Manifestability Verification of Discrete Event Systems

Lina Ye, Philippe Dague, Lulu He

Manifestability Verification of Discrete Event Systems

Lina Ye[1,2] and **Philippe Dague**[1] and **Lulu He**[1]
[1]LRI, Univ. Paris-Sud, CNRS, Univ. Paris-Saclay, France
[2]CentraleSupélec, Univ. Paris-Saclay, France
e-mail: firstname.name@lri.fr

Abstract

Fault diagnosis is a crucial and challenging task in the automatic control of complex systems, whose efficiency depends on the diagnosability property of a system, allowing one to determine with certainty whether a given fault has effectively occurred based on the available observations. However, this is a quite strong property that generally requires a high number of sensors. Consequently, it is not rare that developing a diagnosable system is too expensive. In this paper, we analyze a new system property called manifestability, that represents the weakest requirement on observations for having a chance to identify on line fault occurrences and can be verified at design stage. Intuitively, this property makes sure that a faulty system has at least one future behavior after fault occurrence observably distinguishable from all normal behaviors. Then, we propose an algorithm with PSPACE complexity to automatically verify it for finite automata. Furthermore, we prove that the problem of manifestability verification itself is PSPACE-complete. The experimental results show the feasibility of our algorithm from a practical point of view. Then, we extend our approach to real-time systems modeled by timed automata. To do this, we redefine manifestability by taking into account time constraints and we prove that this problem for timed automata is undecidable.

1 Introduction

Fault diagnosis is a crucial and challenging task in the automatic control of complex systems, whose efficiency depends on a system property called diagnosability. Diagnosability is a system property describing whether one can distinguish with certainty fault behaviors from normal ones based on sequences of observable events emitted from the system. In a given system, the existence of two infinite behaviors with the same observations, where exactly one contains the considered fault, violates diagnosability. The existing work concerning discrete event systems (DESs) searches for such ambiguous behaviors, both in centralized and distributed ways [1, 2, 3, 4, 5]. However, in reality, diagnosability turns out to be a quite strong property that generally requires a high number of sensors. Consequently, it is often too expensive to develop a diagnosable system.

To achieve a trade-off between the cost, i.e., a reasonable number of sensors, and the possibility to observe a fault manifestation, we recently introduced a new property called manifestability [6], which is borrowed from philosophy "...which I shall call the "manifestability of the mental", that if two systems are mentally different, then there must be some physical contexts in which this difference will display itself in differential physical consequences" [7]. In the domain of diagnosis, similarly, the manifestability property describes the capability of a system to manifest a fault occurrence in at least one future behavior, which is the weakest property to require. This should be analyzed at design stage on the system model. Under the assumption that no behavior described in the model has zero probability, the fault will then necessarily show itself with nonzero probability after enough runs of the system. Differently, for diagnosability, all future behaviors of all fault occurrences should be distinguishable from all normal behaviors, which is a strong property and sensor demanding. Obviously one has to continue to rely on diagnosability for online safety requirements, i.e., for those faults which may have dramatic consequences if they are not surely detected when they occur, in order to trigger corrective actions. But for all other faults that do not need to be detected at their first occurrence (e.g., whose consequence is a degraded but acceptable functioning that will require maintenance actions in some near future), manifestability checking, which is cheaper in terms of sensors needed, is enough under the probabilistic assumption above.

In this paper, we first present in Section 2 the results of our recent work [8]: we define manifestability for finite automata before providing a sufficient and necessary condition to check it with a formal algorithm based on equivalence checking of languages; then we show that the manifestability problem itself is PSPACE-complete; finally we give experimental results about the efficiency of the algorithm (detailed proofs are omitted and can be found in [8]). In Section 3, we extend this work to real-time systems modeled by timed automata before redefining this property by taking into account time constraints in an explicit way with a sufficient and necessary condition to check it; we prove that the manifestability becomes then undecidable by reducing the undecidable inclusion problem of timed languages to it. Finally, we conclude and draw perspectives in Section 4.

2 Manifestability for DESs

We remind the automaton model for DESs and diagnosability property and introduce manifestability property, showing that it is a weaker property than diagnosability, and we give

a formal sufficient and necessary condition for this property to hold from which we derive an algorithm with PSPACE complexity to check it (we show that manifestability checking is actually PSPACE-complete) and give some experimental results.

2.1 Models of DESs

We model a DES as a finite automaton $G = (Q, \Sigma, \delta, q^0)$, where Q is the finite set of states, Σ the finite set of events, $\delta \subseteq Q \times \Sigma \times Q$ the set of transitions (the same notation will be kept for its natural extension to words of Σ^*), and q^0 the initial state. The set of events Σ is divided into three disjoint parts: $\Sigma = \Sigma_o \uplus \Sigma_u \uplus \Sigma_f$, where Σ_o is the set of observable events, Σ_u the set of unobservable normal events and Σ_f the set of predefined unobservable fault events.

Example 1. *The top part of Figure 1 shows an example of a system model G, where* $\Sigma_o = \{o1, o2, o3\}, \Sigma_u = \{u1, u2\}$, *and* $\Sigma_f = \{F\}$.

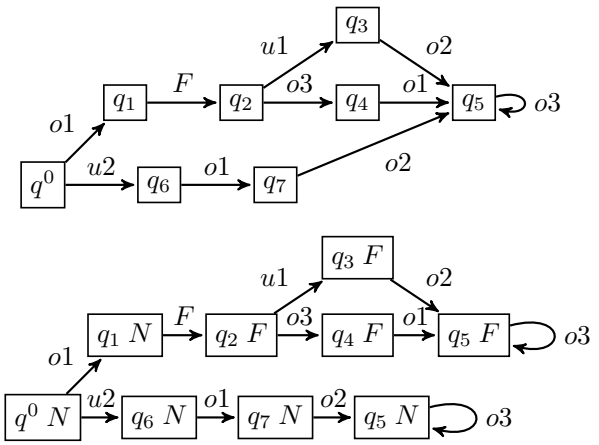

Figure 1: A system model (top) and its diagnoser (bottom).

Similar to diagnosability, the manifestability algorithm that we will propose would have exponential complexity in the number of fault types (i.e., fault labels, or more generally a partition of fault labels) if all those fault types were considered at once. To reduce it to linear complexity in the number of fault types, as in [3, 4], we consider only one fault type at a time. However, multiple occurrences of faults of the given type are allowed. The faults from other types are processed as unobservable normal events. This is justified as the system is manifestable if and only if (iff) it is manifestable for each fault type. In the following, $\Sigma_f = \{F\}$, where F is the currently considered fault type.

Given a system model G, its prefix-closed language $L(G)$, which describes both normal and faulty behaviors of the system, is the set of words produced by G: $L(G) = \{s \in \Sigma^* | \exists q \in Q, (q^0, s, q) \in \delta\}$. Those words containing (resp., not containing) F will be denoted by $L_F(G)$ (resp., $L_N(G)$). In the following, we call a word from $L(G)$ a trajectory in the system G and a sequence $q_0\sigma_0 q_1\sigma_1 ...$, where $q_0 = q^0$ and, for all i, $(q_i, \sigma_i, q_{i+1}) \in \delta$, a path in G whose label $\sigma_0\sigma_1 ...$ is a trajectory in G. Given $s \in L(G)$, we denote the post-language of $L(G)$ after s by $L(G)/s$, formally defined as: $L(G)/s = \{t \in \Sigma^* | s.t \in L(G)\}$. The projection of the trajectory s to observable events of G is denoted by $P(s)$, the observation of s. Two trajectories having same observation are called observably equivalent. This projection can be extended to $L(G)$, i.e., $P(L(G)) = \{P(s) | s \in L(G)\}$, whose elements are called observed trajectories. Traditionally, we do the following assumption about the possibility to always continue a trajectory and to observe infinite trajectories:

Assumption 1: (Alive and observably alive system) G is *alive*, i.e., each state of Q has a successor, so that $L(G)$ is alive (any trajectory has a continuation, i.e., is a strict prefix of another trajectory), and G is *observably alive*, i.e., has no unobservable cycle, i.e., each cycle contains at least one observable event.

We will need some infinite objects. We denote by Σ^ω the set of infinite words on Σ. We define in an obvious way infinite paths in G and thus $L^\omega(G)$ the language of infinite words recognized by G in the sense of Büchi automata [9]. As all states of G are considered as final states, those infinite trajectories are just the labels of infinite paths, and the concept of Büchi automaton coincides with that of Muller automaton, which can be determinized, according to the McNaughton theorem. We can conclude from this that $L^\omega(G)$ is the set of infinite words whose prefixes belong to $L(G)$ and that two equivalent system models, i.e., such that $L(G_1) = L(G_2)$, define the same infinite trajectories, i.e., $L^\omega(G_1) = L^\omega(G_2)$. Particularly, we use $L^\omega_F(G) = L^\omega(G) \cap \Sigma^* F \Sigma^\omega$ for the set of infinite faulty trajectories, and $L^\omega_N(G) = L^\omega(G) \cap (\Sigma \backslash \{F\})^\omega$ for the set of infinite normal trajectories, where $\backslash$ denotes set subtraction (and analogously $L_F(G)$ and $L_N(G)$ for finite trajectories). In the following, we use the classical synchronization operation between two automata G_1 and G_2, denoted by $G_1 \parallel_{\Sigma_s} G_2$, i.e., any event in Σ_s should be synchronized while others can occur whenever possible [10].

The following basic operation is aimed at keeping only information about a given set of events. It boils down to replace by ϵ the events not concerned and eliminate the ϵ-transitions thus created. It will be used to simplify some intermediate structures when checking manifestability without affecting the result obtained.

Definition 1. *(Delay Closure). Given an automaton* $G = (Q, \Sigma, \delta, q^0)$, *its delay closure with respect to* Σ_d, *with* $\Sigma_d \subseteq \Sigma$, *is* $\circlearrowleft_{\Sigma_d}(G) = (Q_d, \Sigma_d, \delta_d, q^0)$, *where: 1)* $Q_d = \{q^0\} \cup \{q \in Q \mid \exists s \in \Sigma^*, \exists \sigma \in \Sigma_d, (q^0, s\sigma, q) \in \delta\}$; *2)* $(q, \sigma, q') \in \delta_d$ *if* $\sigma \in \Sigma_d$ *and* $\exists s \in (\Sigma \backslash \Sigma_d)^*, (q, s\sigma, q') \in \delta$.

2.2 Diagnosability and Manifestability

A fault F is diagnosable in a system model G if it can be detected with certainty when enough events are observed after its occurrence. This property is defined as follows [1], where s^F denotes a trajectory ending with F and $F \in p$, for p a trajectory, means that F appears as a letter of p.

Definition 2. ***(Diagnosability).*** *Given a system model* G *and a fault* F, F *is diagnosable in* G *iff* $\exists k \in \mathbb{N}$ *such that*

$$\forall s^F \in L(G), \forall t \in L(G)/s^F, |t| \geqslant k \Rightarrow$$
$$(\forall p \in L(G), P(p) = P(s^F t) \Rightarrow F \in p).$$

The above definition states that F is diagnosable iff, for each trajectory s^F in G, for each of its extensions t with enough events, then every trajectory p in G that has the same observations as $s^F t$ should contain F. It has been proved that the existence of two indistinguishable infinite trajectories, i.e., holding the same sequence of observable events, with exactly one of them containing the given fault F, is equivalent to the violation of the diagnosability property [2], which is stated as follows.

Definition 3. ***(Critical Pair)****. A pair of infinite (resp., finite) trajectories s, s' is called a critical pair with respect to F, denoted by $s \nprec s'$, if the following conditions are satisfied: 1) $s \in L_F^\omega(G), s' \in L_N^\omega(G)$ (resp., $s \in L_F(G), s' \in L_N(G)$). 2) $P(s) = P(s')$.*

Theorem 1. *A fault F is diagnosable in G iff $\nexists s, s' \in L^\omega(G)$, such that $s \nprec s'$.*

The nonexistence of a critical pair with respect to F witnesses diagnosability of F. To design a diagnosable system, each faulty trajectory should be distinguished from normal trajectories, which is often very expensive in terms of number of sensors required. To reduce such a cost and still make it possible to show the fault after enough runs of the system, another property called manifestability has been recently introduced [6], which is much weaker than diagnosability. Intuitively, manifestability describes whether or not a fault occurrence has the possibility to manifest itself through observations. More precisely, if a fault is not manifestable, then we can never be sure about its occurrence no matter which trajectory is executed after it. Thus, the system model should be necessarily revised.

Definition 4. ***(Manifestability)****. F is manifestable in a system model G iff*

$$\exists s \in L_F(G),\ \forall p \in L(G),\ P(p) = P(s) \Rightarrow F \in p.$$

F is manifestable iff there exists at least one faulty trajectory s in G such that every trajectory p that is observably equivalent to s should contain F. In other words, manifestability is violated iff each occurrence of the fault can never manifest itself in any future. This can be rephrased in terms of diagnosis. Let $Diag$ be the diagnosis procedure with input an observation in Σ_o^* and output a diagnosis in $\{N, F, \{N, F\}\}$. Then, F is manifestable in G iff there exists a trajectory s in G such that $Diag(P(s)) = \{F\}$, i.e., the correct diagnosis of the occurrence of F can be made for at least one faulty trajectory. This emphasizes that manifestability is actually the weakest requirement for the existence of a useful (i.e., not always ambiguous from any observed fauly trajectory) diagnosis procedure.

Theorem 2. *A fault F is manifestable in a system model G iff one or the other of the following equivalent conditions is satisfied:*

$$(\Im)\quad \exists s \in L_F(G),\ \nexists s' \in L_N(G),\ s \nprec s',$$
$$(\Im_\omega)\quad \exists s \in L_F^\omega(G),\ \nexists s' \in L_N^\omega(G),\ s \nprec s'.$$

Manifestability concerns the possibility for the system to manifest at least one occurrence of the fault, i.e., there exists such an occurrence that shows itself in at least one of its futures. In a similar way, one can define a strong version of manifestability, which requires that any occurrence of the fault should show itself in at least one of its futures [8]. It is clear from definitions that diagnosability entails (strong) manifestability.

2.3 Manifestability Verification

Manifestability verification consists in checking whether the condition $\Im_\omega$ (or $\Im$) in Theorem 2 is satisfied for a given system model. In this section, we show how to construct different structures based on a system model to obtain $L_F^\omega(G)$, $L_N^\omega(G)$ as well as the set of critical pairs. The condition $\Im_\omega$ (or $\Im$) can then be checked by using equivalence techniques with these intermediate structures.

System Diagnosers

Given a system model, the first step is to construct a structure showing fault information for each state, i.e., whether the fault has effectively occurred up to this state from the initial state.

Definition 5. *(Diagnoser). Given a system model G, its diagnoser with respect to a considered fault F is the automaton $D_G = (Q_D, \Sigma_D, \delta_D, q_D^0)$, where: 1) $Q_D \subseteq Q \times \{N, F\}$ is the set of states; 2) $\Sigma_D = \Sigma$ is the set of events; 3) $\delta_D \subseteq Q_D \times \Sigma_D \times Q_D$ is the set of transitions; 4) $q_D^0 = (q^0, N)$ is the initial state. The transitions of δ_D are those $((q, \ell), e, (q', \ell'))$, with (q, ℓ) reachable from q_D^0, such that there is a transition $(q, e, q') \in \delta$, and $\ell' = F$ if $\ell = F \vee e = F$, otherwise $\ell' = N$.*

The bottom part of Figure 1 shows the diagnoser for the system depicted in the top part, where each state has its own fault information. More precisely, given a system state q, if the fault has occurred in all paths from q^0 to q, then the fault label for q is F. Such a state is called fault (diagnoser) state. If the fault has not occurred in any path from q^0 to q, then the fault label for q is N and the state is called normal (diagnoser) state. Diagnoser construction keeps the same set of trajectories and splits into two those states reachable by both a faulty and a normal path (q_5 in the example).

Lemma 1. *Given a system model G and its corresponding diagnoser D_G, then we have $L(G) = L(D_G)$ and $L^\omega(G) = L^\omega(D_G)$.*

In order to simplify the automata handled, the idea is to keep only the minimal subparts of D_G containing all faulty (resp., normal) trajectories.

Definition 6. *(Fault (Refined) Diagnoser). Given a diagnoser D_G, its fault diagnoser is the automaton $D_G^F = (Q_{D^F}, \Sigma_{D^F}, \delta_{D^F}, q_{D^F}^0)$, where: 1) $q_{D^F}^0 = q_D^0$; 2) $Q_{D^F} = \{q_D \in Q_D \mid \exists q_D' = (q, F) \in Q_D, \exists s' \in \Sigma_D^*, (q_D, s', q_D') \in \delta_D^*\}$; 3) $\delta_{D^F} = \{(q_D^1, \sigma, q_D^2) \in \delta_D \mid q_D^2 \in Q_{D^F}\}$; 4) $\Sigma_{D^F} = \{\sigma \in \Sigma_D \mid \exists (q_D^1, \sigma, q_D^2) \in \delta_{D^F}\}$. The fault refined diagnoser is obtained by performing the delay closure with respect to the set of observable events Σ_o on the fault diagnoser: $D_G^{FR} = \mathcal{C}_{\Sigma_o}(D_G^F)$.*

The fault diagnoser keeps all fault states as well as all transitions and intermediate normal states on paths from q_D^0 to any fault state. Then we refine this fault diagnoser by only keeping the observable information, which is sufficient to obtain the set of critical pairs. The top (resp., bottom) part of Figure 2 shows the fault diagnoser (resp., fault refined diagnoser) for Example 1.

By construction, the sets of faulty trajectories in D_G^F and in G are equal and this is still true for infinite faulty trajectories. This is also the case for faulty trajectories in D_G^{FR} and observed faulty trajectories in G (finite or infinite). But take care that it may exist infinite normal trajectories in D_G^F (resp., D_G^{FR}) if it exists in G a normal cycle in a path to a fault state (e.g., adding a loop in state q_1 of the system model of Example 1).

Lemma 2. *Given a system model G and its corresponding fault diagnoser D_G^F and fault refined diagnoser D_G^{FR}, we have $L_F(G) = L_F(D_G^F)$, $L_F^\omega(G) = L_F^\omega(D_G^F)$ and $P(L_F(G)) = L_F(D_G^{FR}), P(L_F^\omega(G)) = L_F^\omega(D_G^{FR})$.*

Similarly, we obtain the subpart of D_G containing only normal trajectories.

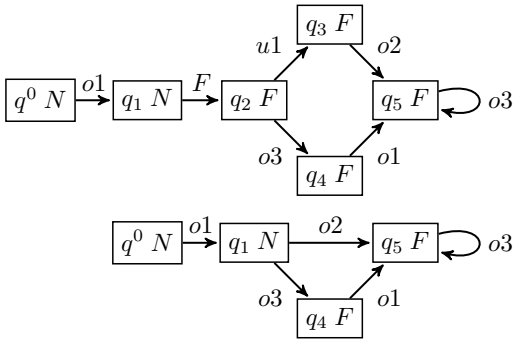

Figure 2: Fault diagnoser (top) and its refined version (bottom) for Example 1.

Definition 7. *(Normal (Refined) Diagnoser). Given a diagnoser D_G, its normal diagnoser is the automaton $D_G^N = (Q_{D^N}, \Sigma_{D^N}, \delta_{D^N}, q_{D^N}^0)$, where: 1) $q_{D^N}^0 = q_D^0$; 2) $Q_{D^N} = \{(q, N) \in Q_D\}$; 3) $\delta_{D^N} = \{(q_D^1, \sigma, q_D^2) \in \delta_D \mid q_D^2 \in Q_{D^N}\}$; 4) $\Sigma_{D^N} = \{\sigma \in \Sigma_D \mid \exists (q_D^1, \sigma, q_D^2) \in \delta_{D^N}\}$. The normal refined diagnoser is obtained by performing the delay closure with respect to Σ_o on the normal diagnoser: $D_G^{NR} = \mathbb{C}_{\Sigma_o}(D_G^N)$.*

Lemma 3. *Given a system model G and its corresponding normal diagnoser D_G^N and normal refined diagnoser D_G^{NR}, we have $L_N(G) = L(D_G^N), L_N^\omega(G) = L^\omega(D_G^N)$ and $P(L_N(G)) = L(D_G^{NR}), P(L_N^\omega(G)) = L^\omega(D_G^{NR})$.*

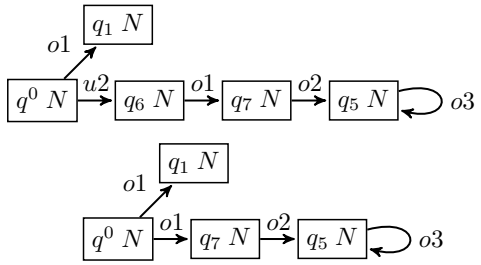

Figure 3: Normal diagnoser (top) and its refined version (bottom) for Example 1.

The top (resp., bottom) part of Figure 3 shows the normal diagnoser (resp., normal refined diagnoser) for Example 1.

Manifestability Checking

In this section, we show how to obtain the set of critical pairs based on the diagnosers described in the precedent section. Based on this, equivalence checking will be used to examine the manifestability condition $\Im_\omega$ (or $\Im$) in Theorem 2.

Definition 8. *(Pair Verifier). Given a system model G, its pair verifier V_G is obtained by synchronizing the corresponding fault and normal refined diagnosers D_G^{FR} and D_G^{NR} based on the set of observable events, i.e., $V_G = D_G^{FR} \parallel_{\Sigma_o} D_G^{NR}$.*

To construct a pair verifier, we impose that the synchronized events are the whole set of observable events. Then V_G is actually the product of D_G^{FR} and D_G^{NR} and the language of the pair verifier is thus the intersection of the language of the fault refined diagnoser and that of the normal refined diagnoser. In the pair verifier, each state is composed of two diagnoser states, whose label (F or N) of the first one indicates whether the fault has effectively occurred in the first of the two corresponding trajectories. If the first of these two states is a fault state, then this verifier state is called ambiguous state since, reaching this state, the first trajectory contains the fault and the second not, while both have the same observations. Trajectories of V_G are thus either normal (all states labels are (N,N)) or ambiguous (all states labels from a certain state are (F,N)), the latter ones being denoted by $L_a(V_G)$ (resp., $L_a^\omega(V_G)$ for infinite ones).

Lemma 4. *Given a system model G with its D_G^{FR}, D_G^{NR} and V_G, we have $L_a(V_G) = L_F(D_G^{FR}) \cap L(D_G^{NR})$ and $L_a^\omega(V_G) = L_F^\omega(D_G^{FR}) \cap L^\omega(D_G^{NR})$.*

In the pair verifier depicted in Figure 4, the gray node represents an ambiguous state.

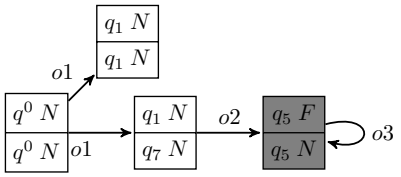

Figure 4: The pair verifier for the system in Example 1.

Using Lemmas 2, 3, 4 and Definition 3, Theorems 1 and 2 rephrase as (the first result being well-known):

Theorem 3. *Given a system model G, a fault F is diagnosable iff $L_a^\omega(V_G) = \varnothing$.*

Theorem 4. *Given a system model G, a fault F is manifestable iff $L_a(V_G) \subset L_F(D_G^{FR})$ or, equivalently, iff $L_a^\omega(V_G) \subset L_F^\omega(D_G^{FR})$, where $\subset$ is the strict inclusion.*

Algorithm

Algorithm 1 is the pseudo-code to verify manifestability, which can simultaneously verify diagnosability.

Algorithm 1 Manifestability and Diagnosability Algorithm for DESs

```
1: INPUT: System model G; the considered fault F
2: D_G ← ConstructDiagnoser(G)
3: D_G^FR ← ConstructFRDiagnoser(D_G)
4: D_G^NR ← ConstructNRDiagnoser(D_G)
5: V_G ← D_G^FR ∥_Σo D_G^NR
6: if L_a^ω(V_G) = ∅ then
7:     return "F is diagnosable and manifestable in G"
8: else if L_a^ω(V_G) = L_F^ω(D_G^FR)
   (or, equivalently, L_a(V_G) = L_F(D_G^FR)) then
9:     return "F is neither diagnosable nor manifestable in
       G"
10: else
11:    return "F is not diagnosable but manifestable in G"
```

Given the input (line 1) as the system model G and the fault F, we first construct the diagnoser (line 2) as described by Definition 5. We then construct fault and normal refined diagnosers (lines 3-4) as defined by Definitions 6 and 7. The next step is to synchronize D_G^{FR} and D_G^{NR} to obtain the pair verifier V_G (line 5). With D_G^{FR} and V_G, we have the following verdicts:

- if $L_a^\omega(V_G) = \varnothing$ (line 6), F is diagnosable and thus manifestable (line 7).

LitSys	|S|/|T|	|S|/|T|(PV)	Time	verdict	HCSys	|S|/|T|	|S|/|T|(PV)	Time	verdict
Ex. 1	8/10	4/4	15	Manifes	h-c1	22/24	18/18	32	Manifes
[4]	16/23	21/23	51	Manifes	h-c2	36/39	74/77	90	Manifes
[3]	16/20	7/9	25	Manifes	h-c3	46/50	105/110	120	Manifes
[11]	3/6	4/6	12	Manifes	h-c4	52/57	160/183	151	Manifes
[5]	18/21	53/57	69	Manifes	h-c5	57/69	32/37	78	Manifes
[12]	9/11	2/1	16	Diagno	h-c6	509/570	79/81	132	Manifes
[1]	12/28	45/51	68	NManifes	h-c7	320/390	1752/1791	323	NManifes

Table 1: Experimental results of manifestability checking for DESs

- if $L_a^\omega(V_G) = L_F^\omega(D_G^{FR})$ or, equivalently, $L_a(V_G) = L_F(D_G^{FR})$ (line 8), necessarily both nonempty, F is not manifestable and thus not diagnosable (line 9).
- if $L_a^\omega(V_G) \neq \varnothing$ and if $L_a^\omega(V_G) \subset L_F^\omega(D_G^{FR})$ or, equivalently, $L_a(V_G) \subset L_F(D_G^{FR})$ (line 10), F is not diagnosable but manifestable (line 11).

Note that $L_F^\omega(D_G^{FR}) = L^\omega(D_G'^{FR})$ (resp., $L_a^\omega(V_G) = L^\omega(V_G')$) where $D_G'^{FR}$ is identical to D_G^{FR} (resp., V_G' identical to V_G), except that the final states, for Büchi acceptance conditions, are limited to fault (resp., ambiguous) states. Note also that the condition $L_a^\omega(V_G) = L_F^\omega(D_G^{FR})$ is equivalent to $L^\omega(V_G) = L^\omega(D_G^{FR})$ as the infinite normal trajectories are identical in V_G and in D_G^{FR} (and idem for finite trajectories).

In Algorithm 1, the complexity of the different diagnosers constructions is polynomial. Building the pair verifier by synchronizing the fault and the normal refined diagnosers is polynomial with the number of system states. To finally check the manifestability, the equivalence checking (line 8) is known to be a PSPACE-complete problem (even for infinite words, see [13]). Thus, the total complexity of this algorithm is poly-space. Algorithm 1 suggests that the manifestability problem is more complex than diagnosability, for which a test of language emptiness is sufficient (line 6), which implies a total NLOGSPACE complexity (in fact it is a result already known that checking diagnosability is NLOGSPACE-complete). Actually, we have shown that the problem of manifestability verification itself is PSPACE-complete by the polynomial reduction to it of rational languages equivalence checking.

Theorem 5. *Given a system model G and a fault F, the problem of checking whether F is manifestable in G is PSPACE-complete.*

2.4 Experimental Results

We have implemented our algorithm (including emptiness and equivalence checking, but existing external solver could be used for this) and applied it on more than one hundred examples taken from literature and hand-crafted ones. The latter ones are constructed to show the scalability since the sizes of the former ones are very small. All our experimental results are obtained by running our program on a Mac OS laptop with a 1.7 GHz Intel Core i7 processor and 8 Go 1600 MHz DDR3 of memory.

Table 1 shows part of our experimental results, where the verdicts (e.g., Manifes(tability), Diagno(sability), N(on) Manifes(tability)) show the strongest property satisfied by the system. We give the number of states and transitions of the system (|S|/|T|), of the pair verifier (|S|/|T|(PV)), as well as the execution time (in milliseconds). The examples (LitSys) include Example 1 with illustrative examples of other papers. We construct the hand-crafted examples (HCSys) by extending the examples (LitSys), focusing on non-diagnosable examples. For example, for a manifestable system, an arbitrary automaton without fault is added such that at least one faulty infinite trajectory can always manifest itself (and obviously critical pairs are preserved).

From our experimental results, the executed time is also dependent on the size of the pair verifier besides that of the system. To achieve a worst case, one way is to employ the construction in the proof of Theorem 5. The hand-crafted example h-c7 is constructed in such a way. We can see that the original HVAC system in [1] (as well as its extension h-c7) is not manifestable. It is thus necessary to go back to design stage to revise the system model. For other manifestable but not diagnosable systems, one interesting future work is to study bounded-manifestability, making sure to detect the fault in bounded time after its occurrence.

3 Manifestability for Real-time Systems

For real-time systems, it is important to take into account during analysis phase explicit time constraints, which are naturally present in real-life systems (e.g., transmission delays, response time, etc...) and thus cannot be neglected considering their impact on some properties, including manifestability. For example, two ambiguous behaviors for an untimed DES may be distinguishable by adding explicit time constraints, e.g., the delay between some two successive observable events is bounded. Considering that classical models (e.g., finite automata, Petri nets) cannot express such real-time constraints, we will analyze manifestability for timed automata (TA), which are one of the most studied models for real-time systems since their introduction by [14]. In such a model, quantitative properties of delays between events can easily be expressed. Executions traces of TA are modeled by timed words, i.e., sequences of events which are attached to the time at which they occur. Hence, TA are seen as acceptors of languages of timed words.

We extend in this section our approach to handle the manifestability problem for TA, demonstrating that it is undecidable for general TA.

3.1 Manifestability for TA

TA constitute a framework for modeling and verifying real-time systems. A TA is essentially a finite automaton, thus with a finite set of states and a finite set of labeled transitions between them, extended with a finite set of real-valued variables modeling $clocks$. During a run of a TA, clock values are initialized with zero when starting in the initial state, and then are increased all with the same speed. Clock values can be compared to constants or between them. These comparisons form guards (resp., invariants of states) that may enable instantaneous transitions (resp., restrict the time during

which one can stay in the corresponding state), constraining thus the possible behaviors of the TA. Furthermore, clocks can be also reset to zero by some of the transitions.

The set of possible clock constraints considered in this paper is formally described by:

$$g ::= true \mid x \bowtie c \mid x - y \bowtie c \mid g \wedge g,$$
where x, y are clock variables, c is a constant and
$$\bowtie \in \{<, \leqslant, =, \geqslant, >\}.$$

Note that a TA allowing such clock constraints is exponentially more concise than its classical variant with only diagonal-free constraints (where the comparison can be done only between a clock value and a constant), but both have same expressiveness. Let X be a finite set of clock variables. A clock valuation over X is a function $v : X \to R$, where R denotes the set $\mathbb{R}_+$ of non-negative real numbers. Then the set of all clock valuations over X is denoted by R^X and the set of time constraints over X by $\mathbb{C}(X)$, where such a constraint is given by a collection of clock constraints. If a clock valuation v satisfies the time constraint g, then it is denoted by $v \models g$. In the following, we denote $[\![g]\!]$ the set of clock valuations that satisfy g, i.e., $[\![g]\!] = \{v \in R^X \mid v \models g\}$.

Definition 9. *(Timed Automaton) A $timed\ automaton$ (TA) is a tuple $A = (Q, \Sigma, X, \delta^X, q^0, I)$, where:*

- *Q is a finite set of states;*
- *Σ is a finite set of events;*
- *X is a finite set of clock variables;*
- *$\delta^X \subseteq Q \times \mathbb{C}(X) \times \Sigma \times 2^X \times Q$ is a finite set of transitions (q, g, σ, r, q'), where the guard $g \in \mathbb{C}(X)$, which has to be satisfied for the transition to be fired, and the clocks $r \subseteq X$ reset to zero, when not specified, are by default $true$ and $\varnothing$, respectively;*
- *$q^0 \in Q$ is the initial state;*
- *$I : Q \to \mathbb{C}(X)$ is the invariant function that associates with each state q the invariant $I(q)$, a constraint that has to be satisfied by clocks in state q ($true$ by default, when not specified). We require $\mathbf{0} \in [\![I(q_0)]\!]$.*

We will again assume the given partition $\Sigma = \Sigma_o \uplus \Sigma_u \uplus \Sigma_f$ and we can without restriction take $\Sigma_f = \{F\}$.

Example 2. *Figure 5 is a TA obtained by adding some time constraints to the system model shown at the top part of Figure 1 and modifying some observable events and the place of the fault. Here c is a clock variable that is used to impose certain periods between events.*

In this example of TA, $(q_3, 0 < c \leqslant 3, o2, \varnothing, q_5) \in \delta^X$ means that only when the guard $0 < c \leqslant 3$ is satisfied, the event $o2$ can occur, inducing an instantaneous state change from q_3 to q_5 with the clock value unchanged. Since the last reset of c before this occurrence of $o2$ happens with the occurrence of $o1$, the period between those occurrences of $o1$ and $o2$ should be greater than 0 and not greater than 3. We denote this transition also as $q_3 \xrightarrow{0<c\leqslant 3;\ o2} q_5$. For the sake of simplicity, we do not assign specific invariants to states, i.e., we use the default value $true$ for all states, which means that there is no time limit for the system to stay in any state (in general, once the invariant ceases to be satisfied, one is obliged to leave the corresponding state).

We call a state with a clock valuation an *extension state*, shortly *state* in the following, i.e., (q, v) with $q \in Q$ and

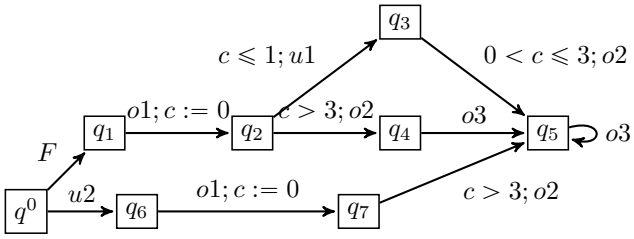

Figure 5: A real-time system model TA.

$v \in R^X$. Let $t \in R$, the valuation $v + t$ is defined by $(v + t)(x) = v(x) + t, \forall x \in X$. Suppose $X' \subseteq X$, we denote by $v[X' \leftarrow 0]$ the valuation such that $\forall x \in X'$, $v[X' \leftarrow 0](x) = 0$ and $\forall x \in X \backslash X'$, $v[X' \leftarrow 0](x) = v(x)$. A TA gives rise to an infinite transition system with two types of transitions between extension states. One is a $time$ transition representing time passage in the same state q, during which the invariant $inv = I(q)$ for q should be always respected. The other one is a $discrete$ transition issued from a labeled transition $q \xrightarrow{g;\ \sigma;\ r} q'$ for TA, associated with an event σ, which is fired (a necessary condition being that the guard g is satisfied) and should be executed instantaneously, i.e., the clock valuation cannot be modified by the transition itself but only by the reset to 0 of those clock variables belonging to r, if any. In the following, both are denoted by $(q, v) \xrightarrow{\nu} (q', v')$, where $\nu \in \Sigma \cup R$. Thus, if $\nu \in \Sigma$, then v should satisfy the guard g in the corresponding TA labeled transition and $v' = v[r \leftarrow 0]$ for r the clock variables reset to 0 in this transition, if any. Otherwise, if $\nu \in R$, then $q' = q$ and $v' = v + \nu$, where all of $v + t$, for $0 \leqslant t \leqslant \nu$, should satisfy the invariant inv associated to q.

Given a TA A, a sequence $(q^0, v_0 = 0) \xrightarrow{\nu_1} (q_1, v_1) \ldots \xrightarrow{\nu_n} (q_n, v_n)$ is a feasible execution in A if $\forall i \in \{0, ..., n-1\}, (q_i, v_i) \xrightarrow{\nu_{i+1}} (q_{i+1}, v_{i+1})$ is either a time or a discrete transition in it. Then the word $\nu_1 ... \nu_n \in (\Sigma \cup R)^\star$ is called a *timed trajectory* or a *run*. This extends to infinite sequences and trajectories. The set of finite (resp., infinite) timed trajectories for A is denoted by $L(A)$ (resp., $L^\omega(A)$, where acceptance is in the sense of Büchi automata or, equivalently, of Muller automata if all states are considered as final). The faulty runs, i.e., containing F, are noted $L_F(A)$ (resp., $L_F^\omega(A)$) and the normal runs, i.e., not containing F, are noted $L_N(A)$ (resp., $L_N^\omega(A)$). By summing up successive time periods and introducing a zero time period between two successive events if any, we can always assume that between any two successive events there is exactly one time period, i.e., periods and events alternate in a timed trajectory. For ρ a timed trajectory, we denote by $time(\rho) \in R \cup \{+\infty\}$ the total time duration for ρ, i.e., $time(\rho) = \sum_{\nu_i \in R \wedge \nu_i \in \rho} \nu_i$ (note that $time(\rho) = +\infty$ implies that ρ is an infinite run). We note $L^\infty(A)$ (resp., $L_F^\infty(A)$, $L_N^\infty(A)$) the time-infinite runs (resp., time-infinite faulty runs, time-infinite normal runs) and thus we have $L^\infty(A) \subseteq L^\omega(A)$ (resp., $L_F^\infty(A) \subseteq L_F^\omega(A)$, $L_N^\infty(A) \subseteq L_N^\omega(A)$). Now we redefine a projection operator P for TA. Given a timed trajectory ρ and a set of events $\Sigma' \subseteq \Sigma$, $P(\rho, \Sigma')$ is the timed trajectory obtained by erasing from ρ all events not in Σ' and summing up the periods between successive events in the resulting sequence. For example, if $\rho = 2\ o1\ 3\ u\ 2\ o2\ 3\ o1$, then $P(\rho, \{o1, o2\}) = 2\ o1\ 5\ o2\ 3\ o1$. In the following, we simply denote $P(\rho)$ the projection of the timed trajectory ρ

to observable events, i.e., $P(\rho) = P(\rho, \Sigma_o)$.

Analogously to DESs, we make for TA the assumption about (time-infinite) continuation of any finite (timed) trajectory and observation of any infinite (timed) trajectory.

Assumption 2: (Time alive and observably alive system) The TA A is *time alive* (also called *timelock-free*) , i.e., from each reachable (by a finite run from q^0) state, starts a time-infinite run (which is equivalent to say that $L(A)$ is exactly made up of all the prefixes of $L^\infty(A)$), and *observably alive*, i.e., there is no infinite run without any observable event, i.e., any infinite run has infinitely many observable events occurrences (this implies in particular that the system cannot stay infinitely, and thus cannot stay an infinitely long time, in a same state with only time transitions).

The TA of Figure 5 is time alive and observably alive.

We will use the following notion, first introduced by [15].

Definition 10. *(Δ-faulty runs) Given A a TA, let $\rho = \nu_1\nu_2\ldots$ be a faulty run. Let then j be the smallest i such that $\nu_i = F$ and let $\rho' = \nu_{j+1}\ldots$. We denote time(ρ') by time(ρ, F) and call it the period from (the first occurrence of) fault F in ρ. If time(ρ, F) $\geqslant \Delta$, where $\Delta \in R$, then we say that at least Δ time units pass after the first occurrence of F in ρ, or, in short, that ρ is Δ-faulty.*

Definition 2 extends to define diagnosability of TA by replacing the length parameter k by the time parameter Δ.

Definition 11. ***(Diagnosability of TA)**. Given a TA A and a fault F, F is diagnosable in A iff $\exists\Delta \in R$ such that*

$$\forall\rho \in L(A), \rho \text{ } \Delta\text{-faulty} \Rightarrow$$
$$(\forall\rho' \in L(A), P(\rho) = P(\rho') \Rightarrow F \in \rho').$$

Note that it is enough to consider only finite runs as, if one Δ is suitable for guaranteeing diagnosability with finite runs, any $\Delta' > \Delta$ is suitable with finite or infinite runs.

Similarly, Definition 3 is transposed to the TA framework.

Definition 12. ***(Timed Critical Pair)**. A pair of infinite (resp., finite) timed trajectories ρ, ρ' is called a timed critical pair with respect to F, denoted by $\rho \not\approx \rho'$, if the following conditions are satisfied: 1) $\rho \in L^\omega_F(A), \rho' \in L^\omega_N(A)$ (resp., $\rho \in L_F(A), \rho' \in L_N(A)$). 2) $P(\rho) = P(\rho')$.*

Finally, the characterization of diagnosability of DESs provided by Theorem 1 extends to TA [15].

Theorem 6. *A fault F is diagnosable in A iff $\nexists\rho, \rho' \in L^\infty(A)$, such that $\rho \not\approx \rho'$.*

From this characterization and from the extension to TA of the construction of a pair verifier V_A, it has been proved that diagnosability of F in A is equivalent to emptiness of $L^\infty_a(V_A)$, a problem known to be PSPACE. And reducing TA reachability to diagnosability proves that checking diagnosability is actually PSPACE-complete for TA [15].

Now we adapt Definition 4 to TA.

Definition 13. ***(Manifestability of TA)**. F is manifestable in a TA A iff*

$$\exists\rho \in L_F(A),$$
$$\forall\rho' \in L(A),\ P(\rho') = P(\rho) \Rightarrow F \in \rho'.$$

Note that we could also adopt a weaker definition of manifestability allowing ρ to be an arbitrary time-finite run, i.e., not only a finite run in $L_F(A)$, but also a run in $L^\omega_F(A)\backslash L^\infty_F(A)$, called Zeno run. But, as Zeno runs are in general non-desirable behaviors due to modeling errors, we adopted this stronger version to exclude manifestabilty through Zeno runs only. An immediate rephrasing of this definition gives, by using Definition 12, the following analog to Theorem 2.

Theorem 7. *A fault F is manifestable in a TA A iff the following condition is satisfied:*

$$(\Im^t) \quad \exists\rho \in L_F(A),\ \nexists\rho' \in L_N(A),\ \rho \not\approx \rho'.$$

Thus, in a similar way as for DESs, the manifestability verification for TA consists in checking the existence of a faulty trajectory that can be distinguishable by observations from all normal ones. The difference is that for TA, the occurrence time of observable events should also be taken into account. In other words, a non-manifestable DES has a chance to become manifestable by adding some time constraints such that at least one faulty trajectory can be distinguishable form normal ones thanks to the different occurrence time of the same observable events. For example, the automaton version (without time constraints) of the system modeled by the TA of Figure 5 is actually not manifestable since all faulty trajectories have the same observations as the normal one, i.e. $o1o2o3^*$. But with time constraints, any faulty trajectory with the event $u1$ is distinguishable from the normal ones since the time duration between the successive observable events $o1$ and $o2$ is at most 3 time units for the former, while greater than 3 time units for the latter.

3.2 Undecidability of Manifestability for TA

From a given TA A modeling a real-time system, the idea is to construct its corresponding fault diagnoser D^F_A (see Definition 6 for the non-refined version) and pair verifier V_A, the latter being constructed by synchronizing D^F_A with normal refined diagnoser D^{NR}_A (see Definition 7) based on the set of observable events (it is not necessary, as we did for automata in order to get more compact representation, to refine D^F_A; the reason to limit as much as possible the use of the refinement process is explained just below). We define the final states in D^F_A as the faulty states and the final states in V_A as the ambiguous states. Thus, manifestability verification consists in checking whether there exists an accepted timed trajectory in D^F_A that is not accepted by V_A. The reason is that each ambiguous timed trajectory in V_A corresponds to a faulty timed trajectory in the original system, for which there exists at least one normal timed trajectory with the same observations, i.e., such that the fault cannot manifest itself. For the example depicted in Figure 5, its trim V_A and D^F_A are shown in Figure 6. Note that in V_A, since we synchronize two timed trajectories, their corresponding clock variable c is distinguished by renaming as c_1 and c_2 [15]. It is obvious that any timed trajectory of D^F_A containing $u1$ (and $o3$) is not accepted by V_A (as the transition in V_A following $u1$ can never be fired due to its clocks constraints), proving thus that F is manifestable.

The problem in the general case is that, to construct D^{NR}_A from D^N_A, we are obliged to rest on the delay closure process, i.e., on removing unobservable events or equivalently removing ϵ-transitions. But it is known that this is not always possible. Actually, it has been proved [16] that, contrary to the case of DES, ϵ-transitions strictly increase the power of TA, if there is a self-loop containing ϵ-transitions which reset some clocks. But ϵ-transitions can be removed if they do not reset clocks, to obtain a TA accepting the same timed language. Thus, we will assume that there is no clock reset for the transitions with non-observable events in the normal diagnoser D^N_A (other non-observable events

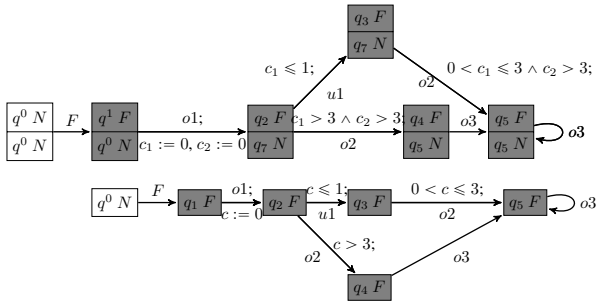

Figure 6: The trim pair verifier V_A (top) and the fault diagnoser D_A^F (bottom) for the system model depicted in Figure 5.

are not handled as ϵ-transitions) and adopt the method proposed in [16] to remove non-observable events in D_A^N to get D_A^{NR}. This assumption is fulfilled by Example 2 (c is not reset in transition $u2$).

Assumption 3: (Limited clock reset TA) In the normal diagnoser D_A^N of A, there is no clock reset for the transitions with non-observable events.

Now, from the construction of those two structures, Theorem 4 extends to TA.

Theorem 8. *Given a real-time system model A with limited clock reset, a fault F is manifestable iff $L_a(V_A) \subset L_F(D_A^F)$, with $V_A = D_A^F \parallel_{\Sigma_o} D_A^{NR}$.*

So we get a way to check manifestability as checking inclusion between languages defined by two TA. But it is well-known that this problem is undecidable for general TA [17]. Actually, we show now how to reduce the inclusion problem of TA to the manifestability problem of TA, which proves the undecidability of manifestability checking for TA.

Theorem 9. *Given a TA A and a fault F, the problem of checking whether F is manifestable in A is undecidable.*

Proof. Reducing the undecidable inclusion problem of TA to the manifestability problem is achieved by adapting to TA the construction in the proof of Theorem 5. Let $A_1 = (Q_1, \Sigma, X_1, \delta_1^{X_1}, q_1^0, I_1)$, $A_2 = (Q_2, \Sigma, X_2, \delta_2^{X_2}, q_2^0, I_2)$ be two arbitrary (non-deterministic) time alive TA on the same vocabulary. One can assume that $Q_1 \cap Q_2 = \varnothing$. Based on A_1 and A_2, one can construct a new TA representing a system model, $A = (Q, \Sigma \cup \{\tau, F\}, X, \delta^X, q^0, I)$, where $Q = Q_1 \cup Q_2 \cup \{q^0\}$, $X = X_1 \cup X_2 \cup \{x^0\}$ $\delta^X = \delta_1^{X_1} \cup \delta_2^{X_2} \cup \{(q^0, x^0 = 0, F, \varnothing, q_1^0), (q^0, x^0 = 0, \tau, \varnothing, q_2^0)\}$ and $I = I_1 \cup I_2$, with $\Sigma_o = \Sigma$, $\Sigma_u = \{\tau\}$ and $\Sigma_f = \{F\}$. A satisfies the assumption of limited clock reset. From the construction of A, one has $L(A_1) = P(L_F(A))$ and $L(A_2) = P(L_N(A))$. In the same way as the proof of Theorem 5, one gets finally $L(A_1) \cap L(A_2) \subset L(A_1) \iff F$ is manifestable in A, i.e., $L(A_1) \subseteq L(A_2) \iff F$ is not manifestable in A. So, languages inclusion testing for TA boils down to manifestability checking of TA. □

4 Conclusion and Future Work

In order to bring an alternative to diagnosability analysis, whose satisfaction is very demanding in terms of sensors placement, we have defined manifestability, a new weaker property and have addressed its formal verification for both DESs and real-time systems modeled as TA. For this, we have constructed different structures from the system model and have demonstrated that manifestability checking boils down to languages inclusion checking and that the manifestability problem is PSPACE-complete for finite automata (for which we have provided preliminary experimental results showing the efficiency and scalability of this approach) and undecidable for TA. We work presently on defining subclasses of TA (based on determinism conditions) for which this problem becomes decidable, actually PSPACE-complete, and can be encoded into an SMT formula, which can be checked automatically by an SMT solver (our TA example of Figure 5 belongs actually to such a decidable subclass).

References

[1] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of Discrete Event System. *Transactions on Automatic Control*, 40(9):1555–1575, 1995.

[2] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A Polynomial Time Algorithm for Testing Diagnosability of Discrete Event Systems. *Transactions on Automatic Control*, 46(8):1318–1321, 2001.

[3] Y Pencolé. Diagnosability Analysis of Distributed Discrete Event Systems. In *Proceedings of the 16th European Conference on Articifial Intelligent (ECAI'04)*, pages 43–47. Nieuwe Hemweg: IOS Press., 2004.

[4] A. Schumann and J. Huang. A Scalable Jointree Algorithm for Diagnosability. In *Proceedings of the 23rd American National Conference on Artificial Intelligence (AAAI'08)*, pages 535–540. Menlo Park, Calif.: AAAI Press., 2008.

[5] L. Ye and P. Dague. Diagnosability Analysis of Discrete Event Systems with Autonomous Components. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI'10)*, pages 105–110. Nieuwe Hemweg: IOS Press., 2010.

[6] L. Ye, P. Dague, D. Longuet, L. Brandán Briones, and A. Madalinski. Fault Manifestability Verification for Discrete Event Systems. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI'16)*, pages 1718–1719. IOS Press., 2016.

[7] D. Papineau. *Philosophical Naturalism*. Blackwell Pub, 1993.

[8] L. Ye, P. Dague, D. Longuet, L. Brandán Briones, and A. Madalinski. How to be sure a faulty system does not always appear healthy? In *Proceedings of 12th International Conference on Verification and Evaluation of Computer and Communication Systems (VECoS 2018), Grenoble, France, September 26-28, 2018*, pages 114–129, 2018.

[9] J.R. Büchi. On a decision method in restricted second order arithmetic. *Z. Math. Logik Grundlag. Math*, 6:66–92, 1960.

[10] C. G. Cassandras and S. Lafortune. *Introduction To Discrete Event Systems, Second Edition*. Springer, 2008.

[11] S. Haar, S. Haddad, T. Melliti, and S. Schwoon. Optimal constructions for active diagnosis. *J. Comput. Syst. Sci.*, 83(1):101–120, 2017.

[12] A. Schumann and Y. Pencolé. Scalable Diagnosability Checking of Event-driven System. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 575–580. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence, Inc., 2007.

[13] A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49(2-3):217–237, 1987.

[14] R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, April 1994.

[15] S. Tripakis. Fault diagnosis for timed automata. In *Proceedings of International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'02)*, Lecture Notes in Computer Science. Springer, 2002.

[16] B. Bérard, P. Gastin, and A. Petit. On the power of non-observable actions in timed automata. In *Proceedings of 13th Annual Symposium on Theoretical Aspects of Computer Science (STACS 96), Grenoble, France, February 22-24, 1996*, pages 257–268, 1996.

[17] R. Alur and P. Madhusudan. Decision problems for timed automata: A survey. In *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures*, pages 1–24, 2004.