state and one successor). Note that the assignment to A_φ is not relevant since it is determined by the belief state. □

Lemma 6.2. *For every reachable state $b \times s$ of $D_\varphi \otimes P$, for every trace σ reaching $b \times s$, for every state $s' \in b$, there exists a trace σ' reaching $b \times s'$ with $obs(\sigma) = obs(\sigma')$.*

Proof. By induction on σ. All traces are observationally equivalent in the initial state. Let $\langle b_1 \times s_1, e, b \times s \rangle$ be the last transition of σ and let σ_1 be the prefix of σ without this last transition. If $e \in E \setminus E_o$ then $obs(\sigma) = obs(\sigma_1)$. Otherwise, for every state $s' \in b$ there exists a transition $\langle s'_1, e, s' \rangle$ such that $s'_1 \in b_1^*$. By inductive hypothesis there exists a trace σ'_1 reaching $b_1 \times s'_1$ such that $obs(\sigma_1) = obs(\sigma'_1)$. Therefore the concatenation of σ'_1 with the transition $\langle b_1 \times s'_1, e, b \times s' \rangle$ results in a trace σ' reaching $b \times s'$ such that $obs(\sigma) = obs(\sigma')$. □

Theorem 6.3 (Maximality)**.** $D_\varphi \otimes P \models G(\llcorner K\tau(\varphi) \lrcorner \rightarrow \llcorner A_{\varphi} \lrcorner)$.

Proof. Consider a trace σ and $i \geq 0$. If $\sigma, i \models \llcorner K\tau(\varphi) \lrcorner$, then for all traces σ' and points j s.t. $ObsEq((\sigma, i), (\sigma', j))$, $\sigma', j \models \tau(\varphi)$. By Lemma 6.2, all states $s \in \sigma[i]$ there exists a trace σ' with $obs(\sigma) = obs(\sigma')$, and therefore $s \models \tau(\varphi)$ so that $\sigma[i] \models \llcorner A_{\varphi} \lrcorner$. □

Lemma 6.4. *Given a trace σ of $D_\varphi \otimes P$. Let $\sigma[i] = b \times s$. If i is an observation point, then $s \in b$.*

Proof. By assumption, i is the n-th observation point of σ for some n. We prove the lemma by induction on n.

Consider the case $n = 1$. If $\sigma[0] = b_0 \times s_0$, by construction of D_φ, $s_0 \in b_0$. Let $\sigma[i-1] = b' \times s'$ and let e be the i-th (observable) event of σ. If i is the first observation point of σ, it means that $b' = b_0$ and $s' \in b_0^*$. Moreover, $\langle s', s\rangle \in T(e)$ and therefore $s \in b$.

Consider the case $n > 1$. Let j be the $n-1$ observation point, $\sigma[j] = b_j \times s_j$, $\sigma[i-1] = b' \times s'$ and let e be the i-th (observable) event of σ. Similarly to the previous case, $b' = b_j$ and $s' \in b_j^*$. Moreover $\langle s', s\rangle \in T(e)$ and therefore $s \in b$. □

Theorem 6.5 (Correctness)**.** $D_\varphi \otimes P \models G(\llcorner A_{\varphi\lrcorner} \rightarrow \tau(\varphi))$.

Proof. Consider a trace σ and $i \geq 0$. Suppose $\sigma, i \models \llcorner A_{\varphi\lrcorner}$ and let σ_{D_φ} and σ_P be respectively the left and right component of σ. Then, for all $s \in \sigma_{D_\varphi}[i]$, $s \models \tau(\varphi)$. Since i is an observation point, by Lemma 6.4, $\sigma_P[i] \in \sigma_{A_\varphi}[i]$. We can conclude that $\sigma[i] \models \tau(\varphi)$. □

Theorem 6.6 (Completeness)**.** *If φ is an alarm condition required to be trace diagnosable, then D_φ is complete. If φ is a system diagnosable condition and φ is diagnosable in P, then D_φ is complete.*

Proof. Since D_φ is maximal and correct (Theorems 6.3 and 6.5), we can apply Theorem 4.3 (if φ is trace diagnosable) or Theorem 4.4 (if it is system diagnosable) to obtain completeness. □

7. Industrial Experience

The methods described in this paper have been motivated by AUTOGEF, a project [Eur10, AUT, ANY+12] funded by the European Space Agency. The main goal of the project was the definition of a set of requirements for an on-board Fault Detection, Identification and Recovery (FDIR) component and its synthesis. The problem was cast in the frame of discrete event systems, communicating asynchronously, and tackled by synthesizing the Fault Detection (FDI) and Fault Recovery (FR) components separately – with the idea that the FDI provides sufficient diagnosis information for the FR to act on.

A similar problem was further investigated in FAME, another ESA-funded project [Eur11, FAM, GFB+14, BBC+14a, BBC+14b]. In the context of FAME, we addressed the problem of synthesis of FDI and FR components for continuous time systems, with synchronous communication – in particular the diagnoser communicates with the plant by sampling the values of the sensors at periodic time intervals. In both cases, AUTOGEF and FAME, we addressed the problem of FiniteDel diagnosis, which was of interest from an industrial perspective.

Within AUTOGEF, the design approach initially was evaluated using scalable benchmark examples. Then, Thales Alenia Space evaluated AUTOGEF on an industrial case study based on the EXOMARS Trace Gas Orbiter. This case-study is a significant application of the approach described in this paper, since it covers all the phases of the FDIR development process. The (nominal and faulty) behavior of the system was modeled using a formal language. A table-based and pattern-based approach was adopted to describe the mission phases/modes and the observability characteristics of the system. The specification of FDIR requirements by means of patterns greatly simplified the accessibility of the tool to engineers that were not experts in formal methods. Alarms were specified in the case of finite delay, under the assumption of trace diagnosability and maximality of the diagnoser. Different faults and alarms were associated with specific mission phases/modes and configurations of the system, which enabled generation of specific alarms (and recoveries)

for each configuration. The specification was validated, by performing diagnosability analysis on the system model. The synthesis routines were run on a system composed of 11 components, with 10 faults in total, and overall 90 bits of variables, and generated an FDI component with 754 states. Finally, the correctness of the diagnoser was verified by using model-checking routines. Synthesis and verification capabilities have been implemented on top of the nuXmv model checker. We remark that the ability to define trace diagnosable alarms was crucial for the synthesis of the diagnoser, since most of the modeled faults were not system diagnosable.

A similar approach was undertaken in FAME. The industrial evaluation was carried out on a further elaboration of the Trace Gas Orbiter case study, adapted to take into account timings of fault propagation. The specification of the FDIR requirements and the verification, validation and synthesis process were done in a similar way. As a difference with AUTOGEF, the synthesis of FDI in FAME was aided by the specification of a fault propagation model, in the form of a Timed Failure Propagation Graph (TFPG) [BBC+14a, BCGM15]. The case study investigated fault management related to the feared event 'loss of the spacecraft attitude'. A total of 3 faults, instantiated for two (redundant) instances of the Inertial Management Unit (IMU) component were considered. The synthesis of FDI produced an FDI component with 2413 states.

Successful completion of both projects, and positive evaluations from the industrial partner and ESA, suggest that a significant first step towards a formal model-based design process for FDIR was achieved.

8. Related Work

8.1. **From Synchronous to Asynchronous FDI.** This work is closely related to [BCGT14]. The key difference is that we extended the approach to include the asynchronous composition of the plant with the diagnoser. This extension is useful in practice, since many real-life systems as well as many high-level modeling languages adopt an asynchronous, event-based view. In the synchronous case system and diagnoser share the same time scale, and the diagnoser takes a step every time the system does. In the asynchronous setting, on the other hand, the diagnoser takes a step only when the system exhibits an observable behavior, (i.e., an observable event).

Although this could be seen as a minor difference, it poses nontrivial problems. First of all, since the diagnoser cannot update the value of the alarms at every point in time, we need to restrict the definition of Correctness and Completeness to the occurrence of a synchronization, in which the diagnoser can update the alarms, by introducing observation points and using the *observed* version $\llcorner A \lrcorner$ of A. Similarly, since the diagnoser can update its knowledge of the plant only during synchronizations, also the epistemic operator is considered in the observation points. Therefore, we define K as usual, but then introduce a stronger version $\llcorner K \lrcorner$ that is the basis for most of our definitions.

The synthesis algorithm also needs to take into account multiple transitions from the plant that are executed without synchronization. This is done by introducing the u-transitive closure of the belief states.

Finally, to keep the formalism simple, we modeled the observability of state variables as observable events. This is mainly due to the fact that a change in observable state variables requires the introduction of a new synchronization event between the plant and the

diagnoser in order to allow the diagnoser to update its knowledge. This idea is consistent with the approach defined in [SSL+96]. Also, in other works on knowledge in an asynchronous setting (e.g., [vdM07]), the fact that the observer sees every observable state changes implicitly assumes that the observable state change triggers a synchronization. Note that this is somehow different from asynchronous systems with shared variables, where a process can see the change of the shared variable only when/if scheduled.

We notice that the synchronous case can be embedded in the asynchronous one. In fact, according to Def. 2.2, a synchronous product is obtained by making all events of the plant observable: $E_o^P = E^P$. This implies that all points are observation points. Therefore, $\llcorner A \lrcorner = A$, and the restriction of K to observation points has no effect. Also the u-transitive closure has no effect, and we see that $b^* = b$.

8.2. **FDI Specification.** In order to formally verify the effectiveness of an FDI component as part of an overall fault-management strategy, both a formal model of the FDI component (e.g., as an automaton) and of its expected behavior (requirements) is required. Contrary to works related to diagnosis compilation, we are also interested in verifying that an FDI satisfies a given specification. This has tremendous value when we consider the problem of checking whether an existing system (that is familiar to the system designer) satisfies the specification and thus is functionally equivalent to an automatically synthesized one (that could be complex and hard to understand).

Previous works on formal FDI development have considered the specification and synthesis in isolation. Our approach differs with the state of the art because we provide a comprehensive view on the problem. Due to the lack of specification formalism for diagnosers, the problem of verifying their correctness, completeness and maximality was, to the best of our knowledge, unexplored.

Concerning specification and synthesis, [JK01] is close to our work. The authors present a way to specify the diagnoser using LTL properties, and present a synthesis algorithm for this specification. However, problems such as maximality and trace diagnosability are not taken into account. Another remarkable difference is that [JK01] considers diagnosis conditions with future operators. This enables the definition of alarms that predict the occurrence of an event (i.e. prognosis), that is currently not captured in our work.

8.3. **Diagnosability.** In many practical situations it is not possible to require system diagnosability, due, for example, to critical pairs that exists only in a particular configuration of the system. We introduce the concept of trace diagnosability, that is a distinguishing feature of our approach, and overcomes a strong limitation in the current state-of-the-art.

The idea of using epistemic properties to analyze the diagnosability of a system had been already proposed in [ELMV11] and [Hua13]. Notably, the latter extends the problem to a probabilistic setting, and draws a link with the classical definition of diagnosability, introducing the idea of L-diagnosability (that is equivalent to our finite-delay diagnosability). Our approach extends these works by considering other types of delay and the problem of trace diagnosability. Moreover, we do not focus only on the diagnosability problem, but also provide a way of specifying the diagnoser and characterize its completeness in terms of epistemic temporal logic.

We extend the results on diagnosability checking from [CPC03] in order to provide an alternative way of checking diagnosability and redefine the concept of diagnosability at the trace level.

8.4. **Runtime Verification.** The main difference between diagnosis and runtime verification is the partial observability of the plant. Works on runtime verification assume [HR04] that the properties to be verified are expressed over observable variables of the system. In diagnosis, instead, we define the properties over non-observable parts of the system and then ask whether it is possible to infer them by looking at the observable part of the system. Therefore, while some approaches for runtime verification do not need a model of the system (i.e., black-box approach), in diagnosis we need to have some information about the behavior of the system. Finally, in [BLS11] the authors propose the use of a three-valued LTL variant to define whether a trace satisfies a property, does not satisfy it or whether there is not enough information to come to a conclusion. This might resemble the approach presented in Section 6 by our synthesis algorithm. However, the difference is substantial. Every time our diagnoser is uncertain, it means that there are two traces σ_1 and σ_2 that are observationally equivalent, but one satisfies the property and the other does not. However, if we could have an oracle that would tell us whether the system is in σ_1 or in σ_2, we could state (without uncertainty) whether the property is satisfied or not. In [BLS11] instead, the inconclusiveness of the monitor is intrinsic in the fact that the given trace does neither satisfy nor violate the property.

9. Conclusions and Future Work

This paper presents a formal approach for the design of FDI components, that covers many practically-relevant issues such as delays, non-diagnosability and maximality. The design is based on a formal semantics provided by temporal epistemic logic and can be used both in a synchronous and asynchronous setting. We cover the specification, validation, verification and synthesis steps of the FDI design, and discuss the applicability of the approach on a case-study from aerospace. To the best of our knowledge, this is the first work that provides a formal and unified view to all the phases of FDI design.

In the future, we plan to explore the following research directions. First, we will extend FDI to deal with infinite-state systems. Secondly, we will experiment with different assumptions on the memory requirements for the diagnoser, i.e., relax the perfect recall assumption.

Another interesting line of research is the development of optimized reasoning techniques for temporal epistemic logic. The idea is to consider the fragment that we are using, both for verification and validation, and to evaluate and improve the scalability of the synthesis algorithms.

Finally, we will work on integrating the FDI component with the recovery procedures.

References

[ANY+12] E. Alaña, H. Naranjo, Y. Yushtein, M. Bozzano, A. Cimatti, M. Gario, R. de Ferluc, and G. Garcia. Automated generation of FDIR for the compass integrated toolset (AUTOGEF). In *Proc. DAta Systems In Aerospace, DASIA 2012*, volume ESA SP 701, 2012.

[AUT] AUTOGEF. Dependability Design Approach for Critical Flight Software. `https://es.fbk.eu/projects/autogef`.

[BBC+14a] B. Bittner, M. Bozzano, A. Cimatti, R. de Ferluc, M. Gario, A. Guiotto, and Y. Yushtein. An Integrated Process for FDIR Design in Aerospace. In *Proc. IMBSA 2014*, volume 8822 of *LNCS*, pages 82–95, 2014.

[BBC+14b] B. Bittner, M. Bozzano, A. Cimatti, R. de Ferluc, M. Gario, A. Guiotto, and Y. Yushtein. FAME: A Model-Based Environment for FDIR Design in Aerospace. In *Short and Tutorial Proceedings of IMBSA 2014*, pages 61–62, 2014.

[BCGM15] M. Bozzano, A. Cimatti, M. Gario, and A. Micheli. SMT-Based Validation of Timed Failure Propagation Graphs, 2015.

[BCGT13] M. Bozzano, A. Cimatti, M. Gario, and S. Tonetta. Formal Specification and Synthesis of FDI through an Example. In *Workshop on Principles of Diagnosis (DX'13)*, pages 174–179, 2013. Available at URL http://www.dx-2013.org/dx13-proceedings.pdf.

[BCGT14] M. Bozzano, A. Cimatti, M. Gario, and S. Tonetta. Formal Design of Fault Detection and Identification Components Using Temporal Epistemic Logic. In E. Ábrahám and K. Havelund, editors, *Proceedings of TACAS'14*, volume 8413 of *Lecture Notes in Computer Science*, pages 326–340, Grenoble, France, 2014. Springer.

[BLS11] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for ltl and tltl. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4):14, 2011.

[CCD+14] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In *Computer Aided Verification*, pages 334–342. Springer, 2014.

[CPC03] A. Cimatti, C. Pecheur, and R. Cavada. Formal Verification of Diagnosability via Symbolic Model Checking. In *IJCAI*, pages 363–369, 2003.

[CRST12] A. Cimatti, M. Roveri, A. Susi, and S. Tonetta. Validation of requirements for hybrid systems: A formal approach. *ACM Transactions on Software Engineering and Methodology*, 21(4):22, 2012.

[Dim09] Cătălin Dima. Revisiting satisfiability and model-checking for ctlk with synchrony and perfect recall. In *Computational Logic in Multi-Agent Systems*, pages 117–131. Springer, 2009.

[dKK04] J. de Kleer and J. Kurien. Fundamentals of model-based diagnosis. In *Proceedings of IFAC Safeprocess*, volume 3, pages 25–36, 2004.

[ELMV11] J. Ezekiel, A. Lomuscio, L. Molnar, and S.M. Veres. Verifying Fault Tolerance and Self-Diagnosability of an Autonomous Underwater Vehicle. In *IJCAI*, pages 1659–1664, 2011.

[Eur10] European Space Agency. ESTEC ITT AO/1-6570/10/NL/LvH "Dependability Design Approach for Critical Flight Software", 2010.

[Eur11] European Space Agency. ESTEC ITT AO/1-6992/11/NL/JK "FDIR Development and Verification & Validation Process", 2011.

[Eur13] European Space Agency. ESTEC ITT AO/1-7263/12/NL/AK "Hardware-Software Dependability for Launchers", 2013.

[FAM] FAME. FDIR Development and Verification & Validation Process. `https://es.fbk.eu/projects/fame`.

[FKN+10] A. Feldman, T. Kurtoglu, S. Narasimhan, S. Poll, and D. Garcia. Empirical evaluation of diagnostic algorithm performance using a generic framework. *International Journal of Prognostics and Health Management Volume 1*, page 24, 2010.

[GARK07] A. Grastien, A. Anbulagan, J. Rintanen, and E. Kelareva. Diagnosis of discrete-event systems using satisfiability algorithms. In *AAAI*, pages 305–310, 2007.

[GFB+14] A. Guiotto, R. De Ferluc, M. Bozzano, A. Cimatti, M. Gario, and Y.Yushtein. Fame process: A dedicated development and V&V process for FDIR. In *Proc. DAta Systems In Aerospace, DASIA 2014*, volume ESA SP 725 of *European Space Agency, (Special Publication)*, 2014.

[GM04] P. Gammie and R. Van Der Meyden. Mck: Model checking the logic of knowledge. *Computer Aided Verification*, pages 256–259, 2004.

[HD05] Jinbo Huang and Adnan Darwiche. On compiling system models for faster and more scalable diagnosis. In *Proceedings Of The National Conference On Artificial Intelligence*, volume 20, page 300, 2005.

[HR04] K. Havelund and G. Roşu. Efficient monitoring of safety properties. *International Journal on Software Tools for Technology Transfer*, 6(2):158–173, 2004.

[Hua13] X. Huang. Diagnosability in concurrent probabilistic systems. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, 2013.

[HV89] J. Halpern and M. Vardi. The complexity of reasoning about knowledge and time. lower bounds. *Journal of Computer and System Sciences*, 38(1):195–237, 1989.

[JK01] S. Jiang and R. Kumar. Failure diagnosis of discrete event systems with linear-time temporal logic fault specifications. In *IEEE Transactions on Automatic Control*, pages 128–133, 2001.

[LMS02] F. Laroussinie, N. Markey, and P. Schnoebelen. Temporal logic with forgettable past. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, pages 383–392. IEEE Computer Society, 2002.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In Rohit Parikh, editor, *Logics of Programs*, volume 193, pages 196–218. Springer Berlin Heidelberg, 1985.

[Pnu77] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977.

[Sch04] A. Schumann. Diagnosis of discrete-event systems using binary decision diagrams. *Workshop on Principles of Diagnosis (DX'04)*, pages 197–202, 2004.

[SSL+95] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. C. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9), 1995.

[SSL+96] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. C. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, 4(2):105–124, 1996.

[vdM07] R. van der Meyden. What, Indeed, Is Intransitive Noninterference? In *Computer Security - ESORICS 2007, 12th European Symposium On Research In Computer Security, Dresden, Germany, September 24-26, 2007, Proceedings*, pages 235–250, 2007.