How to be sure a faulty system does not always appear healthy?

Philippe Dague, Lulu He, Lina Ye

Noname manuscript No.
(will be inserted by the editor)

How to be Sure a Faulty System Does not Always Appear Healthy?

Fault Manifestability Analysis for Discrete Event and Timed Systems

Philippe Dague · Lulu He · Lina Ye

Received: date / Accepted: date

Abstract Fault diagnosability (allowing one to determine with certainty whether a given fault has effectively occurred based on the available observations) is a crucial and challenging property in complex system automatic control, which generally requires a high number of sensors, increasing the system cost, since it is quite a strong property. In this paper, we analyze a new system property called manifestability, that is a weaker requirement on system observations for having a chance to identify on-line faults: that a faulty system cannot always appear healthy. We propose an algorithm with PSPACE complexity to automatically verify it for finite automata, and 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 verify manifestability of real-time systems modeled by timed automata, proving that it is undecidable in general but under some restricted conditions it becomes PSPACE-complete. Finally we encode this property into an SMT formula, whose satisfiability witnesses manifestability, before presenting experimental results showing the scalability of our approach.

Philippe Dague
LRI, University Paris-Saclay & CNRS
Inria & LSV, ENS Paris-Saclay
E-mail: philippe.dague@lri.fr

Lulu He
LRI, University Paris-Saclay & CNRS
E-mail: lulu.he@lri.fr

Lina Ye
Inria & LSV, ENS Paris-Saclay
CentraleSupélec & LRI, University Paris-Saclay
E-mail: lina.ye@lri.fr

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 expresses 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 [43,31,37]. However, in reality, diagnosability turns out to be quite a 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 [54], 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" [36]. 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. 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. In other words, given a system, if this property holds, this system cannot always appear healthy when

a fault occurs in it, i.e., at least one future behavior observably distinguishes from normal behaviors. In all cases, manifestability is the weakest property to require from the system to have a chance to identify the fault occurrence, i.e., to allow one to establish a diagnostic mechanism. If a fault is not manifestable, then it is useless to try to design a diagnoser for the system or to analyze active diagnosability [27]. 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.

Note that in our precedent work [55], we have defined (strong) manifestability for finite automata before providing a sufficient and necessary condition to check it with a formal algorithm based on equivalence checking of languages of infinite words. Then we have proved that the manifestability problem itself is a PSPACE-complete problem. Furthermore, the correctness and efficiency of the algorithm have also been shown by our experimental results. In this paper, we extend this work to real-time systems modeled by timed automata with several new contributions.

- For finite automata, we show that the manifestability verification can be done equivalently by checking the equivalence of languages of finite words.
- We redefine (strong) manifestability property for timed automata that takes into account time constraints in an explicit way and provide a sufficient and necessary condition to check it.
- Then we prove that the manifestability problem for timed automata is undecidable by reducing the undecidable inclusion problem of timed automata to it. We also study a subclass of timed automata by providing corresponding conditions (in particular related to determinism), under which the manifestability problem becomes decidable.
- For those decidable cases, we propose to encode this problem in an SMT formula, whose satisfiability witnesses manifestability.
- We also provide some preliminary experimental results for this SMT-based algorithm to check manifestability for timed automata, which shows its feasibility and scalability.
- Some more precise comparison with other, already existing notions in the literature are also provided, in particular with opacity, especially secrecy, offering a connection between the work done on safety and the work done on security.

2 Motivating Example

In this section, we explain why it is worth analyzing the manifestability property with a motivating example.

Example 1 Figure 1 shows a modified version of the HVAC system from [43], which is a composite model that captures the interactions between the component models, i.e., a pump, a valve, and a controller. In this system, the initial state is q^0, the events *Valve_open*, *Pump_start*, *Pump_stop*, *Valve_close* are observable and the fault events *Pump_failed*, *Sensor_failed* are not observable, as well as the silent event τ, the latter is used to represent non-deterministic behaviors after the occurrence of the fault event *Sensor_failed*.

In this system, the correct behavior is $(\mathit{Valve\_open}\ \mathit{Pump\_start}\ \mathit{Pump\_stop}\ \mathit{Valve\_close})^\omega$, where ω denotes the infinite concatenation. After the unobservable fault *Pump_failed*, the system exhibits different observations from the correct behavior, this event is thus diagnosable (see Definition 2). Now consider the other fault event *Sensor_failed*, whose occurrence leads to non-deterministic behaviors: one with the same observations as the correct behavior, the other with different observations. Actually temperature sensor fault can cause valve improperly controlled [7]. Here we consider two independent typical behaviors: either entering q_7 by only degrading the efficiency of energy consumption while the system can still assume its basic functionality, or entering q_{11}, where the valve closes immediately without executing the pump, leading to observations that are distinguishable from the correct behavior. Suppose that the issue of energy consumption is not the priority of diagnosis. Hence the fact that the fault *Sensor_failed* can be detected only when it makes enter the state q_{11} is still acceptable in this case. The original (stochastic) diagnosability property is not suitable to handle such situations. If we consider manifestability, the fault *Sensor_failed* is effectively manifestable since its occurrence has at least one future that is distinguishable from the correct behavior.

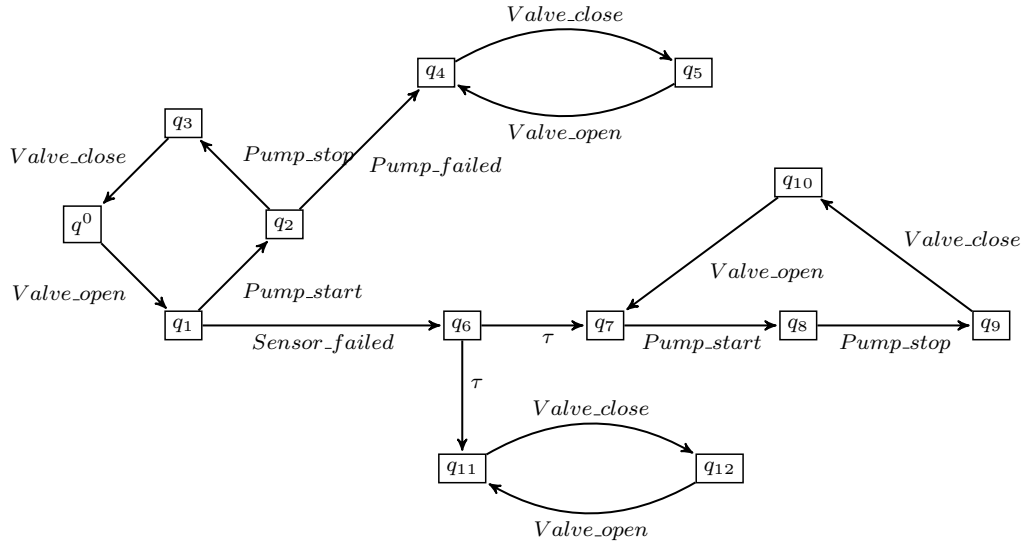

Fig. 1: A simplified HVAC system.

3 Manifestability for DESs

We now present our system model, recall diagnosability, and introduce (strong) manifestability, before giving a formal sufficient and necessary condition for this property to hold. We demonstrate that (strong) manifestability is a weaker property than diagnosability.

3.1 Models of DESs

We model a DES as a finite automaton, denoted by $G = (Q, \Sigma, \delta, q^0)$, where Q is the finite set of states, Σ is the finite set of events, $\delta \subseteq Q \times \Sigma \times Q$ is the set of transitions (the same notation will be kept for its natural extension to words of Σ^*), and q^0 is 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 unobservable fault events.

Example 2 The top part of Figure 2 shows an example of a system model G, where $\Sigma_o = \{o1, o2, o3\}$, $\Sigma_u = \{u1, u2\}$, and $\Sigma_f = \{F\}$. Notice that for diagnosis problem, fault is predefined as an unobservable event in the model. This is different from testing, where faulty behaviors are judged against a specification.

Similar to diagnosability, the manifestability algorithm that we will propose has exponential complexity in the number of fault types (i.e., fault labels). To reduce it to linear complexity, as in [37], 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

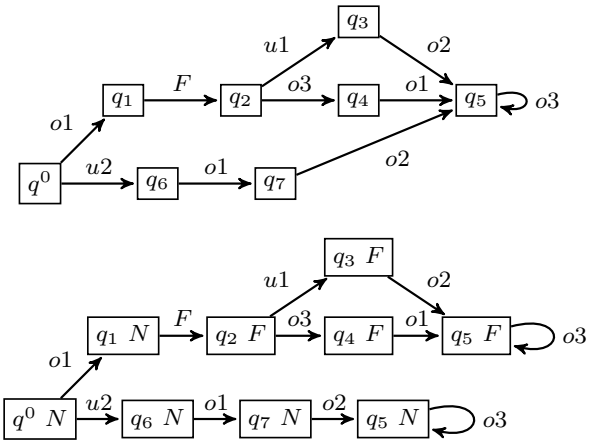

Fig. 2: A system example (top) and its diagnoser (bottom).

is justified as the system is manifestable if and only if (iff) it is manifestable for each fault type. Thus, to check the manifestability of a system with several fault types, one can check its manifestability with respect to each fault type in turn. 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 $q_0\sigma_0q_1\sigma_1...$ a path in G, where $q_0 = q^0$ and, for all i, $(q_i, \sigma_i, q_{i+1}) \in \delta$, 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. 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 [18]. 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. 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. It is easy to generalize the synchronization to a set of automata using its associative property [19]. To verify manifestability, we define the following basic operation, which is to keep only information about a given set of events. It boils down to replacing by ϵ the events not concerned and eliminate the ϵ-transitions (silent transitions) thus created (bisimulation with sequence of silent actions plus one event concerned $\epsilon^* a$). It will be used to simplify some intermediate structures when checking manifestability without affecting the validity of the result obtained.

Definition 1 (Delay Closure). Given a automata $G = (Q, \Sigma, \delta, q^0)$, its delay closure with respect to Σ_d, with $\Sigma_d \subseteq \Sigma$, is $\complement_{\Sigma_d}(G) = (Q_d, \Sigma_d, \delta_d, q^0)$, 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 $\exists s \in (\Sigma \backslash \Sigma_d)^*, (q, s\sigma, q') \in \delta$.

3.2 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 from G after its occurrence. This property is formally defined as follows [43], 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:

1. given $k \in \mathbb{N}$, F is k-diagnosable in G if
$$\forall s^F \in L(G), \forall t \in L(G)/s^F, |t| \geqslant k \Rightarrow (\forall p \in L(G), P(p) = P(s^F t) \Rightarrow F \in p).$$
2. F is diagnosable in G if
$$\exists k \in \mathbb{N} \text{ such that } F \text{ is } k\text{-diagnosable in } G.$$

The above definition states that F is diagnosable if, 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 [31].

Definition 3 (Critical Pair). A pair of trajectories s, s' is called a k-critical pair (resp., an infinite-critical pair or, in short, a critical pair) with respect to F, denoted by $s \nprec_k s'$ (resp., $s \nprec s'$), if the following conditions are satisfied: 1) $s = s^F t \in L_F(G), |t| = k, s' \in L_N(G)$ (resp., $s \in L^\omega_F(G), s' \in L^\omega_N(G)$). 2) $P(s) = P(s')$.

Obviously, the existence of a critical pair implies the existence of a k-critical pair for any k. But, conversely, the existence, for all k, of a k-critical pair implies the existence of a critical pair (by a finitude argument, considering a critical pair as a trajectory in the automaton which is the synchronized product of G by itself on observable events, as it will be used in section 3.3). This leads to the following characterization of diagnosability in terms of critical pairs.

Theorem 1 *A fault F is k-diagnosable (resp., diagnosable) in G iff $\nexists s, s' \in L(G)$, such that $s \nprec_k s'$ (resp., $\nexists s, s' \in L^\omega(G)$, 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 [54], 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 if

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

F is manifestable if 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 faulty 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:*

$$(\mathcal{M}) \quad \exists s^F t \in L_F(G),\ \nexists s' \in L_N(G),\ s^F t \not\prec_{|t|} s',$$
$$(\mathcal{M}_\omega) \quad \exists s \in L_F^\omega(G),\ \nexists s' \in L_N^\omega(G),\ s \not\prec s'.$$

Proof Condition $\mathcal{M}$ is a straightforward rephrasing of Definition 4. We demonstrate condition $\mathcal{M}_\omega$.
$\Rightarrow$ Suppose that F is manifestable in G. Thus from Definition 4, $\exists s \in L_F(G)$ such that $\nexists s' \in L_N(G)$ with $P(s) = P(s')$. By extending s with enough events, which is possible since the language is alive, we obtain then $\exists s \in L_F^\omega(G)$, $\nexists s' \in L_N^\omega(G)$, such that $s \not\prec s'$.
$\Leftarrow$ Suppose now that F is not manifestable in G and show that the condition $\mathcal{M}_\omega$ is consequently not true. From non-manifestability of F and Definition 4, we have $\forall s \in L_F(G)$, $\exists p \in L_N(G), P(p) = P(s)$. This can be formulated as equality of the languages of two automata, as it will be seen in section 3.3 ($L_a(V_G) = L_F(D_G^{FR})$). It results that this equality of the languages still holds for infinite words ($L_a^\omega(V_G) = L_F^\omega(D_G^{FR})$), i.e., $\forall s \in L_F^\omega(G)$, $\exists p \in L_N^\omega(G)$ such that $s \not\prec p$, which is $\neg\mathcal{M}_\omega$, i.e., the condition $\mathcal{M}_\omega$ is not true. ■

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. Now we propose a strong version of manifestability, which requires that all occurrences of the fault should show themselves in at least one of their futures.

Definition 5 (Strong Manifestability). Given a system model G and a fault F:

1. given $k \in \mathbb{N}$, F is strongly k-manifestable in G if
$$\forall s^F \in L(G), \exists t \in L(G)/s^F, |t| \leqslant k,$$
$$\forall p \in L(G),\ P(p) = P(s^F t) \Rightarrow F \in p.$$
2. F is strongly manifestable in G if
$$\forall s^F \in L(G), \exists t \in L(G)/s^F,$$
$$\forall p \in L(G),\ P(p) = P(s^F t) \Rightarrow F \in p.$$

F is strongly manifestable if, for each s^F in G (and not just for only one as in Definition 4) there exists at least one extension t of s^F in G, such that every trajectory p in G that is observably equivalent to $s^F t$ should contain F. In other words, each occurrence of F should show itself in at least one of its futures. In terms of the diagnosis procedure $Diag$, it means that any occurrence s^F of F in G owns a future t such that $Diag(P(s^F t)) = \{F\}$, i.e., the correct diagnosis of any occurrence of F can be made for at least one future trajectory. In a similar way as Theorem 2, we can prove the following theorem, which provides a sufficient and necessary condition for strong manifestability.

Theorem 3 *Given a system model G and a fault F:*

1. *given $k \in \mathbb{N}$, F is strongly k-manifestable in G iff the following condition is satisfied:*
$$(\mathcal{M}_k^s) \quad \forall s^F \in L(G), \exists t \in L(G)/s^F, |t| \leqslant k,$$
$$\nexists s' \in L_N(G),\ s^F t \not\prec_{|t|} s'.$$
2. *F is strongly manifestable in G iff one or the other of the following equivalent conditions is satisfied:*
$$(\mathcal{M}^s) \quad \forall s^F \in L(G), \exists t \in L(G)/s^F,$$
$$\nexists s' \in L_N(G),\ s^F t \not\prec s',$$
$$(\mathcal{M}_\omega^s) \quad \forall s^F \in L(G), \exists t \in L^\omega(G)/s^F,$$
$$\nexists s' \in L_N^\omega(G),\ s^F t \not\prec s'.$$

Proof 1. is just a rephrasing of Definition 5.1 and condition $\mathcal{M}^s$ of 2. a rephrasing of Definition 5.2. It is straightforward, by using aliveness of $L(G)$, that strong manifestability implies $\mathcal{M}_\omega^s$. Consider the reverse. If F is not strongly manifestable, then $\exists s^F \in L(G), \forall t \in L(G)/s^F, \exists s' \in L_N(G), s^F t \not\prec_{|t|} s'$. This means that any faulty trajectory of prefix s^F in G is equal to a trajectory in the synchronized product of G by itself on

observable events (after having erased the unobservable events of the second copy), which can be expressed as languages equality of two automata (see section 3.3), which still holds for infinite words, giving $\neg\mathcal{M}^s_\omega$. ■

Theorem 4 *Given a system model G and a fault F, we have:*

1. *F is k-diagnosable (resp., diagnosable) in G implies that F is strongly k-manifestable (resp., strongly manifestable) in G.*
2. *F is strongly manifestable in G implies that F is manifestable in G.*

Proof 1. Suppose that F is not strongly manifestable, then from Theorem 3, we have $\neg\mathcal{M}^s_\omega$, i.e., $\exists s^F \in L(G), \forall t \in L^\omega(G)/s^F$, $\exists s' \in L^\omega_N(G)$ such that $s^F t \not\asymp s'$. This implies that there does exist at least one critical pair in the system. From Theorem 1, F is not diagnosable (the proof is similar for k).

2. Suppose that F is not manifestable. From Theorem 2, we have $\forall s \in L^\omega_F(G)$, $\exists s' \in L^\omega_N(G)$, such that $s \not\asymp s'$. By choosing arbitrarily one $s^F \in L(G)$ and taking all s of prefix s^F, we obtain $\exists s^F \in L(G), \forall t \in L^\omega(G)/s^F$, $\exists s' \in L^\omega_N(G)$ such that $s^F t \not\asymp s'$, i.e., $\neg\mathcal{M}^s_\omega$. Hence F is not strongly manifestable. ■

3.3 Manifestability Verification

Manifestability verification consists in checking whether the condition $\mathcal{M}_\omega$ (or $\mathcal{M}$) in Theorem 2 is satisfied for a given system model. We now show how to construct different structures based on a system model to obtain $L^\omega_F(G)$, $L^\omega_N(G)$ as well as the set of critical pairs. The condition $\mathcal{M}_\omega$ can then be checked by using equivalence techniques with these intermediate structures. More precisely, if for each infinite faulty trajectory $s \in L^\omega_F(G)$, there exists a corresponding critical pair, then the considered fault is not manifestable. Otherwise, it is manifestable. For the sake of simplicity, we concentrate on how to check manifestability, which can be extended in a straightforward way to handle strong manifestability. This extension will be explained explicitly later.

3.3.1 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 6 (Diagnoser). Given a system model G and a considered fault F, its diagnoser is the automaton $D_G = (Q_D, \Sigma, \delta_D, q^0_D)$, where: 1) $Q_D \subseteq Q \times \{N, F\}$ is the set of states; 2) Σ is the set of events of G; 3) $\delta_D \subseteq Q_D \times \Sigma \times Q_D$ is the set of transitions; 4) $q^0_D = (q^0, N)$ is the initial state. The transitions of δ_D are those $((q,\ell), e, (q',\ell'))$, with (q,ℓ) reachable from q^0_D, 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 2 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 7 (Fault (Refined) Diagnoser). Given a diagnoser D_G, its fault diagnoser is the automaton $D^F_G = (Q_{D^F}, \Sigma_{D^F}, \delta_{D^F}, q^0_D)$, where: 1) $Q_{D^F} = \{q_D \in Q_D \mid \exists q'_D = (q, F) \in Q_D, \exists s' \in \Sigma^*, (q_D, s', q'_D) \in \delta^*_D\}$; 2) $\delta_{D^F} = \{(q^1_D, \sigma, q^2_D) \in \delta_D \mid q^2_D \in Q_{D^F}\}$; 3) $\Sigma_{D^F} = \{\sigma \in \Sigma \mid \exists (q^1_D, \sigma, q^2_D) \in \delta_{D^F}\}$. 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: $D^{FR}_G = \mathbb{C}_{\Sigma_o}(D^F_G)$.

The fault diagnoser keeps all reachable fault states as well as all transitions and intermediate normal states on paths from q^0_D to any fault state. Note that we consider classical permanent fault events, then once the system enter a fault state, it will always stay in a fault state whatever behavior it engages in later. 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 3 shows the fault diagnoser (resp., fault refined diagnoser) for Example 2.

By construction, the sets of faulty trajectories in D^F_G and in G are equal and this is still true for infinite faulty trajectories. This is also the case for faulty trajectories in D^{FR}_G (we call like this labels of paths in D^{FR}_G containing a fault state or whose last state reached owns a transition to a fault state and denote them by $L_F(D^{FR}_G)$) and observed faulty trajectories in G (finite

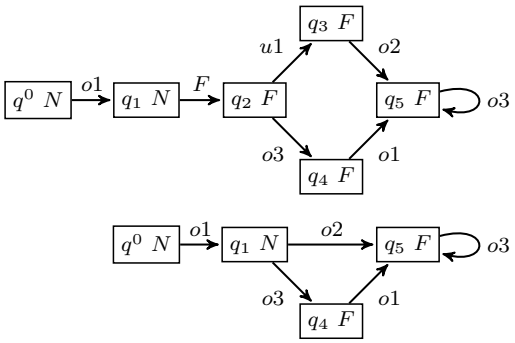

Fig. 3: Fault diagnoser (top) and its refined version (bottom) for Example 2.

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 2).

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.

Definition 8 (Normal (Refined) Diagnoser). Given a diagnoser D_G, its normal diagnoser is the automaton $D_G^N = (Q_{D^N}, \Sigma_{D^N}, \delta_{D^N}, q_D^0)$, where: 1) $Q_{D^N} = \{(q, N) \in Q_D\}$; 2) $\delta_{D^N} = \{(q_D^1, \sigma, q_D^2) \in \delta_D \mid q_D^2 \in Q_{D^N}\}$; 3) $\Sigma_{D^N} = \{\sigma \in \Sigma \mid \exists (q_D^1, \sigma, q_D^2) \in \delta_{D^N}\}$. 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 $P(L_N(G)) = L(D_G^{NR}), P(L_N^\omega(G)) = L^\omega(D_G^{NR})$.*

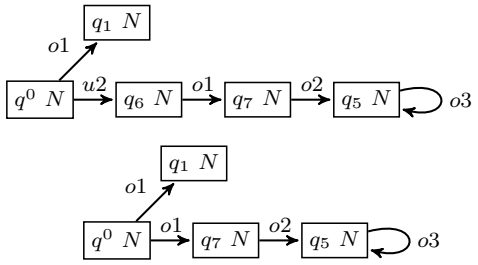

Fig. 4: Normal diagnoser (top) and its refined version (bottom) for Example 2.

The top (resp., bottom) part of Figure 4 shows the normal diagnoser (resp., normal refined diagnoser) for Example 2. Note the presence of the deadlock state (q_1, N), showing that $L_N(G)$ is not necessarily alive.

3.3.2 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 $\mathcal{M}_\omega$ (or $\mathcal{M}$) in Theorem 2.

Definition 9 (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 that are labels of paths in V_G containing an ambiguous state or whose last state reached owns a transition to an ambiguous state are called ambiguous and 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 5, the gray node represents an ambiguous state.

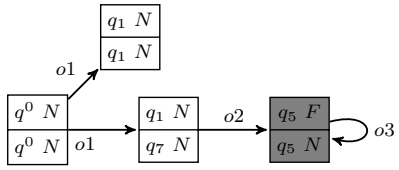

Fig. 5: The pair verifier for the system in Example 2.

Theorem 5 *Given a system model G, a fault F is diagnosable in G iff $L_a^\omega(V_G) = \varnothing$.*

Proof $L_a^\omega(V_G) \neq \varnothing \Leftrightarrow L_F^\omega(D_G^{FR}) \cap L^\omega(D_G^{NR}) \neq \varnothing$ (from Lemma 4) $\Leftrightarrow \quad P(L_F^\omega(G)) \cap P(L_N^\omega(G)) \neq \varnothing$ (from Lemmas 2 and 3) $\Leftrightarrow \exists s \in L_F^\omega(G), \exists s' \in L_N^\omega(G)$ $P(s) = P(s') \Leftrightarrow \exists s, s' \in L^\omega(G)$ $s \not\asymp s'$ (from Definition 3) $\Leftrightarrow F$ is not diagnosable (from Theorem 1). ■

From this Theorem, it follows that the system in Example 2 is not diagnosable as Figure 5 shows that $L_a^\omega(V_G) = \{o_1 o_2 o_3^\omega\}$.

Theorem 6 *Given a system model G, a fault F is manifestable in G iff $L_a(V_G) \subset L_F(D_G^{FR})$ or, equivalently, iff $L_a^\omega(V_G) \subset L_F^\omega(D_G^{FR})$.*

Proof $L_a^\omega(V_G) \not\Subset L_F^\omega(D_G^{FR}) \Leftrightarrow L_F^\omega(D_G^{FR}) \subseteq L^\omega(D_G^{NR})$ (from Lemma 4) $\Leftrightarrow P(L_F^\omega(G)) \subseteq P(L_N^\omega(G))$ (from Lemmas 2 and 3) $\Leftrightarrow \forall s \in L_F^\omega(G), \exists s' \in L_N^\omega(G)$ $P(s) = P(s') \Leftrightarrow \forall s \in L_F^\omega(G), \exists s' \in L_N^\omega(G)$ $s \not\asymp s'$ (from Definition 3) $\Leftrightarrow \neg\mathcal{M}_\omega \Leftrightarrow F$ is not manifestable (from Theorem 2). The proof is identical when using finite trajectories and property $\mathcal{M}$. ■

From this Theorem, it follows that the system in Example 2 is manifestable as Figure 3 and Figure 5 show that $L_F^\omega(D_G^{FR}) \backslash L_a^\omega(V_G) = \{o_1 o_3 o_1 o_3^\omega\}$.

Adapting the proof of Theorem 6 by using Theorem 3 instead of Theorem 2, i.e., by reasoning on property $\mathcal{M}_\omega^s$ (or equivalently $\mathcal{M}^s$) instead of property $\mathcal{M}_\omega$ (or equivalently $\mathcal{M}$), one obtains:

Theorem 7 *Given a system model G, a fault F is strongly manifestable in G iff $\forall s^F \in L(G), L_a(V_G) \cap P(s^F)\Sigma_o^* \subset L_F(D_G^{FR}) \cap P(s^F)\Sigma_o^*$ or, equivalently, $L_a^\omega(V_G) \cap P(s^F)\Sigma_o^\omega \subset L_F^\omega(D_G^{FR}) \cap P(s^F)\Sigma_o^\omega$.*

So, when manifestabilty requires only strict inclusion of the language $L_a(V_G)$ into the language $L_F(D_G^{FR})$, strong manifestability requires that this strict inclusion holds for all corresponding sub-languages made up of the words of both languages having a given prefix equal to the observation of an arbitrary trajectory ending by an occurrence of F. Conversely, to verify non-strong manifestability, it is enough to find one fault trajectory s^F such that there is equality of both sub-languages made up of words of prefix $P(s^F)$: $L_a(V_G) \cap P(s^F)\Sigma_o^* = L_F(D_G^{FR}) \cap P(s^F)\Sigma_o^*$.

3.3.3 Algorithm and Complexity

Algorithm 1 is the pseudo-code to verify manifestability, which can simultaneously verify diagnosability. 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 6. We then construct fault and normal refined diagnosers (lines 3-4) as defined by Definitions 7 and 8. 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 from Theorem 5 and thus manifestable (even strongly manifestable) from Theorem 4 (line 7).
- 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, we can deduce from Theorem 6 that F is not manifestable. Thus, by Theorem 4, F is 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), which can be deduced because of Lemma 4, the former condition means by Theorem 5 that F is not diagnosable, and the latter means by Theorem 6 that F is manifestable (line 11).

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 ||_{Σ_o} 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"
12: end if
```

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 problem (even for infinite words, see [48]). Thus, the total complexity of this algorithm is PSPACE. As we will formally prove just below, Algorithm 1 suggests that the manifestability problem is more complex than diagnosability (for which a test of language emptiness is sufficient, which implies a total NLOGSPACE complex-

ity; actually it is a result already known that checking diagnosability is NLOGSPACE-complete).

Now we show that the problem of manifestability verification itself is a PSPACE-complete problem by the reduction to it of rational languages equivalence checking. The problem of checking non-deterministic automata equivalence is known to be PSPACE-complete.

Theorem 8 *Given a system model G and a fault F, the problem of checking whether F is manifestable in G is PSPACE-complete.*

Proof The complexity of Algorithm 1 is PSPACE. We show that the problem of checking manifestability is PSPACE-hard. Let $G_1 = (Q_1, \Sigma, \delta_1, q_1^0)$ and $G_2 = (Q_2, \Sigma, \delta_2, q_2^0)$ be two arbitrary (non-deterministic) automata on the same vocabulary defining prefix-closed alive languages. One can always assume that $Q_1 \cap Q_2 = \emptyset$. Based on G_1 and G_2, one can construct a new automaton, representing a system model, $G = (Q, \Sigma \cup \{\tau, F\}, \delta, q^0)$, where $Q = Q_1 \cup Q_2 \cup \{q^0\}$ and $\delta = \delta_1 \cup \delta_2 \cup \{(q^0, F, q_1^0), (q^0, \tau, q_2^0)\}$, with $\Sigma_o = \Sigma$, $\Sigma_u = \{\tau\}$ and $\Sigma_f = \{F\}$. From the construction of G, one has $L(G_1) = P(L_F(G))$ and $L(G_2) = P(L_N(G))$. From Lemmas 2, 3 and 4, one obtains $L_a(V_G) = P(L_F(G)) \cap P(L_N(G))$. This implies $L(G_1) \cap L(G_2) = L_a(V_G)$. From Theorem 6, one has $L(G_1) \cap L(G_2) \subset L(G_1) \iff F$ is manifestable in G, i.e., $L(G_1) \subseteq L(G_2) \iff F$ is not manifestable in G. So, rational languages inclusion testing boils down to manifestability checking, which gives the result. Note that we could do exactly the same proof using the languages of infinite words and again Theorem 6, and the fact that the problem of checking non-deterministic automata equivalence on infinite words has also been proved to be PSPACE-complete [48]. And note that the proof shows also that checking strong manifestability is PSPACE-hard. ■

Let now consider verifying strong manifestability. It is obvious from Definition 5 that F is strongly manifestable in G iff each occurrence of F as a transition label is strongly manifestable in G. We can thus assume that there is only one transition in G labeled by F, say (q_F, F, q'_F). From Theorem 7, proving non-strong manifestability of F in G is equivalent to find $s^F \in L(G)$ such that $L_F(D_G^{FR}) \cap P(s^F)\Sigma_o^* \subseteq L_a(V_G) \cap P(s^F)\Sigma_o^*$. In order to simplify the notations, assume in this paragraph that the fault refined diagnoser D_G^{FR} is obtained by the delay closure with respect to $\Sigma_o \cup \{F\}$, i.e., decide to keep the event F in D_G^{FR} and thus in V_G too (this changes nothing to statement of Theorem 7). So, we check the existence of $sF \in L_F(D_G^{FR})$ such that:

$$L_F(D_G^{FR}) \cap sF\Sigma_o^* \subseteq L_a(V_G) \cap sF\Sigma_o^*. \quad \text{(NSM)}$$

Let $\{(q_F, q_i)\}_{i \in I}$ be the set of all states of V_G (trimmed w.r.t. ambiguous states co-accessibility) whose first component is q_F (we omit to write associated fault labels, N – and possibly also F if the F transition is part of a cycle – for q_F and N for the q_i's). Note that q_F appears once among the q_i's. If the property (NSM) is satisfied for some sF, then any extension of sF in $L_F(D_G^{FR})$ has to appear as an extension of sF in $L_a(V_G)$, i.e., $L_{q'_F}(D_G^{FR}) \subseteq \bigcup_{i \in I} L_{(q'_F, q_i)}(V_G)$, where $L_q(G)$ denotes the set of words produced by G from state q, i.e., as if q was the initial state of G. But this is only a necessary condition, not sufficient in general. Actually, if corresponding extensions in V_G need several (q'_F, q_i) as starting states, (NSM) property to be satisfied requires that a common prefix s exists for all of them, i.e., a common word in the associated languages $L(V_G, (q_F, q_i))$, where $L(G, q)$ denotes the set of words produced from paths from initial state to q in G, i.e., as if q was the only final state (s will then necessarily be a prefix in $L(D_G^{FR}, q_F)$ too). Finally, the existence of sF verifying (NSM) is equivalent to the existence of $J \subseteq I$, such that: $L_{q'_F}(D_G^{FR}) \subseteq \bigcup_{i \in J} L_{(q'_F, q_i)}(V_G)$ and $\bigcap_{i \in J} L(V_G, (q_F, q_i)) \neq \emptyset$. This equivalence provides an algorithm for checking non-strong manifestability, which boils down to a finite number of tests of language equivalence and language emptiness. In the worst case, this algorithm may require testing all subsets J of I, thus giving an EXPTIME complexity in the size of G. Nevertheless, under the particular assumption that there is no cycle in G before the occurrence of F or containing F, the system has then only a finite number of fault occurrences, i.e., of possible prefixes sF, as the language $L(D_G^{FR}, q_F)$ is finite. Processing each word sF of this language separately, one has just to do each time one language equivalence test between the fault refined diagnoser and the pair verifier limited to sF, which gives a PSPACE complexity for the corresponding algorithm. This proves that checking strong manifestability is a PSPACE-complete problem for the class of systems verifying this assumption.

3.4 Experimental Results

We have applied our algorithm 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[1], including those in Section 4.4, are obtained by running our program on a

[1] the examples in Table 1 and Table 2 can be found in `https://www.lri.fr/~linaye/cases.pdf`

LitSys	|S|/|T|	|S|/|T|(PV)	Time	verdict	HCSys	|S|/|T|	|S|/|T|(PV)	Time	verdict
Ex. 2	8/10	4/4	15	SManifes	h-c1	22/24	18/18	32	SManifes
ls_1[44]	16/23	7/11	39	Manifes	h-c2	36/39	74/77	90	Manifes
ls_2[37]	16/20	7/9	25	Manifes	h-c3	87/90	63/68	105	Manifes
ls_3[27]	4/7	3/3	12	SManifes	h-c4	52/57	32/30	63	SManifes
ls_4[53]	15/21	11/16	52	SManifes	h-c5	57/69	32/37	78	SManifes
ls_5[45]	11/15	2/1	16	Diagno	h-c6	509/570	79/81	132	Manifes
ls_6[43]	8/12	8/11	53	NManifes	h-c7	986/1032	870/861	312	NManifes

Table 1: Experimental results of manifestability checking for DESs

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), S(trong)Manifes, Diagno(sability), N(on)Manifes) show the strongest property satisfied by the system. For example, if it is Manifes, then it is not SManifes nor Diagno. Diagno implies both SManifes and Manifes. 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 (millisecond is used as time unit). The size of the pair verifier includes all transitions generated from the synchronization of the fault refined diagnoser and the normal refined diagnoser. The examples shown here include Example 2 in this paper with the (modified) illustrative examples of other papers that handle similar problems.

To construct the hand-crafted examples (HCSys) from those selected from the literature (LitSys), we are not interested in diagnosable examples. First, diagnosable systems are rare in the literature as well as in the industry. Second, diagnosability implies an empty language of ambiguous infinite words for the pair verifier, which can be verified without equivalence checking. The efficiency cannot be convincing by applying our algorithm on diagnosable examples. When extending the examples from the literature, we keep the same verdict. For example, for a manifestable system, an arbitrary automaton without fault is added in a place such that at least one faulty 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 example construction in the proof of Theorem 8 by setting $L(G_1) = L(G_2)$. The hand-crafted example h-c7 is constructed in such a way.

We can see that the original HVAC system in [43] is not manifestable, i.e., any faulty behavior cannot be diagnosed in all its futures. 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.

4 Manifestability for Real-time Systems

Note that 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 always different. 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 [2]. 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 and becomes PSPACE-complete under some special conditions. For the decidable cases, we propose a new approach to efficiently encode the manifestability problem into Satisfiability Modulo Theories (SMT). SMT is an extended form of Boolean satisfiability (SAT), where literals are interpreted with respect to a background theory. In terms of expressiveness, one can provide with SMT a natural symbolic representation for TA. The discrete parts of TA can be represented by the Boolean part and the continuous clocks evolutions can be expressed by the linear real arithmetic theory.

4.1 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), and by doing so constrain the possible behaviors of the TA. Furthermore, clocks can be also reset to zero by some of the transitions.

Before introducing the formal definition of TA, we first give the set of possible clock constraints considered in this paper, 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 (actually, for implementation, the set $\mathbb{Q}_+$ of non-negative rational numbers is used to have an exact computer representation). 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 10 (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, whose value by default, when not specified, is *true*. 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 3 Figure 6 is a TA obtained by adding some time constraints to the system model shown at the top part of Figure 2 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.

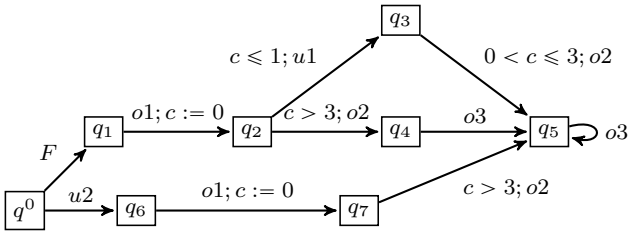

Fig. 6: A real-time system model TA.

In this example of TA, $(q_3, 0 < c <= 3, o2, \varnothing, q_5) \in \delta^X$ means that only when the guard $0 < c <= 3$ 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 smaller than or equal to 3. We denote this transition also as $q_3 \xrightarrow{0<c<=3;\ o2} q_5$. 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. When there is an invariant for a state, 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 $v \in R^X$. 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 the state q.

Given a TA A, a sequence of such transitions $(q^0, v_0 = 0) \xrightarrow{\nu_1} (q_1, v_1) \dots \xrightarrow{\nu_n} (q_n, v_n)$ 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., $P(\rho) = P(\rho, \Sigma_o)$.

We make for TA the analog assumption done for DES about the (time-infinite) continuation of any (timed) trajectory and the necessity to observe 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 6 is time alive and observably alive.

We will use the following notion, originally introduced by [50].

Definition 11 (Δ-faulty runs) Given A a TA, let $\rho = \nu_1 \nu_2 \dots$ be a faulty run. Let then j be the smallest i such that $\nu_i = F$ and let $\rho' = \nu_{j+1} \dots$. We denote $\text{time}(\rho')$ by $time(\rho, F)$ and call it the *period from* (the first occurrence of) *fault F in ρ*. If $\text{time}(\rho, F) \geqslant \Delta$, 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 12 (Diagnosability of TA). Given a TA A and a fault F:

1. given $\Delta \in R$, F is Δ-diagnosable in A if
$$\forall \rho \in L(A), \rho \text{ } \Delta\text{-faulty} \Rightarrow (\forall \rho' \in L(A), P(\rho) = P(\rho') \Rightarrow F \in \rho').$$
2. F is diagnosable in A if
$$\exists \Delta \in R \text{ such that } F \text{ is } \Delta\text{-diagnosable in } A.$$

Note that we used in this definition the language of finite words $L(A)$. This is because Δ-diagnosability with this definition implies Δ'-diagnosability, for any $\Delta' > \Delta$, with the definition allowing both finite and infinite words, as any infinite Δ'-faulty run owns a prefix which is a finite Δ-faulty run. Obviously, in absence of Zeno runs (infinite runs in finite time), both definitions are exactly the same.
In the same way, Definition 3 is transposed to the TA framework.

Definition 13 (Timed Critical Pair). A pair of timed trajectories ρ, ρ' is called a timed Δ-critical pair (resp., a timed infinite-critical pair or, in short, a timed critical pair) with respect to F, denoted by $\rho \not\approx_\Delta \rho'$ (resp., $\rho \not\approx \rho'$), if the following conditions are satisfied:

- $\rho \in L_F(A)$, $\text{time}(\rho, F) = \Delta$, $\rho' \in L_N(A)$ (resp., $\rho \in L_F^\infty(A), \rho' \in L_N^\infty(A)$).
- $P(\rho) = P(\rho')$.

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

Theorem 9 *A fault F is Δ-diagnosable (resp., diagnosable) in A iff $\nexists \rho, \rho' \in L(A)$, such that $\rho \not\approx_\Delta \rho'$ (resp., $\nexists \rho, \rho' \in L^\infty(A)$, 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 $L_a^\infty(V_A)$, a problem known to be PSPACE. And reducing TA reachability to diagnosability proves that checking diagnosability is actually PSPACE-complete for TA [50].

Now we adapt Definition 4 to define manifestability of TA.

Definition 14 (Manifestability of TA). F is manifestable in a TA A if

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

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 Zeno run in $L_F^\omega(A) \backslash L_F^\infty(A)$ but as Zeno runs are in general non-desirable behaviors due to modeling errors, we adopted this stronger version excluding manifestability through Zeno runs only. An immediate rephrasing of this definition gives, by using Definition 13, the following result (analog to Theorem 2).

Theorem 10 *A fault F is manifestable in a TA A iff the following condition is satisfied:*

$$(\mathcal{M}^t) \quad \exists \rho \in L_F(A),\ \nexists \rho' \in L_N(A),\ \rho \not\approx_{time(\rho,F)} \rho'.$$

In the same way, Definition 5 can be adapted to define strong manifestability of TA.

Definition 15 (Strong Manifestability of TA). Given a TA A and a fault F:

1. given $\Delta \in R$, F is strongly Δ-manifestable in A if

$$\forall \rho^F \in L(A), \exists t \in L(A)/\rho^F, time(t) \leqslant \Delta,$$
$$\forall \rho' \in L(A),\ P(\rho') = P(\rho^F t) \Rightarrow F \in \rho'.$$

2. F is strongly manifestable in A if

$$\forall \rho^F \in L(A), \exists t \in L(A)/\rho^F,$$
$$\forall \rho' \in L(A),\ P(\rho') = P(\rho^F t) \Rightarrow F \in \rho'.$$

And, similar to Theorem 3.1, we obtain the following straightforward result.

Theorem 11 *Given a TA A, a fault F and $\Delta \in R$, F is strongly Δ-manifestable in A iff the following condition is satisfied:*

$$(\mathcal{M}_\Delta^{ts}) \quad \forall \rho^F \in L(A), \exists t \in L(A)/\rho^F, time(t) \leqslant \Delta,$$
$$\nexists \rho' \in L_N(A),\ \rho^F t \not\approx_{time(t)} \rho'.$$

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 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 from normal ones thanks to the different occurrence time of the same observable events. For example, consider the system modeled by the TA of Figure 6. Its simple automaton version without time constraints is actually not manifestable since all faulty trajectories have the same observations as the normal one, i.e. $o1o2o3^*$. After adding time constraints, at least one faulty timed trajectory can manifest itself, distinguishable from the normal one. More precisely, the faulty trajectory with the event $u1$ can be distinguishable from the normal one since the time duration between the successive observable events $o1$ and $o2$ is smaller than or equal to 3 time units for the former, while that is strictly greater than 3 time units for the latter. When the time between observing $o1$ and $o2$ is not greater than 3, we can be sure about the occurrence of the fault. One can clearly see that adding time constraints sometimes makes a non-manifestable system manifestable by distinguishing temporally a faulty trajectory, which cannot manifest itself in the untimed setting, from all normal trajectories.

4.2 Undecidability and Decidability Results

From a given TA A modeling a real-time system, the idea is to construct its corresponding fault diagnoser D_A^F (see Definition 7 for the non-refined version) and fault pair verifier V_A^F, the latter being constructed by synchronizing D_A^F with normal refined diagnoser D_A^{NR} (see Definition 8) based on the set of observable events (it is not necessary, as we did for automata in order to get more compact representation, to do the refinement of D_A^F; the reason for limiting as much as possible the use of the refinement process is explained just below). We define the final states in D_A^F as the faulty states and the final states in V_A^F as the ambiguous states. Thus, manifestability verification consists in checking whether there does exist an accepted timed trajectory in D_A^F that is not accepted by V_A^F. The reason is that each ambiguous timed trajectory in V_A^F corresponds to a faulty timed trajectory in the original system, for which there exists at least one normal timed trajectory with the

same observation, i.e., such that the fault cannot manifest itself. For the example depicted in Figure 6, its trim V_A^F and D_A^F are shown in Figure 7. Note that in V_A^F, since we synchronize two timed trajectories, their

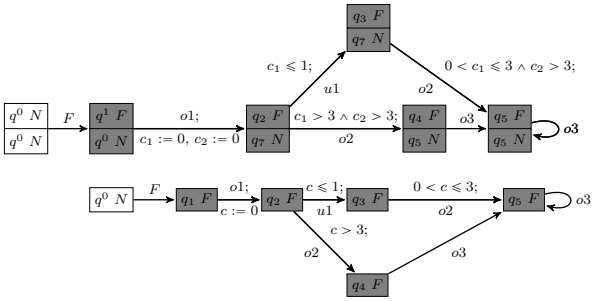

Fig. 7: The fault pair verifier V_A^F for the system whose model is depicted in Figure 6 (top); the fault diagnoser D_A^F (bottom).

corresponding clock variable c is distinguished by renaming as c_1 and c_2 [50]. It is obvious that any timed trajectory of D_A^F containing $u1$ (and $o3$) is not accepted by V_A^F (as the transition in V_A^F 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_A^{NR} from D_A^N, 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 [8] that, contrary to the case of FSM, 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. More generally, it has been proved [22] that a TA such that no ϵ-transition with nonempty reset set lies on any directed cycle can be effectively transformed into a TA without ϵ-transitions that accepts at least the timed language of the initial TA and whose non-Zeno accepted timed words are the same with those accepted by the initial TA. In this paper, as we do not exclude Zeno runs, we will assume simply that there is no clock reset for the transitions with unobservable events in the normal diagnoser D_A^N. Other unobservable events are not handled as ϵ-transitions. This assumption is fulfilled by Example 3, depicted in Figure 6, as c is not reset in transition u_2 (but it could be reset in transition u_1). So in our case, we adopt the method proposed in [8] to remove unobservable events in D_A^N to get D_A^{NR}.

Assumption 3: (Limited clock reset TA) Any transition in A from a state q reachable from q^0 by a normal (i.e., not containing F) execution, and associated to an unobservable normal event σ, is of the form $(q, g, \sigma, \varnothing, q')$, i.e., has no clock reset.

This is equivalent to say that there is no clock reset for transitions with unobservable events in D_A^N. Now, from the construction of the two structures D_A^F and D_A^{NR}, Theorem 6 extends to TA.

Theorem 12 *Given a real-time system model A with limited clock reset, a fault F is manifestable in A iff $L_a(V_A^F) \subset L_F(D_A^F)$, with $V_A^F = D_A^F \parallel_{\Sigma_o} D_A^{NR}$.*

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 [3].
Actually, in a similar way to that in the discrete framework, we show how to reduce the inclusion problem of TA to the manifestability problem of TA, which proves the undecidability of manifestability checking for TA.

Theorem 13 *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 8. 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 defining prefix-closed timed languages. 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 8, 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. The proof shows also that checking strong manifestability is undecidable. ■

Since the manifestability problem of TA is undecidable, we now analyze a subclass of TA whose manifestability problem is decidable. The idea comes from the fact that the inclusion problem of deterministic TA is PSPACE-complete [3]. For a TA A, we say it is deterministic whenever given two distinct discrete transitions from the same state with the same label $(q, g_1, \sigma, r_1, q_1')$ and $(q, g_2, \sigma, r_2, q_2')$, it holds that $g1 \wedge g_2$ is not satisfiable, i.e., there is no common time where one or the other could be indifferently fired.

Definition 16 (Single-Silent Deterministic TA)
Given a TA A with limited clock reset, we call it Single-Silent Deterministic TA (SS-DTA), if A is deterministic and, from any state q reachable from q^0 by a normal execution, if a transition by an unobservable normal event exists from q, then it is the only one normal transition from q, i.e., if $(q, g_1, \sigma_1, r_1, q'_1)$ and $(q, g_2, \sigma_2, r_2, q'_2)$ are two different transitions with $\sigma_1, \sigma_2 \neq F$, then $\sigma_1 \in \Sigma_o$ and $\sigma_2 \in \Sigma_o$.

The TA of Figure 6 is an SS-DTA. One can notice that for an SS-DTA, as A is deterministic, so are both its normal diagnoser D_A^N and its fault diagnoser D_A^F. The condition of Definition 16 also implies that any unobservable normal transition in D_A^N is the only one transition issued from its source state. And with Assumption 3, it is easy to show that D_A^{NR}, constructed from D_A^N by deleting unobservable events, keeps deterministic. Since both D_A^{NR} and D_A^F are deterministic, then V_A^F obtained from their synchronization is also deterministic (this can be verified for the example in Figure 7). Thus in a similar way as that for Theorem 8, the following theorem can be proved by reducing the inclusion problem of two deterministic TA to manifestability problem of an SS-DTA (note that Assumption 3 and the second condition of Definition 16 are trivially satisfied as τ is the unique unobservable normal event).

Theorem 14 *Given an SS-DTA and a fault F, its manifestability problem is PSPACE-complete.*

4.3 Encoding Bounded Manifestability

In this section, we show how to verify the manifestability of an SS-DTA by encoding it into SMT formula. We do not consider the problem of deciding if an arbitrary TA can be transformed into an SS-DTA, as it is already known that the problem of deciding whether a TA is determinizable is actually undecidable [24]. Note that there are some subclasses of TA that are determinizable by using for example algorithm proposed in [6]. To facilitate SMT encoding for the inclusion checking, we add an additional non-final state *sink* to the fault pair verifier V_A^F, which is deterministic, such that it is deterministic and complete. This is done by adding transitions from other states to the state *sink* and a self-loop for state *sink* (see [2] for details, in the case where invariants are $True$). By Theorem 12, checking that an SS-DTA A satisfies manifestability is equivalent to find a faulty timed trajectory ρ (i.e., accepted by D_A^F), such that the timed trajectory of V_A^F identical to ρ (which exists as V_A^F is complete and is unique as V_A^F is deterministic) is not accepted by V_A^F. Thus manifestability checking boils down to finding a timed trajectory which is accepted by D_A^F and rejected by V_A^F. To code this problem as a (finite) logical formula whose satisfiability will be determined by bounded model checking, it is necessary to bound the length of the timed trajectories considered. An input integer parameter k will thus be given, and only timed trajectories ρ such that $|\rho| \leqslant k$ will be considered. Now, what will matter for the end-user, in case of manifestability, is that the fault will manifest itself after an acceptably long time. That is, his requirement will not be a length k, but a time delay $\Delta \in R$ representing a time upper-bound after its occurrence for the fault to manifest itself (similar to the concept of Δ-manifestability used in Definition 15.1). As there is in general no relationship between k and Δ among timed trajectories (except that, for a given timed trajectory, the two parameters vary in the same sense), as a longer timed trajectory may have a smaller time, the usage of the method is as follows: one checks if it exists a timed trajectory of length at most k (input of the algorithm) and of time after fault at most Δ (input from the end-user), then the requirements of the end-user are fulfilled; otherwise one repeats the process with a greater k. One can at any step query without the parameter Δ in order to get back a delay Δ' (greater than Δ), proving the manifestability in time Δ' after the fault, and see if it could be acceptable by the end-user. Obviously, if no theoretical length upper-bound exists for manifestability, the absence of solution will not be a guarantee that the fault is not manifestable.

4.3.1 Encoding (deterministic) TA

We now show how to logically encode in SMT the existence of a timed trajectory (of length at most k and possibly of time after fault at most Δ) accepted by D_A^F and rejected by V_A^F such that the satisfiability of the logical formula is equivalent to the existence of such a trajectory. In case of satisfiability, a model is returned, which actually provides such a timed trajectory. As explained before, we can assume that time and discrete transitions alternate in any timed trajectory. Hence, we rewrite $(q, v) \xrightarrow{t} (q, v'') \xrightarrow{\sigma} (q', v')$, where $t \in R$ and $\sigma \in \Sigma$, as $(q, v) \xrightarrow{t,\sigma} (q', v')$. In the following, we consider this kind of combined time-discrete transition during the encoding. Accordingly, a timed trajectory of length k is a finite sequence $(t_0, \sigma_0), (t_1, \sigma_1), ..., (t_{k-1}, \sigma_{k-1})$, where $t_i \in R$, $\sigma_i \in \Sigma$, and $\forall i$, $0 \leqslant i \leqslant k-1$, $(q_i, v_i) \xrightarrow{t_i,\sigma_i} (q_{i+1}, v_{i+1})$ is allowed by A. We can assume that the timed trajectory ends by a time transition, that we will represent by setting $\sigma_{k-1} = \epsilon$ as a silent event. For the example depicted in Figure 6, one 4-length timed trajectory is $\rho = (1.5, u2), (3, o1), (5, o2), (1, \epsilon)$ that is

witnessed by the feasible execution $(q_0, c = 0) \xrightarrow{1.5,u2} (q_6, c = 1.5) \xrightarrow{3,o1} (q_7, c = 0) \xrightarrow{5,o2} (q_5, c = 5) \xrightarrow{1,\epsilon} (q_5, c = 6)$.

Given a TA A and a given fault F, to check its manifestability, we first construct from A the fault pair verifier V_A^F and the fault diagnoser D_A^F as described before. We denote them $V_A^F = (\hat{Q}, \Sigma, \hat{X}, \hat{\delta}^X, \hat{q}_0, \hat{I})$ and $D_A^F = (Q, \Sigma, X, \delta^X, q_0, I)$, both with Σ, the set of events of A. Then we encode essential static parts in V_A^F and D_A^F as follows.

- The set of states is encoded by positive integers with the function $E_Q : Q \to Q^E = \{1, ..., \|Q\|\}$ (resp., $\hat{E_Q} : \hat{Q} \to \hat{Q}^E = \{1, ..., \|\hat{Q}\|\}$, where $Q^F \subseteq Q^E$ (resp. $\hat{Q}^F \subseteq \hat{Q}^E$) codes the final states, i.e., Q^F corresponds to the set of faulty states (resp., $\hat{Q}^F$ to the set of ambiguous states).
- The set of events for both TA is encoded by positive integers $E_\Sigma : \Sigma \to \Sigma^E = \{1, ..., \|\Sigma\|\}$. where $\Sigma^E = \Sigma_o^E \uplus \Sigma_u^E \uplus \Sigma_f^E$, corresponding to $\Sigma = \Sigma_o \uplus \Sigma_u \uplus \Sigma_f$. The normal events $\Sigma_n = \Sigma_o \uplus \Sigma_u$ are encoded by integers from 1 to $\|\Sigma_n\|$ and fault events by integers from $\|\Sigma_n\| + 1$ to $\|\Sigma\|$.
- The set of symbolic transitions is encoded by a set of tuples $E_{\delta^X} : \delta^X \to \delta^E = (Q^E \times \mathbb{C}(X) \times \Sigma^E \times 2^X \times Q^E)$ with $E_{\delta^X}(q, g, \sigma, r, q') = (E_Q(q), g, E_\Sigma(\sigma), r, E_Q(q'))$. A similar way to define $\hat{E}_{\delta^X}$ on $\hat{\delta}^X$ for $\hat{\delta}^E$.

4.3.2 Encoding timed trajectories

Given k and Δ, the essential point is to define a formula Ψ_Δ^k whose satisfiability is equivalent to the existence of a timed trajectory ρ with $|\rho| = k$ and $\text{time}(\rho, F) \leqslant \Delta$ which is accepted by D_A^F and rejected by V_A^F. Such ρ is actually a witness of manifestability (in time after the fault at most Δ). Before presenting this formula, we need to distinguish the value of variables representing the timed trajectory in D_A^F and in V_A^F. To do this, the variables equipped with a hat are associated to V_A^F while the variables without a hat are attached to D_A^F, except for variables representing the events and the time periods which are the same.

- The integer-valued variables $e_0, ..., e_{k-1}$ encode the events of the timed trajectory both in D_A^F and V_A^F (with e_{k-1} encoding ϵ).
- The integer-valued variables $s_0, ..., s_k$ (resp., $\hat{s}_0, ..., \hat{s}_{\hat{k}}$) represent the states of the timed trajectory in D_A^F (resp., V_A^F).
- The real-valued variables $t_0, ..., t_{k-1}$ encode the time periods for the timed trajectory in both TA.
- The real-valued variables $v_0^x, ..., v_k^x$, for all $x \in X$ (resp., $\hat{v}_0^x, ..., \hat{v}_{\hat{k}}^x$, for all $x \in \hat{X}$) represent the values of the corresponding clock x in each state in D_A^F (resp., V_A^F), initialized as 0, i.e., $v_0^x = \hat{v}_0^x = 0$.
- The additional real-valued variables $v_0^F, ..., v_k^F$ represent the time elapsed after the first fault occurrence in D_A^F (-1 by convention before the fault occurrence).

4.3.3 Encoding bounded manifestability

In order to describe the formula Ψ_Δ^k as intuitively as possible, we present it with different separate parts.

- Initialization. The two timed trajectories should start in the initial state with the initialization of all clock variables.
  - For the timed trajectory in D_A^F:
$$\Phi^{Init} := (\bigwedge_{x \in X} v_0^x = 0) \wedge (s_0 = E_Q(q_0)) \wedge (v_0^F = -1).$$
  - For the timed trajectory in V_A^F:
$$\hat{\Phi}^{Init} := (\bigwedge_{x \in \hat{X}} \hat{v}_0^x = 0) \wedge (\hat{s}_0 = \hat{E}_Q(\hat{q}_0)).$$
- Well-formedness of timed trajectories. Three points have to be verified for well-formedness: 1) each time period between two discrete transitions should be non-negative; 2) the values of integer-valued variables representing all events should be in $\{1...\|\Sigma\|\}$; 3) the values of variables representing all states should be in $\{1...\|Q\|\}$ for the D_A^F and in $\{1...\|\hat{Q}\|\}$ for V_A^F. As it is about the same timed word, it is enough to check the first two points only once.
  - For the timed trajectory in D_A^F:
$$\Phi^{WF} := (\bigwedge_{i=0}^{k-1} 0 \leqslant t_i) \wedge (\bigwedge_{i=0}^{k-1} 1 \leqslant e_i \wedge e_i \leqslant \|\Sigma\|) \wedge (\bigwedge_{i=0}^{k-1} 1 \leqslant s_i \wedge s_i \leqslant \|Q\|).$$
  - For the timed trajectory in V_A^F:
$$\hat{\Phi}^{WF} := (\bigwedge_{i=0}^{k-1} 1 \leqslant \hat{s}_i \wedge \hat{s}_i \leqslant \|\hat{Q}\|).$$
- Acceptance of the timed trajectory in D_A^F and rejection of the timed trajectory in V_A^F. We formalize here that the timed trajectory represented by values for the predefined variables without hat should be accepted by D_A^F, where final states are faulty ones. And the timed trajectory represented by those for variables with hat should be rejected by V_A^F, where final states are ambiguous ones. Precisely, in each timed trajectory, each pair of adjacent states has to be connected by a transition that is allowed in the corresponding TA. The last state in the trajectory in V_A^F is not a final one, while the last state in D_A^F is a final one with the length bound k.
  - For the timed trajectory in D_A^F:
$$\Phi^{Acc} := (\bigwedge_{i=0}^{k-1} (\bigvee_{(s_i, g, e_i, r, s_{i+1}) \in \delta^E} [[g]]_i \wedge TP_i^r)) \wedge (\bigvee_{q \in Q^F} \hat{s}_k = q).$$

Here $[[g]]_i$ represents that the clock valuations after the i-th step in this timed trajectory, i.e., $v_i^x + t_i$, should satisfy the guard g, such as:

- $[[x \bowtie c]]_i := (v_i^x + t_i) \bowtie c$.
- $[[x - y \bowtie c]]_i := (v_i^x - v_i^y) \bowtie c$.
- $[[g_1 \wedge g_2]]_i := [[g_1]]_i \wedge [[g_2]]_i$.

TP_i^r in the above expression formalizes the time progression, i.e., time transition, by resetting clocks in the subset r and by increasing all other clocks, including the time elapsed from the first fault occurrence if triggered, with the corresponding period t_i:

$$TP_i^r := (\bigwedge_{x \in r} v_{i+1}^x = 0) \wedge (\bigwedge_{x \in (X \setminus r)} v_{i+1}^x = v_i^x + t_i) \wedge (0 \leqslant v_i^F \Rightarrow v_{i+1}^F = v_i^F + t_i).$$

– For the timed trajectory in V_A^F:

$$\hat{\Phi}^{Rej} := (\bigwedge_{i=0}^{k-1} (\bigvee_{(\hat{s}_i, \hat{g}, e_i, \hat{r}, \hat{s}_{i+1}) \in \hat{\delta}^E} [[\hat{g}]]_i \wedge \widehat{TP}_i^r)) \wedge (\bigwedge_{q \in \hat{Q}^F} \hat{s}_k \neq q).$$

In a similar way, $[[\hat{g}]]_i$ for the timed trajectory in V_A^F is encoded as follows:

- $[[\widehat{x \bowtie c}]]_i := (\hat{v}_i^x + t_i) \bowtie c$.
- $[[\widehat{x - y \bowtie c}]]_i := (\hat{v}_i^x - \hat{v}_i^y) \bowtie c$.
- $[[\widehat{g_1 \wedge g_2}]]_i := [[\hat{g_1}]]_i \wedge [[\hat{g_2}]]_i$.

The following is the time progression for this timed trajectory:

$$\widehat{TP}_i^{\hat{r}} := (\bigwedge_{x \in \hat{r}} \hat{v}_{i+1}^x = 0) \wedge (\bigwedge_{x \in (X \setminus \hat{r})} \hat{v}_{i+1}^x = \hat{v}_i^x + t_i).$$

– The timed trajectory contains a fault occurrence (with one fault type, the fault occurrence coding can be simplified as $\|\Sigma_n\| + 1 = e_i$). Furthermore, after the first occurrence of a fault at step i, the value of the variable v_{i+1}^F is assigned to 0 to trigger counting the time elapsed from this fault occurrence (otherwise it stays equal to -1). Finally, we check whether the time elapsed after fault is at most Δ (in absence of given Δ, nothing is added).

$$\Phi^{\Delta} := (\bigwedge_{i=0}^{k-1} (v_i^F = -1 \Rightarrow ((\|\Sigma_n\| < e_i \Rightarrow v_{i+1}^F = 0) \wedge (e_i \leqslant \|\Sigma_n\| \Rightarrow v_{i+1}^F = -1)))) \wedge v_k^F \leqslant \Delta.$$

Now the formula Ψ_{Δ}^k whose satisfiability witnesses manifestability (in time after fault at most Δ) is presented as follows:

$$\Psi_{\Delta}^k := \Phi^{Init} \wedge \hat{\Phi}^{Init} \wedge \Phi^{WF} \wedge \hat{\Phi}^{WF} \wedge \Phi^{Acc} \wedge \hat{\Phi}^{Rej} \wedge \Phi^{\Delta}.$$

Note that for the sake of simplicity, in the proposed formula, there is no state invariant. But considering timed automata without state invariants does not entail any loss of generality as the invariants can be added to the guards [29]. And, if really wanted, the formula Ψ_{Δ}^k can be extended to handle such invariants (by verifying that the clock valuations in each state do not violate the corresponding invariant, which has to be done only when entering the state and leaving it).

4.4 Preliminary Experimental Results

To show the correctness and efficiency of our approach to check manifestability of TA, we show some preliminary experimental results in this section. We realized a prototype implementation in Python by using the SMT solver Z3. The program was executed on the same machine as for the first set of experiences.

Given an SS-DTA A, we construct its fault pair verifier V_A^F as described in Section 4.2, which is done at the syntactic level. Then, based on D_A^F and V_A^F, we encode the formula Ψ_{Δ}^k as described in Section 4.3. The satisfiability of Ψ_{Δ}^k, i.e., the construction of a corresponding adequate timed trajectory, is checked by Z3. With a bounded model checking process, we test for different values of the bound k (length of the trajectory and thus measure of the size of the formula). We report on different versions of three literature examples, including Example 3 which is ex_{00}, that are modified by adding different temporal constraints such that we have both manifestable and non-manifestable models for each of them. Note that some original literature examples are finite automata. For example, ex_{01} is obtained from ex_{00} by changing the guard from q_3 to q_5 as $c \geqslant 3$ and becomes thus non-manifestable because no faulty timed trajectory can manifest itself. Furthermore, considering that such literature examples are normally quite small, to show the scalability, we have tested also some handcrafted systems (hcs), constructed in a partially random way based on the chosen literature ones without changing the verdict. For example, ex_{02} is constructed based on ex_{00} by adding a deterministic TA whose initial state is the destination state of an additional transition with source state q_2, remaining thus manifestable. Similarly, ex_{03} is generated from ex_{01} by adding a deterministic TA without fault to the state q_6, and remains thus non-manifestable.

Table 2 shows part of our experimental results, where column 2 shows the transitions number of the corresponding system model, columns 3 and 4 the upper bound k for the length of timed trajectories and the time upper bound Δ after fault occurrence. Then one can find the size of the formula expressed by its number of clauses, the required memory and the execution time in seconds in the columns 5, 6 and 7, respectively. The final column shows the verdict for each system, where *SAT* witnesses manifestability, while *UNSAT* implies non-manifestability. For the manifestable systems, we try to give k and Δ as small as possible. A small Δ is

Sys	$\|trans.\|$	k	Δ	$\|clauses\|$	$mem.$	time	SAT?
ex_{00}	10	5	1	668	3.98	0.09	SAT
ex_{01}	10	430	10000	89423	18.28	859.72	UNSAT
ex_{02}(hcs)	233	5	3	16781	8.76	1.78	SAT
ex_{03}(hcs)	365	51	10000	510972	17.50	802.53	UNSAT
ex_{04}(hcs)	1782	7	5	441563	29.27	573.36	SAT
ex_{05}(hcs)	1620	15	1000	512308	30.22	1216.82	UNSAT
ex_{10} [27]	6	2	3	243	2.50	0.03	SAT
ex_{11} [27]	6	420	10000	118267	16.62	767.73	UNSAT
ex_{12}(hcs)	287	3	5	6051	5.31	0.63	SAT
ex_{13}(hcs)	381	56	20000	557073	18.21	721.15	UNSAT
ex_{14}(hcs)	2030	7	5	36754	25.09	37.81	SAT
ex_{15}(hcs)	2436	9	20000	521857	32.63	763.02	UNSAT
ex_{20} [31]	9	5	1	578	3.6	0.12	SAT
ex_{21} [31]	9	380	15000	127778	17.60	743.43	UNSAT
ex_{22}(hcs)	296	5	2	16008	5.52	2.1	SAT
ex_{23}(hcs)	315	32	20000	507318	17.53	933.21	UNSAT
ex_{24}(hcs)	2120	5	3	120003	27.09	90.06	SAT
ex_{25}(hcs)	1695	8	23000	423305	28.36	1545.20	UNSAT

Table 2: Experimental results of manifestability checking for SS-DTA

interesting from a practical point of view since it represents how much time after the fault occurrence this fault manifests itself. Another important observation is that for non-manifestable systems, we increase the value of k as well as Δ to show the scalability. From the formulas size, one can see that SMT solvers can check for satisfiability relatively large formulas.

5 Comparison with Opacity

A very close research field worth comparing in a dedicated section is the opacity analysis of discrete event systems, introduced in 2005, which has become a very fertile field of research over the last decade, driven by safety and privacy concerns in network communications and online services (see [30] for a survey). A system is opaque if an external observer (the intruder) is unable to infer a "secret" about the system behavior, i.e., if for any secret behavior, there exists at least one other non-secret behavior that looks the same (for observation) to the intruder. In our context, if we consider the occurrence of a fault as the secret, and thus faulty trajectories as secret behavior and normal trajectories as non-secret behavior, then intuitively fault manifestability and opacity are dual concepts, each one being in some sense the negation of the other. But, as there are various notions of opacity and as fault occurrence is a specific type of secret, the various concepts and their relationships have to be studied carefully.

For DESs, opacity properties are classified into two families: language-based opacity (LBO) and state-based opacity (SBO), depending if a language or a set of states is the secret. The closest to fault manifestability is LBO, which is not surprising as it has been already shown in [35] that related properties such as observability, diagnosability and detectability can all be reformulated as opacity. Indeed, defining, for a system model G with fault F, the secret language L_S as $L_F(G)$ and the non-secret language L_{NS} as $L_N(G)$, then the strong opacity of L_S with respect to L_{NS} and P, defined in [35] as any word of L_S has same projection by P that some word of L_{NS}, is exactly equivalent to the negation of F manifestability. Actually, manifestability is directly related to a special case of opacity, called secrecy [5]. A language property of a system is said strongly secret if it is strongly opaque with respect to its complement. Considering to be faulty as property, i.e., considering as language the faulty trajectories, we obtain that strong secrecy is equivalent to the negation of manifestability. As checking strong secrecy has been proved to be PSPACE-complete [21], it results that checking manifestability is at most PSPACE (actually also PSPACE-complete as we proved, and it is the same for strong opacity).

A smoother LBO property, named weak opacity, is also defined in [35] as some word of L_S has same projection by P that some word of L_{NS}. And, analogously, weak secrecy for a property is defined as its weak opacity with respect to its complement (i.e., $L_{NS} = L(G)\backslash L_S$). It is proved in [57] that checking weak opacity is polynomial. But this concept of weak secrecy is not pertinent in the context of fault manifestability, as its negation would mean that any faulty trajectory is distinguishable from all normal trajectories, which never happens (any trajectory ending by a first occurrence of the fault cannot be distinguished from its normal longer strict prefix). Nevertheless, changing slightly

the definition of L_S as faulty trajectories with at least one observable event after the fault occurrence, then the negation of weak secrecy would be exactly 1-step diagnosability, i.e., each occurrence of the fault is diagnosable from the first observation after its occurrence, which is a very strong property. Our strong manifestability is actually much more smooth, while having no studied equivalence in opacity. Indeed, the negation of strong manifestability means that it exists a trajectory s^F ended by the fault F such that any trajectory with prefix s^F remains secret with respect to normal trajectories and P (i.e., has same observation that some normal trajectory). Thus this particular secrecy does not concern any faulty trajectory as strong secrecy or some faulty trajectory as would do weak secrecy, but any faulty trajectory having some given minimal faulty prefix. One points here a *specificity* of fault manifestability with respect to general secrecy or opacity properties, i.e., by construction the secret language L_S considered is suffix-closed in $L(G)$ (and thus L_{NS} is prefix-closed), expressing that the faults we consider are permanent.

Different in its approach, SBO, introduced by [40] for automata, relates to the intruder ability to infer that the secret is or has been in a given secret state or set of states. Depending on the nature of the secret set, different SBO properties have been defined [30]. Thus one can distinguish among others Current-State Opacity (CSO), if the intruder can never infer, from its observations, whether the current state of the system is a secret state or not (i.e., for every trajectory that leads to a secret state, there exists another trajectory with same observation leading to a non-secret state) and Initial-State Opacity (ISO), if the intruder is never sure whether the system's initial state was a secret state or not (i.e., for every trajectory that originates from a secret initial state, there exists another trajectory with same observation originating from a non-secret initial state). Both CSO and ISO have been proven to be PSPACE-complete and transformation mappings between LBO, CSO and ISO have been studied in [52]. *Note that our approach can be adapted by duality in a very straightforward way to analyze ISO, which can be considered as a special case of manifestability*: it is enough to add an initial state and transitions from this new initial state to the previous initial states, labeled with the fault event for those who are secret and with an unobservable normal event for those who are non-secret. However, the approach proposed in [42] to analyze ISO requires space complexity that is exponential in the number of states of the given automaton, which is hence improved by our method. Regarding CSO, we may have the following constatation: if we define secret states either as all states reachable by a faulty trajectory or those states that are destination states of a fault event, CSO does not apply to manifestability analysis (in particular, in CSO, a trajectory leading to a secret state may be normal).

In fact, most SBO properties are to mask the critical moments of the system, such that they cannot be revealed immediately to an external observer, and do not consider the system behavior once it has exited a secret state (in particular, the set of secret states is not required to be stable). Actually, the more general problem to keep secret the fact the system was in a secret state a few steps ago has been studied under the name of K-step opacity [40,38], i.e., for every trajectory that leads to a secret state and every extension of it with at most K observable events, there exists another trajectory with same observation leading to a non-secret state with an extension with same observation that the previous extension (thus CSO is 0-step opacity). It has been proven to be NP-hard and was extended to infinite-step opacity [41,38,39,56], proven to be PSPACE-hard. Note that here the goal is to mask a secret state by a non-secret state at the same place in the sequence of observations, which is insufficient in general to prevent an intruder for discovering that a secret state was crossed at some place during the last K observations. To avoid this, a language-based translation of K-step opacity is suggested in [38] as trajectory-based K-step opacity, a stronger property ensuring that an intruder cannot determine whether the system has reached a secret state at any point during the last K observations (independently of its exact place). Actually, it looks to be identical to K-step strong opacity, introduced later in [23] to express that, for each trajectory, there exists a trajectory with same observation that never crossed a secret state during the last K observations. But again the dual notion, i.e., the presence of a secret sate in the last K observations necessarily manifests itself, is different from our strong k-manifestability, i.e., any fault event manifests itself in at least one of its future in at most k steps (could be as well k observations) after its occurrence. This is because our approach of fault manifestability, as fault diagnosability, is event-based and not state-based and thus the "faulty" character of a state is not related to that state but to the way it can be reached. In particular, a same state can be reached by a faulty trajectory and a normal one, i.e., a normal, so non-secret, trajectory may contain secret states. In a state-based framework of faulty systems, i.e., if a fault was characteristic of a state and possibly intermittent (i.e., the set of faulty states is not required to be stable), then there would exist a duality worthwhile to study between fault manifestability and SBO.

Opacity analysis for TA has been studied (almost exclusively) in [20], where the (language-based) opacity property for a secret timed language S with respect to a TA A and P is defined as the property that, for any run, it exists a run with same observation that does not belong to S. A state-based opacity property, called L-opacity is also defined, where a set SL of secret locations is said to be opaque with respect to A and P if, for any run, it exists a run with same observation whose last location reached does not belong to SL. L-opacity problem is proven to be undecidable, not only for general TA but also for DTA and even for the subclass of event-recording automata (ERA), where each clock is associated with an event and is reset when this event occurs. It is then shown that opacity can be reduced to L-opacity, with the consequence that opacity problem is undecidable even for ERA with secrets given by ERA. In the context of fault manifestability for a TA A, taking for S the language of faulty runs, we obtain that the opacity of S is equivalent to the negation of manifestability as we defined it. So, our undecidability result for checking manifestability of TA, for which we gave a direct proof, can be obtained by adapting the proof in [20]. In [51] the language-based opacity problem for real-time automata (RTA), a subclass of TA (not comparable with ERA) which has a single clock which is reset at each transition and thus can be regarded as finite automata with time information for each transition, has been proven to be decidable without more precision. But, except this very particular subclass, there is no work, as far as we know, that succeeded to give a sufficient condition on TA such that the opacity problem becomes decidable. In this paper, we proved that, for the subclass of SS-DTA, the manifestability problem is PSPACE-complete and we proposed an SMT-based approach to check it. Another close property called non-interference is to guarantee the safety of flow information by capturing causal dependency between high-level actions (private) and low-level behavior (public). The authors of [25] analyzed different variants of this property for TA and proved some of them decidable by transforming them into weak simulation problem between TA with events set excluding private events and TA with that hiding private events.

6 Related Work

The first approach to verify the diagnosability of DESs is to check the existence of critical pairs based on a deterministic automaton[43], which has exponential complexity in the number of system states. The authors of [31] proposed twin plant method (based on the construction of the verifier) with polynomial complexity. Here we have adapted the twin plant method, plus equivalence checking to verify manifestability. The existence of critical pairs, that excludes diagnosability, does not exclude manifestability. Intuitively, manifestability is a more complicated problem than diagnosability, which was demonstrated by proving that the problem itself is PSPACE-complete instead of polynomial (actually NLOGSPACE-complete) for diagnosability.

In [46,47], the authors proposed different variants of detectability (such as strong detectability) about state estimation. The system is detectable (resp., strongly detectable) if, based on a sequence of observations, one can be sure about the state in which the system is for some given trajectory (resp., all trajectories). They proposed a polynomial algorithm for strong detectability, for which two different trajectories with the same observations witness its violation. However, to analyze detectability, they constructed a deterministic observer that has exponential complexity with the number of system states. Our approach can be adapted to handle state estimation by considering an ambiguous state as one that contains different system states. Thus, we can improve their state estimation by using the improved equivalence checking techniques (e.g., the approach of [14] normally constructs a small part of the deterministic automaton). Furthermore, we proved that the problem of manifestability itself is PSPACE-complete.

The authors of [1,26] proposed an approach for weak diagnosability in a concurrent system by using Petri nets, i.e., impose a constraint of weak fairness by disallowing the enabled transition to be perpetually ignored. The idea is to make impossible some non-diagnosable scenarios in order to upgrade the diagnosability level. They focused on how to get a more appropriate model, based on which a polynomial solution like that for classical diagnosability can be applied.

Two definitions for stochastic diagnosability were introduced and analyzed in [49], which are weaker than diagnosability. A-diagnosability requires that the ambiguous behaviors have a null probability. While AA-diagnosability admits errors in the provided information which should have an arbitrary small probability. Then four variants of diagnosability (FA, IA, FF, IF) were introduced and studied for different probabilistic system models [10,11]. Different ambiguity criteria were then defined according to different types of runs: for faulty runs only or for all runs; for infinite runs or for finite sub-runs. Among them IF-diagnosability (for infinite faulty runs) is the weakest one. Note that IF-diagnosability of a finite probabilistic system is equivalent to A-diagnosability.

The authors of [27,9] analyzed (safe) active diagnosability by introducing controllable actions for (proba-

bilistic) DESs, where the complexity of these problems was also studied. The idea is to design controllers (resp., label activation strategies for probabilistic version) to enable a subset of actions in order to make the system diagnosable (resp., stochastically diagnosable).

On the other hand, the use of TA to model real-time systems has been largely studied since their introduction by [2]. [50] proposed for the first time the diagnosability definition of TA and then adapted the twin plant method to check it before proving the PSPACE-completeness of this problem. As indicated by the author, the reachability in the twin plant for TA can be checked by model-checking tools such as Kronos. However, it is worth noting that the tools such as Kronos and UPPAAL that have implemented the standard forward reachability algorithm based on zones have been shown to be incorrect for diagonal constraints due to the problem of the abstraction operator over zones ([15, 32]). There is no such problem when using normal SMT solvers since they do not use abstraction techniques. Then [16] analyzed the diagnosability problem of TA by constraining the class of diagnosers considered and demonstrated that it is 2EXPTIME-complete for a deterministic TA diagnoser, by using timed game construction.

Some works proposed to use SMT techniques to perform verification on TA with quite good results. In [4], a SMT-based approach was proposed to incrementally analyze TA for some special decidable problems, including universality for deterministic TA and language inclusion of a non-deterministic one into a deterministic one. This is done by adopting bounded version for the sake of efficiency. To verify reachability for TA, [33] introduced a SMT-based bounded model checking to handle non-lasso-shaped infinite runs by integrating region abstraction. More recently, attention was payed to verification of special failure models, called Failure Propagation Models (FPMs), where failure propagation information is abstracted from the original system model. The approach proposed in [17] presents how to encode in SMT the diagnosability problem for a given timed FPM. It is worth noting that TA are totally different from FPMs, the former being considered as original system models, based on which FPMs can be abstracted. However, this transformation is not trivial at all, as demonstrated in ([12, 13]. Then, we have proposed in [28] a new approach to verify diagnosability directly on TA by using SMT techniques, which provides an alternative to systems for which the abstraction to a FPM is not convenient.

7 Conclusion and future work

In this paper we have addressed the formal verification of manifestability for both DESs and real-time systems. To bring an alternative to (stochastic) diagnosability analysis, whose satisfaction is very demanding in terms of sensors placement, we have defined (strong) manifestability, a new weaker property, actually the weakest one to satisfy to have a chance to diagnose a given fault. It is especially useful when the stochastic model is not available during diagnosability analysis. Note that non-manifestability of a system implies its non-stochastic diagnosability, but the converse is not necessarily true. It is worth noting that for today's complex systems, it is not realistic to analyze (stochastic) diagnosability for each type of faults (e.g., hundreds of faults may occur for even one HVAC subsystem in a given building with different categories such as abrupt and degradation [34]). It is more reasonable to verify different properties (e.g., diagnosability for abrupt faults and manifestability for degradation faults) for different faults according to their severity. We also want to emphasize that if stochastic diagnosability is very useful and interesting when the fault occurrence probability distributions are available, very limited studies have been conducted about this availability even for quite mature HVAC systems [34].

We have demonstrated that manifestability problem for finite automata (resp., TA) is PSPACE-complete (resp., undecidable). We further defined SS-DTA, a subclass of deterministic TA, for which this problem becomes PSPACE-complete. It is thus encoded into an SMT formula, which can be checked automatically by an SMT solver. The efficiency and scalability of this approach have also been shown by preliminary experimental results. With such tools at his disposal, the designer may thus check both manifestability and diagnosability of each given fault. If manifestability is not satisfied, he knows that this fault, if it occurs, will never be detectable and he has thus necessarily to add sensors to make it manifest itself. If the fault has been proven manifestable but non-diagnosable, he knows, from the outputs of the algorithms, both a future trajectory of the fault that is distinguishable from correct behavior and another future trajectory that is indistinguishable from correct behavior. Depending on the severity of the fault, of the estimated "probability" of the distinguishable future trajectory and of the impact of the fault in the indistinguishable future trajectory, he can thus decide to change or add sensors and check again both manifestability and diagnosability.

One interesting future work is to find out a larger subclass of TA than SS-DTA, for which manifestabil-

ity problem is decidable and to relate this problem to opacity. Another perspective is to study this problem for distributed systems composed of a set of components with a modular method, more interestingly, by taking into account probabilistic aspects.

References

1. A. Agarwal, A. Madalinski, and S. Haar. Effective Verification of Weak Diagnosability. In *Proceedings of the 8th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS'12)*, pages 636–641. IFAC, 2012.
2. R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, April 1994.
3. 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.
4. B. Badban and M. Lange. Exact incremental analysis of timed automata with an smt-solver. In *Proceedings of International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'11)*, volume 6919 of *Lecture Notes in Computer Science*. Springer, 2011.
5. E. Badouel, M. Bednarczyk, A. Borzyszkowski, B. Caillaud, and P. Darondeau. Concurrent secrets. *Discrete Event Dynamic Systems*, 17(4):425–446, 2007.
6. C. Baier, N. Bertrand, P. Bouyer, and T. Brihaye. When are timed automata determinizable? In *Proceedings of Automata, Languages and Programming, 36th International Colloquium, (ICALP 2009), Part II, Rhodes, Greece, July 5-12, 2009*, pages 43–54, 2009.
7. M. Basarkar, X. Pang, L. Wang, P. Haves, and T. Hong. Modeling and simulation of HVAC faults in EnergyPlus. In *Proceedings of Building Simulation 2011: 12th Conference of International Building Performance Simulation Association*, pages 2897–2903, Jan 2011.
8. 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.
9. N. Bertrand, E. Fabre, S. Haar, S. Haddad, and L. Hélouët. Active diagnosis for probabilistic systems. In *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014*, pages 29–42, 2014.
10. N. Bertrand, S. Haddad, and E. Lefaucheux. Foundation of diagnosis and predictability in probabilistic systems. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 417–429, 2014.
11. N. Bertrand, S. Haddad, and E. Lefaucheux. Diagnosis in infinite-state probabilistic systems. In *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, pages 37:1–37:15, 2016.
12. B. Bittner, M. Bozzano, and A. Cimatti. Automated synthesis of timed failure propagation graphs. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*, pages 972–978, 2016.
13. B. Bittner, M. Bozzano, A. Cimatti, and G. Zampedri. Automated verification and tightening of failure propagation models. In *Proceedings of the 30th Conference on Artificial Intelligence (AAAI'16)*, pages 907–913, 2016.
14. F. Bonchi and D. Pous. Checking NFA Equivalence with Bisimulations Up to Congruence. In *Proceedings of 40th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL-2013)*, pages 457–468. ACM, 2013.
15. P. Bouyer. Untameable timed automata. In *Proceedings of the Annual Symposium on Theoretical Aspects of Computer Science (STACS'03)*, volume 2607 of *Lecture Notes in Computer Science*, pages 620–631. Springer, 2003.
16. P. Bouyer, F. Chevalier, and D. D'Souza. Fault diagnosis using timed automata. In *Proceedings of International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, Lecture Notes in Computer Science. Springer, 2005.
17. M. Bozzano, A. Cimatti, M. Gario, and A. Micheli. Smt-based validation of timed failure propagation graphs. In *Proceedings of the 29th Conference on Artificial Intelligence (AAAI'15)*, pages 3724 –3730, 2015.
18. J.R. Büchi. On a decision method in restricted second order arithmetic. *Z. Math. Logik Grundlag. Math*, 6:66–92, 1960.
19. C. G. Cassandras and S. Lafortune. *Introduction To Discrete Event Systems, Second Edition*. Springer, 2008.
20. F. Cassez. The dark side of timed opacity. In *Proceedings of the 3rd International Conference on Information Security and Assurance (ISA'09), Seoul, South Korea, June, 2009*, pages 21–30, 2009.
21. F. Cassez, J. Dubreil, and H. Marchand. Dynamic observers for the synthesis of opaque systems. In *Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis (ATVA09), Macao, China, October, 2009*, pages 352–367, 2009.
22. V. Diekert, P. Gastin, and A. Petit. Removing ϵ-transitions in timed automata. In *Proceedings of 14th Annual Symposium on Theoretical Aspects of Computer Science (STACS 97), Lübeck, Germany, February 1997*, pages 583–594, 1997.
23. Y. Falcone and H. Marchand. Enforcement and validation (at runtime) of various notions of opacity. *Discrete Event Dynamic Systems*, 25(4):531–570, 2014.
24. O. Finkel. Undecidable problems about timed automata. In *Proceedings of 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2006), Paris, France, September 25-27, 2006*, pages 187–199, 2006.
25. G. Gardey, J. Mullins, and O. Roux. Non-interference control synthesis for security timed automata. *Electronic Notes in Theoretical Computer Science*, 180(1):35–53, June 2007.
26. V. Germanos, S. Haar, V. Khomenko, and S. Schwoon. Diagnosability under Weak Fairness. *ACM Trans. Embedded Comput. Syst.*, 14(4), 2015.
27. S. Haar, S. Haddad, T. Melliti, and S. Schwoon. Optimal constructions for active diagnosis. *J. Comput. Syst. Sci.*, 83(1):101–120, 2017.
28. L. He, L. Ye, and P. Dague. Smt-based diagnosability analysis of real-time systems. In *Proceedings of 10th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS 2018)*, pages 1059–1066, 2018.
29. F. Herbreteau, B. Srivathsan, and I. Walukiewicz. Lazy abstractions for timed automata. In *Proceedings of 25th*

International Conference on Computer Aided Verification (CAV 2013), Saint Petersburg, Russia, July 13-19, 2013, pages 990–1005, 2013.
30. R. Jacob, J.-J. Lesage, and J.-M. Faure. Opacity of discrete event systems: models, validation and quantification. In *Proceedings of 5th IFAC Workshop on Dependable Control of Discrete Systems (DCDS'15), Cancun, Mexico, May, 2015*, pages 174–181, 2015.
31. 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.
32. B. Johan and Y. Wang. On clock difference constraints and termination in reachability analysis of timed automata. In *Proceedings of the 5th International Conference on Formal Engineering Methods (ICFEM'2003)*, volume 2885 of *Lecture Notes in Computer Science*, pages 491–503. Springer, 2003.
33. R. Kindermann, T. Junttila, and I. Niemela. Beyond lassos: Complete smt-based bounded model checking for timed automata. In *Proceedings of Joint FMOODS 2012 and FORTE 2012*, volume 7273 of *Lecture Notes in Computer Science*. Springer, 2012.
34. Y. Li and Z. O'Neill. A critical review of fault modeling of HVAC systems in buildings. *Building Simulation*, 11(5):953–975, Oct 2018.
35. F. Lin. Opacity of discrete event systems and its applications. *Automatica*, 47(3):496–503, 2011.
36. D. Papineau. *Philosophical Naturalism*. Blackwell Pub, 1993.
37. 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.
38. A. Saboori. *Verification and enforcement of state-based notions of opacity in discrete event systems*. Ph.D. thesis, University of Illinois at Urbana-Champaign, 2011.
39. A. Saboori and C. N. Hadjicostis. Verification of infinite-step opacity and complexity considerations. *IEEE Transactions on Automatic Control*, 57(5):1265–1269, 2012.
40. A. Saboori and C.N. Hadjicostis. Notions of security and opacity in discrete event systems. In *Proceedings of 46th IEEE Conference on Decision and Control (CDC07), New Orleans, LA, USA, December, 2007*, pages 5056–5061, 2007.
41. A. Saboori and C.N. Hadjicostis. Verification of infinite-step opacity and analysis of its complexity. In *Proceedings of 2nd IFAC Workshop on Dependable Control of Discrete Systems (DCDS'09), Bari, Italy, June, 2009*, pages 46–51, 2009.
42. A. Saboori and C.N. Hadjicostis. Verification of initial-state opacity in security applications of discrete event systems. *Information Sciences*, 246:115–132, 2013.
43. 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.
44. 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.
45. 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.
46. S. Shu and F. Lin. Detectability of Discrete Event Systems with Dynamic Event Observation. *Systems and Control Letters*, 59(1):9–17, 2010.
47. S. Shu and F. Lin. I-Detectability of Discrete-Event Systems. *IEEE T. Automation Science and Engineering*, 10(1):187–196, 2013.
48. 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.
49. D. Thorsley and D. Teneketzis. Diagnosability of stochastic discrete-event systems. *IEEE Trans. Automat. Contr.*, 50(4):476–492, 2005.
50. 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.
51. L. Wang, N. Zhan, and J. An. The opacity of real-time automata. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2845–2856, 2018.
52. Y.C. Wu and S. Lafortune. Comparative analysis of related notions of opacity in centralized and coordinated architectures. *Discrete Event Dynamic Systems*, 23(3):307–339, 2013.
53. 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.
54. 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.
55. 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.
56. X. Yin and S. Lafortune. A new approach for the verification of infinite-step and k-step opacity using two-way observers. *Automatica*, 80:162–171, 2017.
57. B. Zhang, S. Shu, and F. Lin. Polynomial algorithms to check opacity in discrete event system. In *Proceedings of the 24th Chinese Control and Decision Conference (CCDC12)*, pages 763–769, 2012.