Formal Specification and Synthesis of FDI through an Example

Marco Bozzano and **Alessandro Cimatti** and **Marco Gario** and **Stefano Tonetta**
Fondazione Bruno Kessler, Trento, Italy
e-mail: {bozzano,cimatti,gario,tonettas}@fbk.eu

Abstract

The correct operation of complex critical systems is increasingly relying on the ability to detect whether (and which) faults are occurring. This task, known as Fault Detection and Identification (FDI), is typically carried out by an FDI component that, observing the sequence of values conveyed by some predefined observables, triggers a set of predefined alarms. An effective FDI system can provide vital information to steer the containment and recovery of faults.

We recently proposed a formal framework to support the design of FDI for discrete event systems. The framework is based on a logical language for the specification of FDI requirements and it captures problems such as local diagnosability and maximality, by relying on a semantics based on temporal epistemic logic. This enables the formal verification of the specification and the synthesis of correct-by-construction FDI components.

In this paper, we show the merits of the framework by applying it on a simple example, working out the details of the synthesis algorithm, and by reporting the experience on an industrial case-study from the aerospace domain.

1 Introduction

The correct operation of complex critical systems (e.g. trains, satellites, cars, or industrial plants) increasingly relies on the ability to detect when and which faults occur, since an effective diagnostic system can provide vital information to drive the containment of faults and their recovery. This task, known as Fault Detection and Identification (FDI), is typically carried out by an FDI component that triggers a set of predefined alarms based on the sequence of values conveyed by some predefined observables.

Faults are often not directly observable, thus we can detect their occurrence only by observing the effects they have on the observable parts of the system. Moreover, faults manifest in different ways and they might interact with each other in complex ways. For these reasons, the design of FDI components is an extremely hard task.

The state of the practice lacks a structured and effective methodology to design FDI components. These are often completed late in the system development cycle, by relying on past success stories. The lack of effective validation tools often results in conservative assumptions, so that the overall system features overly simple behaviors. This difficulty is witnessed in the aerospace sector by a recent ITT issued by the European Space Agency [European Space Agency, 2010], which strongly motivated our work, in particular, by stressing the importance of FDI in terms of alarms that are meaningful for performing the recovery of the system.

In a recent work [Bozzano *et al.*, 2013], we proposed a comprehensive framework to support the design of FDI for discrete event systems. The framework is based on a *logical language* for the specification of FDI requirements. The language is based on common patterns for FDI properties, that allow us to specify a wide class of practical requirements, such as alarms corresponding to the occurrence of a given condition in a given preceding time interval. Patterns are formalized in terms of temporal operators (as in Linear Temporal Logic [Pnueli, 1977]) and epistemic operators (such as certainty, or knowledge [Halpern and Vardi, 1989]). The temporal epistemic logic is in turn interpreted over the system being observed.

The approach covers two novel and orthogonal directions, which are important for the specification of FDI requirements. One direction is the ability to express *local diagnosability*, i.e. the diagnosability condition for which the diagnoser is required to satisfy the alarm specification is localized to traces. The other direction is the *maximality* of the diagnoser, that is, the ability of the diagnoser to raise an alarm as soon as and whenever possible. The formal specification languages allows the verification of a given diagnoser with respect to a given specification, and to automatically synthesize a diagnoser from a given specification. We refer to [Bozzano *et al.*, 2013] for a comparison with related work.

Goal of this paper is to present the approach defined in [Bozzano *et al.*, 2013] through the elaboration of a simple case study: the magicbox example. The magicbox is a grid-like structure, where a ball is able to jump from one cell to another according to a predefined pattern. We work-out the details of the specification and of the algorithm for the synthesis of a correct-by-construction FDI component. The synthesis algorithm is based on the exploration of the space of *belief states*, which are annotated with alarms taking into account the temporally extended requirements. In this paper we focus on providing the motivation and an intuitive description of the framework, while we refer the reader to [Bozzano *et al.*, 2013] for a more comprehensive technical description.

The applicability of the framework is also discussed by

presenting preliminary experiences on an industrial case-study coming from aerospace.

This paper is structured as follows. Section 2 provides some introductory background and the magicbox example. Section 3 describes the syntax and semantics of the specification language. In Section 4, we detail the synthesis algorithm. The results of evaluating our implementation of the framework are presented in Section 5. Section 6 concludes the paper with a hint on future work.

2 Background

In the following we use the word *system* to indicate the composition of the plant and the FDI component; we also use *diagnoser* and *FDI component* interchangeably.

Transition Systems

Plants and FDIs are represented with transition systems. A transition system is a tuple $S = \langle V, V_o, W, W_o, I, T\rangle$, where V is the set of state variables, $V_o \subseteq V$ the set of observable state variables; W the set of input variables, $W_o \subseteq W$ the set of observable input variables; I is the initial formula over V, T is the transition formula over V, W, V' (with V' being the next version of the state variables).

A state s is an assignment to the state variables V. We denote with s' the corresponding assignment to V'. An input i is an assignment to the input variables W. The observable part $obs(s)$ of a state s is the projection of s on the subset V_o of observable state variables. The observable part $obs(i)$ of an input i is the projection of i on the subset W_o of observable input variables. Given an assignment a to a set of variables X and $X_1 \subseteq X$, we denote the projection of a over X_1 with $a_{|X_1}$. Thus, $obs(s) = s_{|V_o}$ and $obs(i) = i_{|W_o}$.

A trace of S is a sequence $\pi = s_0, i_1, s_1, i_2, s_2, \ldots$ of states and inputs such that s_0 satisfies I and, for each $k \geq 0$, $\langle s_k, i_{k+1}, s_{k+1}\rangle$ satisfies T. The observable part of π is $obs(\pi) = obs(s_0), obs(i_1), obs(s_1), obs(i_2), obs(s_2), \ldots$.

Synchronous composition

Let $S^1 = \langle V^1, V^1_o, W^1, W^1_o, I^1, T^1\rangle$ and $S^2 = \langle V^2, V^2_o, W^2, W^2_o, I^2, T^2\rangle$ be two transition systems with $\emptyset = (V^1 \setminus V^1_o) \cap V^2 = V^1 \cap (V^2 \setminus V^2_o) = (W^1 \setminus W^1_o) \cap W^2 = W^1 \cap (W^2 \setminus W^2_o)$. We define the synchronous product $S^1 \times S^2$ as the transition system $\langle V^1 \cup V^2, V^1_o \cup V^2_o, W^1 \cup W^2, W^1_o \cup W^2_o, I^1 \wedge I^2, T^1 \wedge T^2\rangle$. 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.

We say that S^1 is compatible with S^2 iff i) for every initial state s^2 of S^2, there exists an initial state s^1 of S^1 such that $s^1_{|V^1_o \cap V^2_o} = s^2_{|V^1_o \cap V^2_o}$ and ii) for every reachable state $s^1 \times s^2$ of $S^1 \times S^2$, for every transition $\langle s^2, i^2, s'^2\rangle$ of S^2, there exists a transition $\langle s^1, i^1, s'^1\rangle$ such that $i^1_{|W^1_o \cap W^2_o} = i^2_{|W^1_o \cap W^2_o}$ and $s'^1_{|V^1_o \cap V^2_o} = s'^2_{|V^1_o \cap V^2_o}$.

Diagnoser

We can now introduce the idea of diagnoser, that is a machine D that synchronizes with observable traces of the plant P. D has a set $\mathcal{A}$ of Boolean alarm variables that are activated in response to the monitoring of P. In other terms, a diagnoser can be seen as a function that maps a sequence of observations into a set of alarms: $obs^* \to 2^{\mathcal{A}}$.

Formally, given a set $\mathcal{A}$ of alarms and a plant transition system $P = \langle V^P, V^P_o, W^P, W^P_o, I^P, T^P\rangle$, a diagnoser is a deterministic transition system $D(\mathcal{A}, P) = \langle V^D, V^D_o, W^D, W^D_o, I^D, T^D\rangle$ such that: $V^P_o \subseteq V^D_o$, $W^P_o \subseteq W^D_o$, $\mathcal{A} \subseteq V^D_o$ and D is compatible with P. When clear from the context, we use D to indicate $D(\mathcal{A}, P)$. Only observable variables can be shared among the two systems and used to perform synchronization. This gives raise to the problem of partial observability: the diagnoser cannot perfectly track the evolution of the original system. This makes the diagnoser synthesis problem hard.

Linear Temporal Logic

Our specification framework is based on Temporal Epistemic Logic, in order to describe reasoning about the knowledge of the diagnoser. We will only use a subset of this logic in this paper, namely Linear Temporal Logic (LTL) with Past operators, to talk about *diagnosis conditions*.

Given a set of propositional variables $\mathcal{P}$, a formula in LTL with Past is defined as

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

where $p \in \mathcal{P}$. We define the usual abbreviations of propositional logic $\vee$,$\rightarrow$ and $\leftrightarrow$, moreover introduce two additional abbreviations: $Y^n\beta = YY^{n-1}\beta$ (with $Y^0\beta = \beta$) and $O^{\leq n}\beta = \beta \vee Y\beta \vee \cdots \vee Y^n\beta$.

The semantics of LTL is given in terms of traces, we denote with $\sigma \models \phi$ the fact that the trace σ satisfies the formula ϕ:

- $\sigma, i \models p$ iff $\sigma[i] \models p$
- $\sigma, i \models \neg\phi$ iff $\sigma, i \not\models \phi$
- $\sigma, i \models \phi \wedge \psi$ iff $\sigma, i \models \phi$ and $\sigma, i \models \psi$
- $\sigma, i \models Y\phi$ iff $i > 0$ and $\sigma, i-1 \models \phi$
- $\sigma, i \models O\phi$ iff for some $0 \leq j \leq i$, $\sigma, j \models \phi$
- $\sigma, i \models G\phi$ iff for all $j \geq 0$, $\sigma, j \models \phi$
- $\sigma, i \models F\phi$ iff for some $j \geq i$, $\sigma, j \models \phi$
- $\sigma, i \models X\phi$ iff $\sigma, i+1 \models \phi$

Intuitively the Yesterday and Once operator make it possible to talk about events in the previous time step, or in the whole past history of the system (respectively). Combining these operators, we can create complex scenarios.

Magicbox

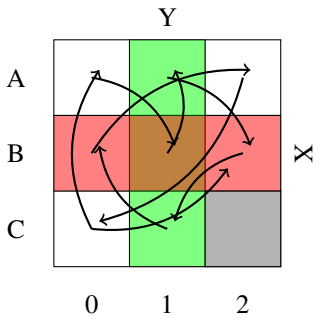

Figure 1: The running example magicbox

A magicbox [Giunchiglia and Ghidini, 1998] is a grid-like structure, in which a ball is able to jump from one cell to another according to a predefined pattern. The movement of the ball is not observable directly, but only through two types of observation points: row and columns. An observer on a row is able to understand when the ball is in its row. However, the observer cannot say anything about the "distance" of the ball, therefore no information on the column

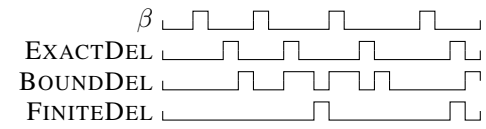

Figure 2: Alarm responses for different delays

can be deduced by this observer. Similarly, the column observers can tell when the ball is in their column, but nothing more. Our running example is the magicbox in Figure 1. This is a 3x3 magicbox, with two observers. The observer x is able to tell when the ball is in the row B, while the observer y is able to tell when the ball is in the column 1. The trajectory of the ball is represented by the arrows, e.g., from A0 the ball will go to B1. To keep the example simple, we blocked the cell C2.

We want to develop a diagnoser that is able to detect the passage of the ball through a certain cell, even if the cell is not observable by itself. Although the ball path is predefined, there is non-determinism in the initial location of the ball and in the possible non-deterministic transitions. For example, from C0 the ball can either go to A0 or to B2.

3 ASL_K

The Epistemic Alarm Specification Language (ASL_K) makes it possible to specify a set of properties that define when the FDI is correct and complete with regard to a set of alarms, and the level of identification required with respect to a set of faults.

An alarm specification is composed of two parts: the *diagnosis condition* and the *diagnosis delay*. The diagnosis condition is the situation to monitor; this includes a fine-grained distinction between fault detection and fault identification. The diagnosis delay is the allowed delay between the occurrence of the diagnosis condition and its detection; it might be the case that the occurrence of a fault can go undetected for a certain amount of time. Our goal is to define clearly how long this interval can be at most.

Let us assume to have a set $\mathcal{P}$ of propositions representing either faults or elementary conditions for the diagnosis. The set $\mathcal{D}_\mathcal{P}$ of *diagnosis conditions* over $\mathcal{P}$ is an LTL Past formula β.

A diagnosis condition $\beta \in \mathcal{D}_\mathcal{P}$ is a condition of the plant that we want to detect. If β is a fault, the fault must be identified. If β is a disjunction of faults, the detection must be performed, but the identification of the single faults is not required.

An *alarm specification* φ is used to specify how the occurrence of a diagnosis condition β determines the raising of an alarm A. This involves three aspects: the delay, the maximality of the alarm, and the diagnosability of the diagnosis condition. For diagnosability, we will consider two possible values: $diag \in \{local, global\}$; for maximality we consider $max \in \{True, False\}$.

We consider three kinds of delay, that provide the name for the templates in ASL_K:

- $\text{EXACTDEL}_K(A, \beta, n, diag, max)$
- $\text{BOUNDDEL}_K(A, \beta, n, diag, max)$
- $\text{FINITEDEL}_K(A, \beta, diag, max)$

Intuitively, the formalization of these template (cf. [Bozzano *et al.*, 2013]) includes two aspects: completeness and correctness. Completeness means that whenever the diagnosis condition is satisfied, the alarm will hold in the future (i.e., no false negative). Correctness means that whenever the alarm holds then the diagnosis condition was met earlier (i.e., no false positives). Specifying the delay provides a more fine-grained way of quantifying the time that an alarm can be silent before considering it as a false negative. In Figure 2, we show the diagnosis condition β in the first row, and the responses for different delay types: exact- (with $n = 2$), bounded- (with $n = 4$) and finite-delay. Lets consider the exact-delay specification: completeness says that whenever the diagnosis condition β is satisfied, the alarm A will hold exactly $n = 2$ steps afterwards; correctness says that whenever the alarm A holds then the diagnosis condition β was met $n = 2$ steps earlier.

If the plant is non-diagnosable for β, then the completeness part of the specification will never be satisfied. This is the classical understanding of diagnosability, and we call this *global* diagnosability [Sampath *et al.*, 1996]. Intuitively, however, it might be possible to contain the non-diagnosability of the system to a subset of the plant. Therefore, we introduce the concept of *local* diagnosability. The idea is to evaluate the diagnosability of a diagnosis condition on single traces instead that on the entire plant. In practice, we require our diagnoser to raise an alarm in all those situations in which there is no uncertainty, given a sequence of observations.

An interesting problem with bounded- and finite-delay specifications is the problem of *maximality*. Lets consider the specification $\text{BOUNDDEL}_K(A, \beta, 4, local, False)$. After the occurrence of β a diagnoser can satisfy this specification by raising the alarm at any given time before 4 time-step pass. However, this does not constraint the amount of time the alarm should stay up. Indeed, the alarm could go up, down and up again within the window of 4 time-steps and its behavior would be considered correct. A graphical representation of this situation is depicted in Figure 3, where the first row represents the diagnosis condition, and the second row represents a maximal alarm.

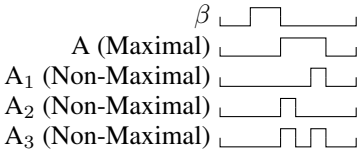

Figure 3: Comparison of non-maximal alarms

Although all the alarms in Figure 3 are correct and complete, some of them introduce an unnecessary delay between the occurrence of the diagnosis condition β and the alarm A; moreover, multiple raising of A could be misinterpreted as multiple occurrences of β.

We address this problem by defining the concept of *maximality*. Intuitively, a diagnoser is maximal if it raises the alarm as soon as possible and keeps it up as much as possible. Adding this requirement makes it possible to specify the unique behavior that a diagnoser must have w.r.t. an alarm specification; therefore, it becomes possible to verify the diagnoser by comparing it against another one.

Example Specification For our magicbox example, we can define several specifications. For example, we could define several diagnosis conditions. The obvious way of do-

ing so is by cell, e.g., $\beta_{C1} = C1$. This could be compared to fault identification, in which we are interested in knowing exactly where the ball is. However, we might loose specificity and say that we want to know if the ball is in one of a given set of cells, e.g., the column A: $\beta_A = A0 \vee A1 \vee A2$. This flexibility makes it possible to specify common problems like fault detection, fault isolation and fault identification.

In ASL_K it is also possible to define temporal properties as diagnosis condition. There are two ways of getting into B2, therefore, we might be interested in knowing whether we arrived from C0 rather than from A1; this can be expressed with the temporal condition $\beta = B2 \wedge YCO$, where Y is the LTL operator "yesterday". The ability of specifying complex scenarios as a diagnosis condition, makes it possible to obtain more informative alarms.

Once we defined the diagnosis conditions, we need to express the acceptable delay, the maximality and type of diagnosability for the alarm. Our example specification consists of the following alarms: $\mathcal{A} = \{$

$$\varphi_1 = \text{EXACTDEL}_K(A(B1), \beta_{B1}, 0, global, True),$$
$$\varphi_2 = \text{EXACTDEL}_K(A(C1), \beta_{C1}, 0, local, True),$$
$$\varphi_3 = \text{BOUNDDEL}_K(A(B0), \beta_{B0}, 2, local, True)\}$$

In the next section we will see how to build a diagnoser that satisfies the specification $\mathcal{A}$.

4 Synthesis of Diagnoser

In [Bozzano *et al.*, 2013] we present an algorithm that is able to deal with any ASL_K specification, by handling both diagnosable and non-diagnosable systems, and producing a maximal diagnoser. In this section we review the main concepts of the algorithm and apply it to our running example.

The idea of the Belief Explorer algorithm is to generate an automaton that keeps track of the possible states in which the plant could be after the given observations. Similarly to [Schumann, 2004], this translates into generating the power-set of the states of the plant, and define a suitable transition relation among the elements of this set. We call the power-sets defined in this way *belief states*. Each belief state of the automaton can be annotated with the alarms that are satisfied in all the states of the belief state. The automaton, together with the annotations, can be encoded symbolically obtaining the diagnoser.

The diagnoser obtained with this procedure is compatible with the plant, maximal and correct ([Bozzano *et al.*, 2013]).

Belief Automaton

Given a plant $P = \langle V^P, V_o^P, W^P, W_o^P, I^P, T^P \rangle$, let S be the set of states of P. The *belief automaton* is defined as $\mathcal{B}(P) = \langle B, E, B_0, T_b \rangle$ where $B = 2^S$, $E = 2^{W_o^P \cup V_o^P}$ and $B_0 \subseteq B$ and $T_b : (B \times E) \to B$ are defined as follows.

We define $B_0 = \{b \mid$ there exists $u \in 2^{V_o^P}$ such that for all $s \in b$, $s \models I^P$ and $obs(s) = u\}$, meaning that each initial belief state is compatible with one of the possible observations of the system. We implicitly assume that we can initialize the diagnoser by observing the state of the system.

The transition function T_b is defined as follows $b' = T_b(b, e) = \{s' \mid \exists s \in b$ such that $\langle s, i, s' \rangle \models T^P$, $obs(s') = e_{|V_o^P}, obs(i) = e_{|W_o^P}\}$. Intuitively, the belief state b' is a successor of b iff all the states in b' are compatible with the observations from a state in b.

Annotation process

Each of the belief states can now be annotated with the alarms, by checking for entailment of the temporal property $\tau(\varphi)$. To explain this procedure we first consider the simplest case $\varphi = \text{EXACTDEL}(A_\varphi, \psi, 0)$, where $\tau(\varphi) = \psi$ is a propositional formula. We can explore the belief automaton, and annotate with A_φ all the states b that are consistent in the satisfaction of $\tau(\varphi)$:

$$b \models A_\varphi \text{ iff } b \models \tau(\varphi) \text{ iff } \forall s \in b.s \models \tau(\varphi)$$

Note that it might occur that neither $b \models \tau(\varphi)$ nor $b \models \neg\tau(\varphi)$. This is the case if in the belief state there is at least one system state in which $\tau(\varphi)$ holds and one in which it does not; this situation is a witness of *uncertainty* in the belief state, caused by by non-diagnosability.

All other alarms specifications can be reduced to the previous case by performing a pre-processing step to the belief automaton construction. For each alarm specification φ, we add a monitor variable $\overline{\tau}$ in the plant obtaining a plant P', s.t. $P' = P \times (G(\tau \leftrightarrow \overline{\tau}))$, where we abuse notation to indicate the automaton that encodes the monitor variable. Any alarm specification on P can be redefined on P' as $\varphi' = \text{EXACTDEL}(A_\varphi, \overline{\tau}, 0)$, so that $D \times P \models \varphi$ iff $D \times P' \models \varphi'$.

Example

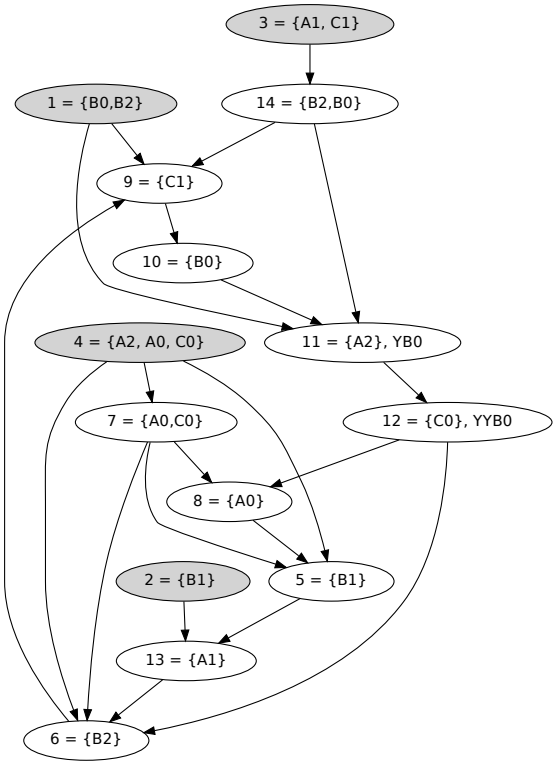

Figure 4: The belief automaton of the running example

Figure 4 shows the belief automaton for our running example, and the specification $\mathcal{A}$ defined in the previous section. To keep the diagram simple, we did not add the observables on the edges. Note that in the diagram state_2 and state_5 have the same belief state but are handled differently because state_2 is an initial state.

At the beginning we do not know where the ball is, therefore our first observation splits the initial belief state in 4 possible belief states (gray nodes) (Figure 5).

X	Y	State
0	0	$4 = \{A2, A0, C0\}$
0	1	$3 = \{A1, C1\}$
1	0	$1 = \{B0, B2\}$
1	1	$2 = \{B1\}$

Figure 5: Initial states observations

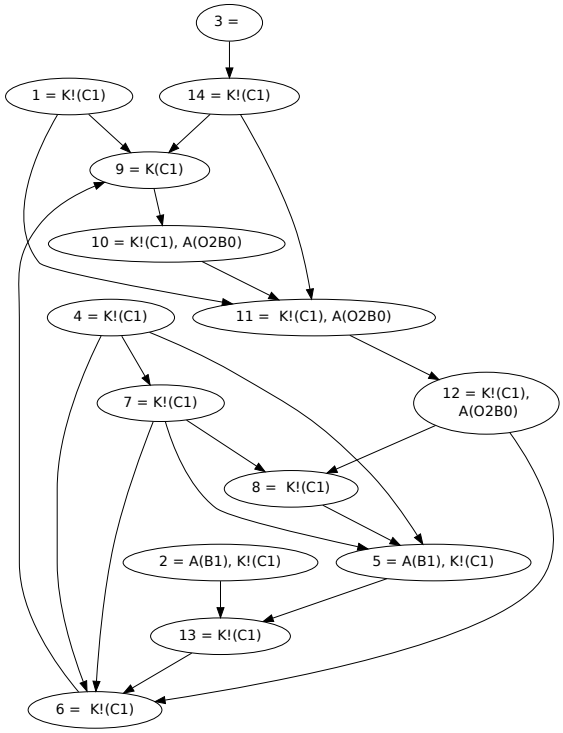

Figure 6: The diagnoser

Lets take the most ambiguous initial state: state_4. This state has transitions to state_5, state_6 and state_7. Each of these transitions will reduce the uncertainty on the location of the ball; in particular, in state_5 and state_6 we will have certainty on the ball location (B1 and B2 respectively). In state_7 we can say for sure that the ball is in the column 0: $\text{state}_7 \models (A0 \vee B0 \vee C0)$.

Finally, note that state_{11} and state_{12} contain the monitor variables $YB0$ and $YYB0$, that will be used for handling the specification $\varphi_3 = \text{BOUNDDEL}_K(A(B0), \beta_{B0}, 2, local, True)$.

Annotations Once we have constructed the belief automaton, we can navigate it and add the alarm A_φ whenever the temporal formula $\tau(\varphi)$ holds. The result of the annotation process is presented in Figure 6, where for clarity we show only the positive annotations. Lets take the first of our specifications: $\varphi_1 = \text{EXACTDEL}_K(A(B1), \beta_{B1}, 0, global, True)$. We annotate with $A(B1)$ all the states b in which $b \models \tau(\varphi_1) = B1$: state_2 and state_5; all other states are marked with the negation of the alarm $!A(B1)$.

Diagnosability For the second specification $\varphi_2 = \text{EXACTDEL}_K(A(C1), \beta_{C1}, 0, local, True)$ we proceed in a similar way. However, we realize that there is only one state b in which $b \models C1$ holds, i.e., state_9. In state_3 we have an ambiguity between A1 and C1. If our specification were to require global diagnosability, state_3 would be a witness of the non-diagnosability of the system. However, our specification requires local diagnosability. Therefore, we annotate state_9 with $A(C1)$ and all other states with $\neg A(C1)$. It should be clear, that using the same annotation ($\neg A(C1)$) for both state_3 and (e.g.) state_{14} is counter-intuitive. In state_3, we do not know whether the ball is in C1, while in state_{14} we are *sure* that the ball is not in C1. To handle this situation, we break the specification φ_2 in two parts:

$$\varphi_{2+} = \text{EXACTDEL}_K(K(C1), \beta_{C1}, 0, local, True)$$
$$\varphi_{2-} = \text{EXACTDEL}_K(K!(C1), \neg\beta_{C1}, 0, local, True)$$

Intuitively, we express the epistemic operator in the alarm name, therefore encoding all situations of interest: the diagnoser *Knows* that C1 holds, or it *Knows* that it does not. We can now annotate state_9 with $K(C1)$, all the other states (except state_3) with $K!(C1)$ and then complete the annotations with $!K(C1)$ and $!K!(C1)$. After performing the annotation, in state_3 both the alarms $K(C1)$ and $K!(C1)$ will not hold, due to the non-diagnosability. This is particularly useful, because now we can react accordingly; for example, a recovery module might act in a different way knowing whether there is uncertainty on the occurrence of $C1$.

Splitting the specification in a positive and negative part is even more interesting for bounded specification. Lets consider $\varphi_3 = \text{BOUNDDEL}_K(A(B0), \beta_{B0}, 2, local, True)$. The diagnosis condition is diagnosable with delay 2. We have uncertainty in state_{14}, but then in state_{11} we know that $KO^{\leq 2}B0$ holds (due to the monitoring variable $YB0$). Therefore, we could use the global diagnosability version of this specification. However, it might be interesting to have both positive and negative alarms, in order to annotate state_{14} and state_1 with the information that $O^{\leq 2}B0$ *might* hold there, and distinguish them from all other states in which we are sure that it does not hold (e.g., state_5).

Another situation in which local diagnosability is needed, are the initial states. For example, we have uncertainty for $C1$ in state_3, however, this is the only situation in which we have uncertainty on $C1$. Excluding this state would give us diagnosability, and this is exactly the idea behind local diagnosability.

Maximality In section 3 we introduced the idea of maximality, to express the completeness w.r.t. what the diagnoser could know given a series of observations. φ_3 is a good candidate to explain this concept more in detail.

In Figure 6 we show a maximal annotation. The annotation $A(O2B0)$ is given to state_{10}, state_{11} and state_{12}. This means that, for each state in which $\tau(\varphi) = O^{\leq 2}\beta_{B0}$ holds, we have the annotation $A(O2B0)$. We could argue that having the alarm $A(O2B0)$ only in state_{11} would be correct too. Even more, we would still be correct if we had the alarm on state_{10} and state_{12} only. The three formalizations satisfy our definition of non-maximal bounded-delay specification (cf. [Bozzano *et al.*, 2013]); however, this situation leaves many details on the behaviour of the diagnoser underspecified, making it hard to use the information provided by the diagnoser to devise (e.g.) a recovery or containment procedures. Therefore, maximality is useful to provide a clear and un-ambiguous specification of the diagnoser, and it enables the verification of a diagnoser by comparison of the outputs with another diagnoser.

5 Experimental Evaluation

The algorithm presented in this paper has been implemented on top of the NuSMV model-checker, making it possible to specify and synthesize a diagnoser. The belief explorer synthesis procedure has been implemented by using BDD's [Bryant, 1986]. BDDs make it easy to identify already visited belief states; moreover, having a symbolic characterization of the belief-state makes it possible to check for entailment of an alarm in a belief-states ($b \models \tau(\varphi)$) with a single operation.

To show the feasibility of our approach, we present in this section an experimental evaluation conducted on magic boxes, and discuss the application of these techniques to an industrial case study.

Magicbox Benchmark

In our experimental evaluation we generated random magicboxes by considering the following variables: size of the magicbox $N \in [2, 50]$; number of observables ($M = \lfloor N * k \rfloor$), with $k \in [0.1, 0.9]$ and number of specifications $S \in [1, 20]$.

For several combinations of (N,M,S) we measured the runtime of the belief explorer, and we noticed the (expected) exponential growth w.r.t. the size of the model. Moreover, we noticed how the percentage of observables influences the runtime and memory usage: few observables give rise to huge belief states, where the uncertainty is substantial; many observables, instead, will make it possible to quickly reduce the uncertainty and generate less and smaller belief states. Finally, adding monitoring variables to the system to does not seem to impact significantly the performances.

AUTOGEF

This framework has been used as the base for the European Space Agency AUTOGEF project [European Space Agency, 2010].

The main goal of the project was to explore the possibility of defining requirements for an on-board Fault Detection, Identification and Recovery component (FDIR) and perform the synthesis of an FDIR satisfying the requirements. The problem was tackled by synthesizing the Fault Detection (FDI) and Fault Recovery (FR) separately. The FDI needs to provide enough information to the FR to act accordingly. A case-study for evaluating the approach was defined and evaluated by Thales Alenia Space based on the EXOMARS Trace Gas Orbiter.

The model and the requirements were formalized within AUTOGEF, the FDIR component generated. The system is composed by 11 components with 10 possible faults in total. The generated FD had 754 states. Crucial for the synthesis of the diagnoser was the ability of defining locally diagnosable alarms, since most of the modeled faults were not globally diagnosable.

The correctness of the synthesized FDIR is guaranteed by construction, however, we are currently working in using epistemic model checking techniques to be able to verify the correctness of any given FDI module against the specification.

6 Conclusions and Future Work

Goal of this paper is to illustrate the features of the formal framework for the specification, verification, and synthesis of Fault Detection and Identification components [Bozzano *et al.*, 2013]. We explored the application of the framework on a toy example, and worked out the details of the synthesis algorithm. Finally, we discussed the applicability of the framework also presenting our preliminary experience on an industrial case-study coming from aerospace.

In the future, we plan to extend the framework to deal with asynchronous and infinite-state (timed/hybrid) systems, and with the specification and synthesis of distributed and hierarchical FDI components.

Acknowledgments We would like to thank Régis de Ferluc for providing us the case-study for AUTOGEF.

References

[Bozzano *et al.*, 2013] M. Bozzano, A. Cimatti, M. Gario, and S. Tonetta. A formal framework for the specification, verification and synthesis of fdi components. In *Technical Report*, 2013. Available at URL https://es.fbk.eu/people/gario/TR-FBK-ES-13a.pdf.

[Bryant, 1986] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Trans. on*, 100(8):677–691, 1986.

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

[Giunchiglia and Ghidini, 1998] F. Giunchiglia and C. Ghidini. Local models semantics, or contextual reasoning = locality + compatibility. In *KR'98*, pages 282–291. Morgan Kaufmann, 1998.

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

[Pnueli, 1977] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

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

[Schumann, 2004] A. Schumann. Diagnosis of discrete-event systems using binary decision diagrams. *DX*, pages 197–202, 2004.