Kalpa Publications in Computing

Volume 4, 2018, Pages 169–183

28th International Workshop on
Principles of Diagnosis (DX'17)

Diagnosing Discrete Event Systems Using Nominal Models Only

Yannick Pencolé[1], Gerald Steinbauer[2], Clemens Mühlbacher[2], and Louise Travé-Massuyès[1]

[1] LAAS-CNRS, Toulouse, France
[2] Graz University of Technology, Graz, Austria

Abstract

Complex technical systems usually show a dynamic behavior that is often conveniently represented with a discrete event model. Such a behavior is the result of dynamic components which interact with each other. Due to the complexity of technical systems faults are not totally avoidable. In order to deal with such faults diagnosing the system at run-time is of great interest. To perform such a diagnosis it is common to use fault models. Such models are in practice often hard to obtain. To address this problem we show a diagnosis approach for discrete event systems which uses the model of the nominal behavior only. In order to perform this diagnosis we adopt the well known idea of consistency based diagnosis.

1 Introduction

Technical systems usually show a certain dynamic behavior. Examples range from the simple control of an elevator to complex systems such as production lines or autonomous robots. Most of these systems have in common that they consist of components which show dynamic behavior. Through the interaction of the different components and their dynamics an overall dynamic behavior of the system emerges that fulfills the given tasks.

System are never designed or implemented without flaws or exposed to an environment which may trigger undesired behaviors. If a flaw manifests without being detected this may lead to catastrophic consequences. Therefore, it is essential to diagnose if system components are malfunctioning. In order to to automate the process of diagnosis, model-based approaches tailored towards different formalisms are available [1], [2], [3]. A classification of diagnosis techniques used across the different modeling frameworks can be found in [4].

Often these approaches are designed in a way that requires a description of the faulty behavior of a component in order to generate a diagnosis. Thus the nominal as well as the faulty system behavior needs to be known and modeled. The latter can be very difficult to obtain and it is not always possible to know all faulty behaviors of a component in advance. Furthermore, it is a cumbersome task to model all these behaviors. This is even more of a problem in the field of discrete event systems (DES) because all the established diagnosis theories require fault models [5].

This paper is concerned with dynamic systems whose dynamics can be conveniently abstracted as a DES (see the introduction chapter in [6]). To address the problems coming from the need for

faulty behavior models, we propose a consistency-based diagnosis approach for DES. The approach allows modeling the system as interacting dynamic components just using the nominal behavior of the components. Thus neither knowledge about fault modes is needed nor the faulty behavior has to be modeled.

The main contribution of this paper is that we show how the concept of consistency-based diagnosis with nominal behavior models only [1] can be adopted for dynamic systems comprising components and their behaviors modeled as DES.

Furthermore, we show how to perform a conflict-driven search to calculate diagnoses.

The remainder of the paper is organized as follows. The next section introduces a simple running example which is used to show the principle of the proposed consistency-based diagnosis approach. In Section 3 the proposed consistency based diagnosis method for DES is defined in a formal way, following the well known idea of consistency-based diagnosis of [1]. In the proceeding section a conflict-driven search to calculate the diagnoses is presented. In Section 5 we briefly discuss related research. Finally we conclude the paper and point out some future work.

2 Running Example

We will use a baggage transfer system, denoted by BTS, as a running example throughout the paper. The system comprises the following eight components as shown in Figure 1:

- two conveyor belts C_1 and C_2 actuated by two motors,
- one piston P and its controller $Cont$,
- four sensors S_k, S_a, S_b, and S_c.

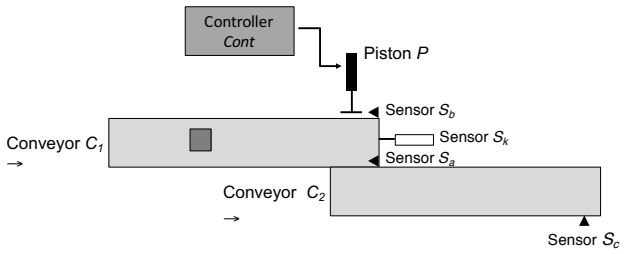

Figure 1: Running example: the baggage transfer system (BTS).

The transfer of the baggage is initiated by putting it on the left end of conveyor C_1. After the transfer the baggage can be removed from conveyor C_2.

The piston transfers the baggage from C_1 to C_2. It is controlled to move out, respectively in, by the commands $pistonOut$, respectively $pistonIn$, sent by the controller $Cont$. The sensors S_a and S_b are associated with the piston and correspond to the outer and inner limit switches. The BTS system has a sensor S_k for the presence of the baggage at the right end of conveyor C_1. The sensor S_c denotes the presence of it at the right end of conveyor C_2, which makes it possible to load the next baggage on C_1. We assume that the baggage is removed from C_2 in time. The nominal behavior of the controller is that it pushes the baggage using the piston form C_1 to C_2 once sensor S_k recognizes it. To control the movement out, respectively in, of the piston the input of sensor S_a, respectively S_b, is used.

Once we have introduced the modelling of the system formally in the Section 3 we will present a formal model representation of the components' behavior.

3 Consistency Based Diagnosis

In this section we will introduce the proposed consistency based diagnosis approach for DES formally. In this paper we make the following basic assumptions:

- *Assumption 1*: a faulty component does not produce any other events than the ones already in the model,
- *Assumption 2*: the synchronization between components works always correctly,
- *Assumption 3*: any event e of a component is only generated by this component.

3.1 Component description

When modelling the behavior of a system we follow a component-based modelling schema. The normal behavior of the individual components is described using a deterministic finite automaton.

Definition 1 (Model of a component)**.** *The model of a component c_i is an automaton $A_i = (Q_i, E_i, T_i, q_{0_i})$ where Q_i is a finite set of states, E_i is a set of events, the transition function $T_i : Q_i \times E_i \rightarrow Q_i$, and q_{0_i} is the initial state.*

Let $T_i^* : Q_i \times E_i^* \rightarrow Q_i$ denote the transitive closure of T_i (i.e. $T_i^*(\tau e) = T_i^*(\tau)T_i(e), \tau \in E_i^*, e \in E_i$ and $T_i^*(\varepsilon) = \varepsilon$)). A component c_i then generates the prefix-closed language $\mathcal{L}(A_i)$ composed of the traces τ such that $T_i^*(\tau)$ is defined in the automaton A_i of component c_i. A component is associated to a local observation mask $obs_i : E_i \rightarrow E_{O_i} \cup \{\varepsilon\}$. If an event $e \in E_i$ is observable then it means $obs_i(e) \neq \varepsilon$ and the event $obs_i(e)$ is observed. In the following, without loss of generality, we consider that if $obs_i(e) \neq \varepsilon$ then $obs_i(e) \neq e$. In this case, it means that the set E_i is divided into the two disjoint sets E_{O_i} and E_{U_i} representing the observable, respectively the unobservable events.

In this framework, we assume that the system is composed of n components $C = \{c_1, \ldots, c_n\}$. By Assumption 3, any set of events E_i is disjoint from any other set of events $E_j, i, j \in \{1, \ldots, n\}, j \neq i$.

The automata A_{C_1} and A_{C_2} representing the normal behaviors of conveyors C_1 and C_2 are shown in Figure 2. The events $Left1$ ($Left2$), $Right1$ ($Right2$), and $Empty1$ ($Empty1$) represent the circumstances that the baggage entered the conveyor on the left, reached the right end, and was removed from the conveyor. None of these events are observable.

The automata A_P and A_{Cont} representing the normal behaviors of the piston P and controller $Cont$ are shown in Figure 3. The events $moveOut$, respectively $moveIn$, represents that the piston received the command to move out, respectively in, while the events $endOut$, respectively $endIn$, represents the fact that the piston reached its outer, respectively inner, terminal position. The controller alternated between the commands for sending the piston out and in. The commands emitted by the controller are observable. Please note that the global behavior of the system is determined by the synchronization of events we will discuss below. Observable events are shown in bold.

The automata A_{S_a} and A_{S_k} representing the normal behaviors of sensors S_a and S_k are shown in Figure 4. Sensor S_a emits the event a once it sensed the piston at its inner terminal position represented by event $sensA$. Sensor S_k emits the event k once it sensed that the baggage reached the end of conveyor C_1 represented by event $sensK$. The events a and k are observable.

The automata A_{S_b} and A_{S_c} representing the normal behaviors of sensors S_b and S_c are shown in Figure 5. Sensor S_b emits the event b once it sensed the piston at its outer terminal position represented by event $sensB$. Sensor S_c emits the event c once it sensed that the baggage reached the end of conveyor C_2 represented by event $sensC$. The events b and c are observable. The intermediate events wcb and wcc are used to model the fact that normally event b occurs before event c as pulling back the piston takes less time than moving baggage along conveyor C_2.

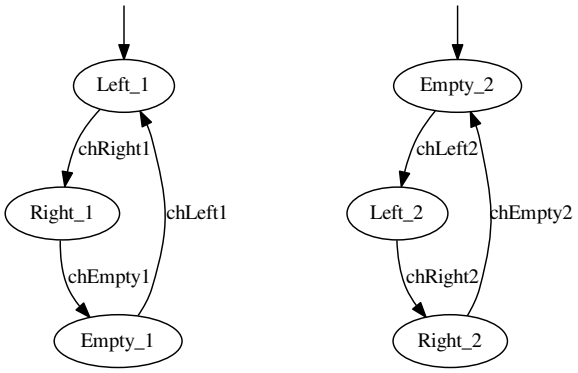

Figure 2: Normal model automata A_{C_1} and A_{C_2} of conveyors C_1 (left) and C_2 (right).

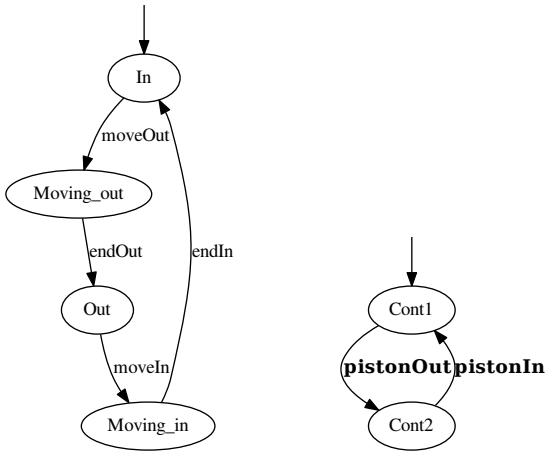

Figure 3: Normal model automata A_P and A_{Cont} of the piston P (left) and its controller $Cont$ (right). Observable events are shown in bold.

3.2 System description

In order to model the system it is now required to model the interactions between the components. We follow here the way it is defined in [7] and implemented in the software DIADES [8]. The components interact with each other by the synchronization of some events. The interaction model between components is then ruled by a synchronization product that relies itself on a set of synchronization rules

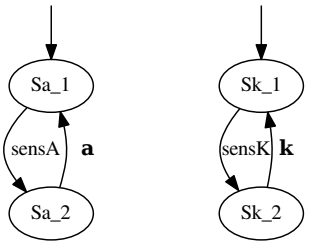

Figure 4: Normal model automata A_{S_a} and A_{S_k} of sensors S_a (left) and S_k (right). Observable events are shown in bold.

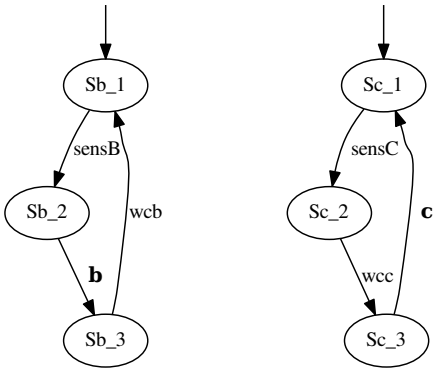

Figure 5: Normal model automata A_{S_b} and A_{S_c} of sensors S_b (left) and S_c (right). Observable events are shown in bold.

$\mathcal{R}$.

Firstly, in order to formally define the generic synchronization product as a synchronised product over the complete set of components, we transform any automaton A_i to the automaton A_i^ε where the associated transition function T_i^ε extends T_i with $\forall q \in Q_i, T_i^\varepsilon(q, \varepsilon) = q$ (such an ε-transition expresses the fact that the component c_i stays in state q while other components evolve).

A synchronisation rule r is a constraint $ev_{i_1} = ev_{i_2} = \cdots = ev_{i_k}$ where any $ev_i \in E_i$ and any $i_j \neq i_l, 2 \leq k \leq n$. A rule r intuitively states that if an event ev_{i_1} occurs in the component c_{i_1} then an event ev_{i_2} (resp. ev_{i_k}) occurs in the component c_{i_2} (resp. c_{i_k}) at the same time. A rule r then implicitly defines a synchronised event $e_{||r}$ over the system as follows. The event $e_{||r}$ is the n-uple $e_{||r} = (e_1, \ldots, e_n)$ such that for any $i \in \{1, \ldots, n\}$, $e_i = \varepsilon$ if r does not involve any event of component c_i or $e_i = ev_i$ if ev_i is in r. Let $E_{sync} = \prod_{i=1}^{n}(E_j \cup \varepsilon)$ be the Cartesian product of the events involved

in the automata A_i^ε, then the event $e_{||r}$ is part of it. Consider now a set of synchronization rules $\mathcal{R}$, it implicitly represents:

1. the set of synchronized events $e_{||r}$, $r \in \mathcal{R}$;
2. any synchronized event of type $(\varepsilon, \ldots, \varepsilon, e_i, \varepsilon, \ldots, \varepsilon)$, $e_i \neq \varepsilon$ such that e_i is not involved in any rule of $\mathcal{R}$ (event e_i is not synchronized with any other event).

Finally, the synchronization rules $\mathcal{R}$ generate the following set of synchronized events $E_\mathcal{R} \subseteq E_{sync}$:

$$E_\mathcal{R} = \{e_{||r} : r \in \mathcal{R}\} \cup \{(\varepsilon, \ldots, \varepsilon)\} \cup$$
$$\{(e_1, \ldots, e_n) : \exists j \in \{1, \ldots, n\}, e_j \in E_j,$$
$$\forall r \in \mathcal{R}, e_j \notin r \wedge \forall i \neq j, e_i = \varepsilon\}.$$

Based on a set of synchronization rules $\mathcal{R}$, we can now formally define the synchronization product between components.

Definition 2 (Operator $\|_\mathcal{R}$)**.** *The synchronized product of $A_1, \ldots, A_n$ with respect to a set of synchronization rules $\mathcal{R}$ denoted by $A_1 \|_\mathcal{R} \cdots \|_\mathcal{R} A_n$ is defined as the automaton $\mathcal{A} = (Q_\mathcal{A}, E_\mathcal{A}, T_\mathcal{A}, q_{0_\mathcal{A}})$ with $Q_\mathcal{A} \subseteq Q_1 \times \cdots \times Q_n$, $E_\mathcal{A} = E_\mathcal{R}$, $q_{0_\mathcal{A}} = (q_{0_1}, \ldots, q_{0_n})$, and the transition function $T_\mathcal{A} : T_\mathcal{A}((q_1, \ldots, q_n), (e_1, \ldots, e_n)) = T_1^\varepsilon(q_1, e_1) \times \cdots \times T_n^\varepsilon(q_n, e_n)$ with $q_i \in Q_i$ and $e_i \in E_i \cup \{\varepsilon\}$ if all $T_i(q_i, e_i)$, $i = 1, \ldots, n$ are defined. $T_\mathcal{A}$ is undefined otherwise.*

The use of rules $\mathcal{R}$ allows for a generic definition of the synchronization product as it defines a specific subspace $E_\mathcal{R}$ of E_{sync}. For instance, if $\mathcal{R} = \varnothing$, then none of the events are synchronised, $\mathcal{R} = \varnothing$ defines the free product of the components. On the other hand, if any rules of $\mathcal{R}$ contains n events (no ε) and any event of any component is a least in a rule of r then $\mathcal{R}$ implements a synchronous product where all the components always evolve simultaneously.

Proposition 1. *The synchronized product operator $\|_\mathcal{R}$ with respect to a set of synchronization rules $\mathcal{R}$ is commutative and associative.*

Proof. By definition $(A_j \|_\mathcal{R} A_i, i \neq j)$ can be obtained by index permutation of $A_i \|_\mathcal{R} A_j$ (see Definition 2). Associativity holds by definition. ■

Table 1 depicts the synchronization rules used to model the global normal behavior of the running example.

r_1	$< chRight1, sensK >$
r_2	$< \mathbf{k}, \mathbf{pistonOut}, moveOut >$
r_3	$< endOut, sensA, chEmpty1, chLeft2 >$
r_4	$< \mathbf{a}, \mathbf{pistonIn}, moveIn >$
r_5	$< endIn, sensB >$
r_6	$< chRight2, sensC >$
r_7	$< \mathbf{c}, chEmpty2, chLeft1 >$
r_8	$< wcc, wcb >$

Table 1: Synchronization rules for the BTS. Observable events are marked in bold.

Following the terminology of [1], the model of the system is as follows.

Definition 3 (System)**.** *A system comprises: (1) a set of components $C = \{c_1, \ldots, c_n\}$, (2) a system description $SD = (\{A_1, \ldots, A_n\}, \mathcal{R})$ where the A_i's are the automata representing the normal behavior of the components c_i's, and $\mathcal{R}$ is a set of synchronization rules.*

The global normal behavior of the system is given by $\mathcal{A}$ which is the synchronized product of the automata in SD. The language generated by the system is $\mathcal{L}(\mathcal{A})$, also denoted by $\mathcal{L}(SD)$.

Figure 6 depicts the synchronized product of the models of the running example components, representing the global normal behavior of the BTS.

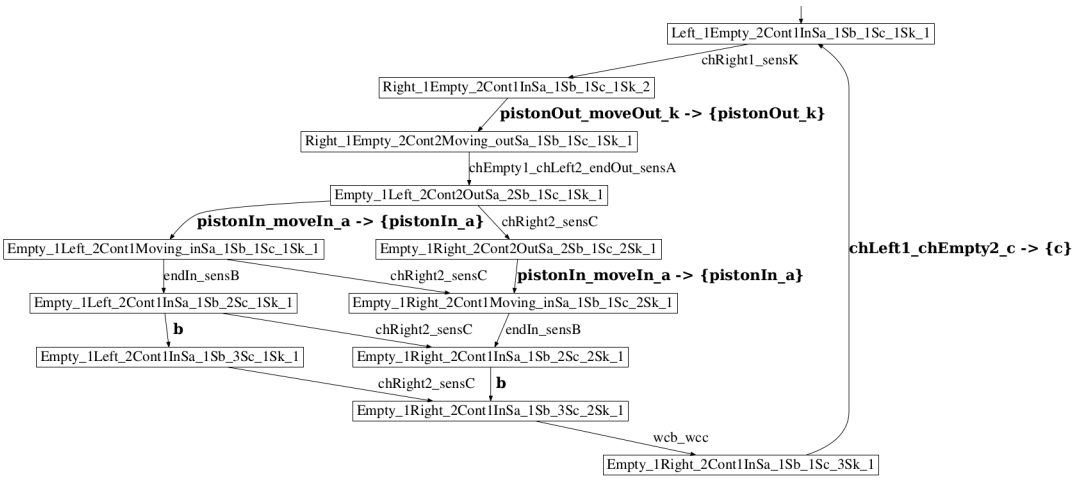

Figure 6: Automaton $\mathcal{A}_{BTS}$ representing the global normal behavior of the BTS system. Observable synchronizing events are mapped to observable events only.

3.3 Consistency based diagnosis problem

This subsection aims at defining the consistency based diagnosis problem over a discrete event system defined by (C, SD). First, we define what is observable within SD by an observation mask [9] also called a viewer in [10].

Definition 4 (Observation mask)**.** *The observation mask $obs : E_{\mathcal{R}} \rightarrow \prod_{i=1}^{n}(E_{O_i} \cup \{\varepsilon\})$ maps a synchronized event to a synchronized observable event or to $(\varepsilon, \ldots, \varepsilon)$:*

$$obs((e_1, \ldots, e_n)) = (obs_1(e_1), \ldots, obs_n(e_n)).$$

The observation function can be easily extended for a sequence of events τ: if $\tau = e_1 e_2 e_3 \ldots$ then $obs^*(\tau) = obs(e_1) obs^*(e_2 e_3 \ldots)$. Note that here we do not consider uncertain observations as in [11], the extension of our framework to deal with uncertain observations is straightforward.

The inverse observation mask for a sequence of observable events σ is defined as $obs^{-1}(\sigma) = \{\tau \in E^*_{\mathcal{R}} | obs(\tau) = \sigma\}$.

In the context of discrete event system, the observation OBS is a sequence of events from E_O. In order to define diagnosis formally in our framework we have to define if a sequence of observable events OBS reflects the normal behavior of a system. This is performed by checking the consistency of the system with OBS.

Definition 5 (Consistency)**.** *A system description SD is consistent with a sequence of observations OBS if $obs^{-1}(OBS) \cap \mathcal{L}(SD) \neq \emptyset$.*

A system description SD is consistent with some observation sequence OBS if there is at least one trace τ generated by the system ($\tau \in \mathcal{L}(SD)$) that can produce the observation OBS. In contrast to [1] where this property is based on the consistency of a first-order theory we define consistency as the problem of checking membership in a language.

Let us denote a sequence of events $e_1, e_2 \ldots, e_{n-1}, e_n$ with brackets by $[e_1, e_{2\_} \ldots, e_{n-1}, e_n]$ and a synchronized event $e_r \in E_{\mathcal{R}}$ by underscoring the events that are not ϵ in the n-tuple $(e_1, \ldots, e_n)$, Table 2 depicts a nominal observation sequence OBS_N as well as two faulty observation sequences OBS_{F1} and OBS_{F2} for the BTS. One can verify on Figure 6 that depicts $\mathcal{A}_{BTS}$, i.e. the synchronized product of the automata in $SD_{BTS} = \{A_{C_1} \| A_{C_2} \| A_P \| A_{Cont} \| A_{S_k} \| A_{S_a} \| A_{S_b} \| A_{S_c}\}$, that $obs^{-1}(OBS_N) \cap \mathcal{L}(SD_{BTS}) \neq \emptyset$. For instance, $[chRight1\_senK, pistonOut\_k, chEmpty1\_chLeft2\_$
$endOut\_sensA, chRight2\_senC, pistonIn\_a, endIn\_$
$sensB, b, c] \in obs^{-1}(OBS_N) \cap \mathcal{L}(SD_{BTS})$. This means that SD_{BTS} is consistent with OBS_N. One can also verify that SD_{BTS} is not consistent neither with OBS_{F1} nor with OBS_{F1}.

Nominal (OBS_N)	$[pistonOut\_k, pistonIn\_a, b, c]$
Faulty (OBS_{F1})	$[pistonOut\_k, pistonIn\_a, b, c,$ $pistonOut\_k, pistonIn\_a, b, c,$ $pistonOut\_k, b, c, b]$
Faulty (OBS_{F2})	$[pistonOut\_k, pistonIn\_a, b, c,$ $pistonOut\_k, b, c, pistonOut\_k]$

Table 2: Nominal and faulty observed sequences.

Definition 6 (universal behavior)**.** *The universal behavior Ub_i of a component c_i is an automaton that represents the language of the Kleene closure of the component's events E_i.*

The universal behavior is an automaton that represents all possible traces generated using all events of a component. For instance by replacing a component c_i by its universal behavior we remove all constraints of a component on possible observations. Thus the universal behavior models the nominal behavior as well as every possible faulty behavior of a component. Please note that the universal behavior is built using observable and unobservable events. The later is important to allow the behavior to be part of all potential synchronizations.

Proposition 2. *The following statements hold:*

1. *For any component i, $\mathcal{L}(A_i) \subseteq \mathcal{L}(Ub_i)$,*
2. *For any component i, under Assumption 1, $\mathcal{L}(A_i^f) \subseteq \mathcal{L}(Ub_i)$, where A_i^f is the (unknown) automaton representing any faulty behavior of the component c_i,*
3. *Let $\{c_{i_1}, \ldots c_{i_m}\}$ be any subset of components with $1 < m$, let B_{i_j} be either the automaton A_{i_j} of component c_{i_j} or its universal behaviour Ub_{i_j}, $1 \leq j < m$,*

$$\mathcal{L}(B_{i_1} \| \ldots \| B_{i_{m-1}} \| A_{i_m}) \subseteq \mathcal{L}(B_{i_1} \| \ldots \| B_{i_{m-1}} \| Ub_{i_m})$$

and

$$\mathcal{L}(B_{i_1} \| \ldots \| B_{i_{m-1}} \| A_{i_m}^f) \subseteq \mathcal{L}(B_{i_1} \| \ldots \| B_{i_{m-1}} \| Ub_{i_m}).$$

Proof.

1. Trivial by definition of the universal behaviour Ub_i.
2. We prove the result for $m = 2$. If $B_{i_1} = A_{i_1}$ then let $\tau \in \mathcal{L}(A_{i_1} \| A_{i_2}), |\tau| = n$, it follows that there exists a state $(q_n^{i_1}, q_n^{i_2}) \in Q_{i_1} \times Q_{i_2}$ such that:

$$T^*_{A_{i_1} \| A_{i_2}}((q_0^{i_1}, q_0^{i_2}), (e_1^{i_1}, e_1^{i_2}) \dots (e_n^{i_1}, e_n^{i_2})) = (q_n^{i_1}, q_n^{i_2}).$$

Therefore $T^*_{A_2}(q_0^{i_2}, e_1^{i_2} \dots e_n^{i_2}) = q_n^{i_2}$ in the automaton A_{i_2} which implies that $T^*_{Ub_{i_2}}(q_0^{Ub_{i_2}}, e_1^{i_2} \dots e_n^{i_2}) = q_0^{Ub_{i_2}}$ is true in Ub_{i_2}. By Definition of the synchronization operator $\|$, we get

$$T^*_{A_{i_1} \| Ub_{i_2}}((q_0^{i_1}, q_0^{Ub_{i_2}}), (e_1^{i_1}, e_1^{i_2}) \dots (e_n^{i_1}, e_n^{i_2})) = (q_n^{i_1}, q_0^{Ub_{i_2}})$$

so $\tau \in \mathcal{L}(A_{i_1} \| Ub_{i_2})$. The same reasoning applies when $B_{i_1} = Ub_{i_1}$, hence the result for $m = 2$. For $m > 2$, as $\|$ is associative (see Proposition 1), it suffices to consider first the automaton $B = B_{i_1} \| \dots \| B_{i_{m-1}}$ and to apply the previous reasoning. ■

Definition 7 (Diagnosis)**.** *A diagnosis for the diagnosis problem (SD, C, OBS) is a set $\Delta \subseteq C$ such that $\|\{SD \backslash \{A_{c_i} | c_i \in \Delta\} \cup \{Ub_{c_i} | c_i \in \Delta\}\}$ is consistent with OBS.*

Definition 8 (Minimal diagnosis)**.** *A diagnosis Δ is minimal if there is no strict subset $\Delta' \subset \Delta$ that is a diagnosis.*

Corollary 1. *Consider the system $C = \{c_1, \dots, c_n\}$ for which a subset of components $\Delta = \{c_{i_1}, \dots c_{i_m}\} \subseteq C$, $1 < m$, are faulty according to unknown faulty behaviors $A^f_{i_1}, \dots A^f_{i_m}$ respectively, and the remaining components $\{c_{i_{m+1}}, \dots c_{i_n}\}$ are normal. Denote the language generated by the faulty system by $\mathcal{L}(SD^f) = A^f_{i_1} \| \dots \| A^f_{i_m} \| A_{i_{m+1}} \| \dots \| A_{i_n}$, then if a sequence of observable synchronized events OBS is such that $obs^{-1}(OBS) \in \mathcal{L}(SD^f)$, then $\mathcal{L}(SD') = Ub_{i_1} \| \dots \| Ub_{i_m} \| A_{i_{m+1}} \| \dots \| A_{i_n}$ is consistent with OBS and Δ is a diagnosis.*

Proof. The proof comes directly from Proposition 2 and Definition 7. ■

Proposition 3. *If a sequence of observable synchronized events OBS is inconsistent with the system description SD then $\Delta = C$ is a diagnosis for the diagnosis problem (SD, C, OBS).*

Proof. According to Definition 5, $obs^{-1}(OBS)$ has an empty intersection with $\mathcal{L}(SD)$. According to Definition 7 we have to show that $\mathcal{L}(SD')$ with $SD' = \|\{Ub_i\}$ has a nonempty intersection with $obs^{-1}(OBS)$.

The universal behavior of a component c_i can be represented by the automaton $Ub_i = (\{q_{0_i}\}, E_{c_i} \cup \{\epsilon\}, T, q_{0_i})$ with the total transition function $T : \{q_{0_i}\} \times E_{c_i} \cup \{\epsilon\} \rightarrow \{q0_i\}$. This automaton consists of only one state and has a transition for each event as well as the empty string.

Following Definition 2 the synchronized product of all universal behaviors is represented by the automaton $SD' = (\{q_{0_1} \times \dots \times q_{0_n}\}, E_{Sync}, T, q_{0_1} \times \dots \times q_{0_n})$ with the total transition function $T : \{q_{0_1} \times \dots \times q_{0_n}\} \times E_{\mathcal{R}} \rightarrow \{q_{0_1} \times \dots \times q_{0_n}\}$. As the universal behaviors allow all individual events in the only one state, in the synchronized product the synchronized events are the full set of $E_{\mathcal{R}}$. Therefore, this automaton consists of only one state and has a transition for each possible synchronized event. Thus, the language represented by SD' is $\mathcal{L}(SD') = {E_{\mathcal{R}}}^*$.

Due to the assumption that a faulty system does not produce any other observable synchronization events than the ones in $E_{O_{Synch}}$, we know that $obs^{-1}(OBS)$ is nonempty because at least OBS is a proper inverse mapping. Given this we can show that $obs^{-1}(OBS)$ has a nonempty intersection with $\mathcal{L}(SD') = {E_{\mathcal{R}}}^*$. Thus SD' with Δ and OBS are consistent and therefore following Definition 7, $\Delta = C$ is a diagnosis for diagnosis problem (SD, C, OBS). ■

Consider the running example of the BTS and the observation sequence OBS_{F1}. We have already assessed earlier that OBS_{F1} is inconsistent with SD_{BTS}. Hence, from Proposition 3, $C_{BTS} = \{C_1, C_2, P, Cont, S_k, S_a, S_b, S_c\}$ is a diagnosis. But this is only usefull to detect that there is a faulty component. Now, if we use Definition 8 and Corollary 1, we find that $\Delta_1 = \{P\}$, and $\Delta_2 = \{S_b\}$ are two minimal diagnoses and we can therefore isolate the faulty components. These results can be physically explained from the expected behavior of the BTS, the interaction of its components illustrated on Figure 1, and the events that are observed.

Let us retrieve the whole scenarios for both diagnoses. The observed events are given by the sequence $OBS_{F1} = [pistonOut\_k, pistonIn\_a, b, c, pistonOut\_k, pistonIn\_a, b, c, pistonOut\_k, b, c, b]$. The subsequence $[pistonOut\_k, pistonIn\_a, b, c,]$ repeats twice and it is consistent with $\mathcal{L}(SD_{BTS})$ since it corresponds to OBS_N. After this subsequence has repeated, the synchronized event $pistonOut\_k$ is observed, which means that a baggage is sensed at the right end of conveyor C_1 trigering the controller $Cont$ to send $pistonOut$, which commands the piston P to move out. At this point, the scenarios of the two diagnoses diverge.

Scenario for diagnosis $\Delta_1 = \{P\}$. The observation of event b can be explained by *the piston being back at its inner limit without having reached its outer limit*, indicating a faulty piston. Note that otherwise the event a or the event c would have been observed. *The piston then moves out again without receiving the command* $pistonOut$ *from the controller* $Cont$, which pushes the baggage at the left end of conveyor C_2. The event c indicating that the baggage has travelled all the way down C_2 is then issued. Finally, *the piston shows a faulty behavior again and moves back and forth* so that b is issued again. We can therefore conclude that the faulty piston alone is consistent with the observed sequence OBS_{F1}.

Scenario for diagnosis $\Delta_2 = \{S_b\}$. The observations of event b can be explained by *sensor* S_b *emitting* b *erronously*. In the meanwhile, the piston finishes its stroke which pushes the baggage at the left end of conveyor C_2. The event c indicating that the baggage has travelled all the way down C_2 is then issued. Finally, *sensor* S_b *gets crazy again and* b *is emitted again*.We can therefore conclude that sensor S_b faulty alone is consistent with the observed sequence OBS_{F1}.

4 Calculating Diagnoses

This section aims at characterizing within the proposed framework a notion of *conflict* that can be used to compute the minimal diagnoses in a similar way as for static systems.

Definition 9 (Conflict)**.** *A conflict set is a set of components* $\Gamma := \{c_1, \ldots, c_k\} \subseteq C$ *such that* $\|\{A_\Gamma \cup \{Ub_i | c_i \in C \backslash \Gamma\}\}$ *is inconsistent with* OBS.

Definition 10. *A conflict* Γ *is minimal if no subset* $\Gamma' \subset \Gamma$ *is a conflict.*

Let C_1 be a set of components of C and $C_2 = C \setminus C_1$, let $\mathcal{L}(C_1, C_2, \sigma)$ denote the language $\{\tau \in \mathcal{L}((\|_{c_i \in C_1} A_i) \| (\|_{c_i \in C_2} Ub_i)) \cap obs^{-1}(\sigma)\}$. Any word of $\mathcal{L}(C_1, C_2, \sigma)$ is a run consistent with the observation σ where the components of C_2 have been replaced by their repsective universal behaviors.

Lemma 1. *For any subsets of components* $C_1, C_2 \subseteq C$, $\mathcal{L}(C_1 \cup C_2, C \setminus (C_1 \cup C_2), \sigma) \subseteq \mathcal{L}(C_1, C \setminus C_1, \sigma)$.

Proof. The result is obvious if $C_2 \subseteq C_1$. Consider now that $C_2 \not\subseteq C_1$. Let us denote $C_1 = \{c_{i_1}, \ldots, c_{i_{m_1}}\}$, $C_2 \setminus C_1 = \{c_{j_1}, \ldots, c_{j_{m_2}}\}$ and $C \setminus (C_1 \cup C_2) = \{c_{k_1}, \ldots, c_{k_{m_3}}\}$. Any run τ in $\mathcal{L}(C_1 \cup C_2, C \setminus (C_1 \cup C_2), \sigma)$ is a run of

$$\mathcal{L}(A_{i_1} \| \ldots \| A_{i_{m_1}} \| Ub_{k_1} \| \ldots \| Ub_{k_{m_3}} \| A_{j_1} \| \ldots \| A_{j_{m_2}}).$$

By Proposition 2, the run τ is also in

$$\mathcal{L}(A_{i_1} \| \ldots \| A_{i_{m_1}} \| Ub_{k_1} \| \ldots \| Ub_{k_{m_3}} \| A_{j_1} \| \ldots \| A_{j_{m_2-1}} \| Ub_{j_{m_2}})$$

and by commutativity, it is in

$$\mathcal{L}(A_{i_1} \| \ldots \| A_{i_{m_1}} \| Ub_{k_1} \| \ldots \| Ub_{k_{m_3}} \| A_{j_1} \| \ldots \| Ub_{j_{m_2}} \| A_{j_{m_2-1}}).$$

Apply $m_2 - 1$ times this reasoning, it follows that the run τ is a run of

$$\mathcal{L}(A_{i_1} \| \ldots \| A_{i_{m_1}} \| Ub_{k_1} \| \ldots \| Ub_{k_{m_3}} \| Ub_{j_1} \| \ldots \| Ub_{j_{m_2-1}} \| Ub_{j_{m_2}}).$$

Therefore, $\tau \in \mathcal{L}(C_1, C \setminus C_1, \sigma)$ as $obs^*(\tau) = \sigma$. ■

Proposition 4. *Let MCS be the set of minimal conflicts, a diagnosis Δ is minimal iff Δ is a minimal hitting set of MCS.*

Proof ($\Rightarrow$) We prove first that a minimal diagnosis Δ is a minimal hitting set of MCS. Consider a minimal conflict Γ of MCS, then by definition:

$$\mathcal{L}(\Gamma, C \setminus \Gamma, \sigma) = \varnothing.$$

From Lemma 1, it follows that $\forall C' \subseteq C$,

$$\mathcal{L}(\Gamma \cup C', C \setminus (\Gamma \cup C'), \sigma) = \varnothing.$$

As Δ is a diagnosis, then there is no set $C' \subseteq C$ such that $\Delta = C \setminus (\Gamma \cup C')$ which means that $\Delta \cap \Gamma \neq \varnothing$ for any conflict Γ of MCS. Now, suppose that there exists in Δ a component c that does not belong to any conflict of MCS. As Δ is a minimal diagnosis,

$$\mathcal{L}(C \setminus \Delta \cup \{c\}, \Delta \setminus \{c\}, \sigma) = \varnothing.$$

So $C \setminus \Delta \cup \{c\}$ is a conflict while $C \setminus \Delta$ is not. Any minimal conflict that would be included in $C \setminus \Delta \cup \{c\}$ would contain c and would be in MCS, hence the contradiction. Till now, we already prove that Δ is a hitting set of MCS, let us prove that it is a minimal one. If Δ is not a minimal hitting set of MCS, there must exist a component c such that for any conflict Γ in MCS, $\Delta \setminus \{c\} \cap \Gamma \neq \varnothing$. As Δ is a minimal diagnosis, it follows that

$$\mathcal{L}(C \setminus \Delta \cup \{c\}, \Delta \setminus \{c\}, \sigma) = \varnothing$$

which means that $C \setminus \Delta \cup \{c\}$ is a conflict. However, for any conflict Γ in MCS, $\Gamma \not\subseteq C \setminus \Delta \cup \{c\}$ (as $\Delta \setminus \{c\} \cap \Gamma \neq \varnothing$). So $C \setminus \Delta \cup \{c\}$ must contain a minimal conflict that is not in MCS, hence the contradiction.

($\Leftarrow$) Let $MCS = \{\Gamma_1, \ldots, \Gamma_k\}$. For any $i \in \{1, \ldots, k\}$, as $\mathcal{L}(\Gamma_i, C \setminus \Gamma_i, \sigma) = \varnothing$, Lemma 1 asserts that any supset $\Gamma_i \cup C'$ is also a conflict

$$\forall C' \subseteq C, \mathcal{L}(\Gamma_i \cup C', C \setminus (\Gamma_i \cup C'), \sigma) = \varnothing.$$

Moreover any possible conflict is such a set (if not, it would mean that there exists a minimal conflict that is not in MCS). Δ is a minimal hitting set of MCS so for any i in $\{1, \ldots, k\}$, $\Delta \cap \Gamma_i \neq \varnothing$ thus $\Gamma_i \not\subseteq C \setminus \Delta$. Therefore, $C \setminus \Delta$ is not a conflict:

$$\mathcal{L}(C \setminus \Delta, \Delta, \sigma) \neq \varnothing$$

so Δ is a diagnosis.

Consider now $c_i \in \Delta$. As Δ is a minimal hitting set, there exists $\Gamma \in MCS$ such that $\Delta \setminus \{c\} \cap \Gamma = \varnothing$ which means that $\Gamma \subseteq C \setminus (\Delta \setminus \{c\}) = C \setminus \Delta \cup \{c\}$ and then

$$\mathcal{L}(C \setminus \Delta \cup \{c_i\}, \Delta \setminus \{c_i\}, \sigma) = \varnothing.$$

Therefore Δ is a minimal diagnosis. ■

Proposition 4 shows that it is possible to search for minimal diagnoses by first computing the set of minimal conflicts and secondly by computing the set of minimal hitting sets. Strategies for computing such hitting sets, like for instance the one of [12] that is used in the model-based diagnostic engine DITO [13], can be straightforwarldy implemented for the consistency based diagnosis of discrete event systems that we introduce in this paper.

Consider the running example of the BTS again and now take the observation sequence OBS_{F2}. Like for sequence OBS_{F1}, OBS_{F2} has been shown inconsistent with SD_{BTS} and from Proposition 3, $C_{BTS} = \{C_1, C_2, P, Cont, S_k, S_a, S_b, S_c\}$ is a diagnosis, which indicates that there is a fault. Let us apply the above results to derive the interesting diagnoses, i.e. minimal diagnoses. As stated in Proposition 4, minimal diagnoses can be obtained from minimal conflicts. For the current scenario, there are 192 conflicts (over 255 possible configurations) and 2 of them are minimal: $\Gamma_1 = \{Cont\}$ and $\Gamma_2 = \{P\}$ (note that, by using the strategy proposed in [12], only the consistency of 65 configurations is checked to get the minimal conflicts). From Proposition 4, we therefore obtain one unique diagnosis $\Delta = \{Cont, P\}$ that indicates a double fault on the controller $Cont$ and the piston P.

Let us use the expected behavior of the BTS, the interaction of its components illustrated on Figure 1, and the events that are observed to explain this result physically. The observed events are given by the sequence $OBS_{F2} = [pistonOut\_k, pistonIn\_a, b, c, \ pistonOut\_k, b, c, pistonOut\_k]$. The subsequence $[pistonOut\_k, pistonIn\_a, b, c,]$ is consistent with $\mathcal{L}(SD_{BTS})$ since, as already seen, it corresponds to OBS_N. After this subsequence, the synchronized event $pistonOut\_k$ is observed, which means that a baggage is sensed at the right end of conveyor C_1 trigering the controller $Cont$ to send $pistonOut$, which commands the piston P to move out.

The observation of event b can be explained by *the piston being back at its inner limit without having reached its outer limit*, indicating a faulty piston. Note that otherwise the event $pistonIn\_a$ or the event c would have been observed. *The piston then moves out again without receiving the command* $pistonOut$ *from the controller* $Cont$, which pushes the baggage at the left end of conveyor C_2. The event c indicating that the baggage has travelled all the way down C_2 is then issued. Finally, one can assume that anoter baggage is charged on conveyor C_1 and reaches the right end so that the sensor S_k emits the event k but this event is synchronized with the event $pistonOut$sent by the controller. However, the observation of $pistonOut\_k$ implies that the controller $Cont$ is faulty because the controller cannot emit $pistonOut$ two times in a row.

We can therefore conclude that the observed sequence OBS_{F1} is explained by the double fault $\Delta = \{Cont, P\}$.

5 Related Work

We start our discussion of related research with the discussion of diagnosis discrete event system with known fault modes. In [2] a method was proposed to generate from a system description comprising the nominal as well as the faulty system behavior a so called diagnoser. This diagnoser represents the system behavior using an automaton where the states are labeled with faults which would explain why the system reaches this state from the initial state. This method is on the one hand fast during the runtime as only the diagnoser needs to be traversed. On the other hand the generation of the diagnoser can be complex and probably infeasible. The major difference to our approach is that the approach needs a description of the faulty behavior of the system. Thus as we argued above such a description is not always easy to obtain.

In order to overcome the problem of the generation of a diagnoser for a large system a merging based method was proposed in [14]. The system is described as a set of components connected to each other through communication channels. Each component is described by an automaton describing the normal and the faulty behavior. If a fault occurs the diagnosis is performed by splitting the system into clusters, calculating a local diagnosis for each cluster and afterwards merging the local diagnosis into a global diagnosis. This approach shows several similarities to our approach but as in the case above it uses fault modes to characterize the faulty behavior of the system.

All the above approaches considered the system as a discrete system. In contrast the work proposed in [15], [16] considers the system as a continuous system. The major difference imposed is that through the use of a continuous system one can apply differential equations for a system. Thus the consistency-based diagnosis can be applied by retracing equations which belong to certain components. This is in contrast to our approach which doesn't retract equations but the constraints imposed by an automaton.

As our approach operates on automatons we have a strong relation to Kripke structures. In principle these structures represent states comprising a set of propositions with their truth value and transitions between these states. This representation is very similar to ours. In order to detect an inconsistency of such a structure an update method was proposed in [17]. The method updates the Kripke structure in order to guarantee that the Kripke structure is consistent with a CTL specification. The same idea was also used in [18] by finding a minimal set of updates. Such a minimal set of updates together with the resulting structure is considered as the preferred repair. This is in contrast to our approach as we minimize the faulty components and not the changes to the automaton. Thus a minimal diagnosis in our case can cause a non-minimal set of updates to the automaton and vice versa.

The idea of updating a Kripke structure was also used in [19]. The Kripke structure represented the observations of a conference system. These observations were checked for consistency with an LTL formula, which was used to impose plausibility constraints. After updating the structure to guaranty consistency, those paths in the structure with the minimal distance to the original observations are accumulated to form the preferred histories of events. Thus the diagnosis was on minimal changing the observation history to be consistent with the LTL formula. This is in contrast our approach which changes the system description to conform the observation history.

Due to the usage of LTL formulas to specify the system behavior the correct design of such formulas is of interest. In order to easy the design process a method was proposed in [20] to diagnose LTL formulas. The idea is to determine those operators in the formula which are the root cause of an inconsistency with a given trace. Thus the diagnosis is a set of operators which needs to be revised in order to reflect the trace correctly. The authors propose a consistency-based as well as a fault mode based version, thus enabling a fast or more precise diagnosis. The main difference to our approach is that we use an automaton which as such can represent any regular expression which is not possible through an LTL formula. Furthermore we consider one automaton of the system to be faulty instead of a specific part of this automaton as it was done in the LTL formula.

Work very much related to our approach was presented in [21]. In the paper the authors how Reiter's approach can be generalized for a much broader class of systems including discrete event systems. Following the consistency-based paradigm the space of diagnosis hypotheses is explored for valid diagnoses. Using an order-relation of the hypotheses-space and conflict-driven search diagnoses can be obtained very efficiently. Following the same consistency-based idea and an advanced search pattern in contrast to our approach the method still needs a model of the faults in order to work.

6 Conclusion and Future Work

Usually complex systems show a dynamic behavior that can be model using discrete event systems. When it comes to diagnosis of such systems information about possible faults and the faulty behavior is needed. Such information is usually hard to obtain. In this paper we presented an approach that adopts the well known idea of consistency-based diagnosis for discrete event systems. The idea is that system's components are modeled as automatons which are synchronized to represent the global behavior of the system. In defining the diagnosis we follow Reiter's consistency-based approach where we represent consistency of the system and some observations using the membership of the language of the system description. We introduced the notion of a universal behavior that does not constrain the consistent observations of a component. We showed that using this representation we are able to detect an inconsistent observation and to derive a diagnosis for it. Finally, we defined the notion of conflict related to our representation and showed that we can use minimal hitting sets to calculate diagnoses as well.

In future work we have to investigate how we can use our notion of conflict to calculate diagnoses more efficiently. Moreover, by dropping the need for fault-modes the diagnosis process becomes less focused. Further research needs be done in the relation of our novel representation and properties like diagnosability.

Acknowledgement

The work presented in this paper was partly supported by the visiting professor program of the INSTITUT NATIONAL POLYTECHNIQUE DE TOULOUSE (INP Toulouse), Toulouse, France.

References

[1] Raymond Reiter. A theory of diagnosis from first principles. *Artif. Intell.*, 32(1):57–95, April 1987.

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

[3] Sriram Narasimhan and Gautam Biswas. Model-based diagnosis of hybrid systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 37(3):348–361, 2007.

[4] Alban Grastien. A spectrum of diagnosis approaches. In *The 24th International Workshop on Principles of Diagnosis*, pages 130–135. Citeseer, 2013.

[5] Janan Zaytoon and Stéphane Lafortune. Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37(2):308–320, 2013.

[6] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[7] Yannick Pencolé, Anika Schumann, and Dmitry Kamenetsky. Towards low-cost fault diagnosis in large component-based systems. In *6th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, pages 1473–1478, Beijing, China, 8 2006.

[8] Yannick Pencolé. Fault diagnosis in discrete-event systems: How to analyse algorithm performance? In *Diagnostic reasoning: Model Analysis and Performance*, pages 19–25, Montpellier, France, 8 2012.

[9] Shengbing Jiang, Zhongdong Huang, Vigyan Chandra, and Ratnesh Kumar. A polynomial algorithm for testing diagnosability of discrete-event systems. *Transactions on Automatic Control*, 46(8):1318–1321, 8 2001.

[10] Gianfranco Lamperti and Marina Zanella. Flexible diagnosis of discrete-event systems by similarity-based reasoning techniques. *Artificial Intelligence*, 170(3):232–297, 2006.

[11] Gianfranco Lamperti and Marina Zanella. Diagnosis of discrete-event systems from uncertain temporal observations. *Artificial Intelligence*, 137(1):91 – 163, 2002.

[12] Xiangfu Zhao and Dantong Ouyang. Improved algorithms for deriving all minimal conflict sets in model-based diagnosis. In *Advanced Intelligent Computing Theories and Applications. With Aspects of The- oretical and Methodological Issues. Third International Conference on Intelligent Computing*, pages 157–166, Qingdao, China, aug 2007.

[13] Yannick Pencolé. Dito: a csp-based diagnostic engine. In *21st European Conference on Artificial Intelligence*, pages 699–704, Prague, Czech Republic, 8 2014.

[14] Pietro Baroni, Gianfranco Lamperti, Paolo Pogliano, and Marina Zanella. Diagnosis of large active systems. *Artificial Intelligence*, 110(1):135–183, 1999.

[15] Hwee Tou Ng. Model-based, multiple-fault diagnosis of dynamic, continuous physical devices. *IEEE Intelligent Systems*, (6):38–43, 1991.

[16] Franz Lackinger and Wolfgang Nejdl. Integrating model-based monitoring and diagnosis of complex dynamic systems. In *IJCAI*, pages 1123–1128, 1991.

[17] Franz Wotawa and Bibiane Angerer. Retaining consistency in temporal knowledge bases. In *Advances in Applied Artificial Intelligence*, pages 600–609. Springer, 2006.

[18] Yulin Ding and Yan Zhang. Model updating ctl systems. In *AI 2005: Advances in Artificial Intelligence*, pages 5–16. Springer, 2005.

[19] Bibiane Angerer, Andreas Griesmayer, and Franz Wotawa. Maintaining temporal consistency in a multimedia knowledge base.

[20] Ingo Pill and Thomas Quaritsch. Behavioral diagnosis of ltl specifications at operator level. In *IJCAI*. Citeseer, 2013.

[21] Alban Grastien, Patrik Haslum, and Sylvie Thiébaux. Conflict-Based Diagnosis of Discrete Event Systems: Theory and Practice. In *International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, 2012.