Model Based Diagnosis of Timed Automata with Model Checkers

Zhe Feng^1 and **Alban Grastien**^{1}
^{1}Australian National University, Canberra, Australia
e-mail: {Zhe.Feng,Alban.Grastien}@anu.edu.au

Abstract

We propose the use of model checkers to solve problems of diagnosis of systems modelled with timed automata. We first show how to reduce a diagnosis query (i.e., the problem of deciding if a set of diagnosis hypotheses intersects the diagnosis) to a model checking problem. We then show how to define the queries in order to retrieve the minimal diagnosis. We also show how to use conflicts to accelerate the computation. Finally we discuss a variant when the system is diagnosable.

1 Introduction

We are interested in model-based diagnosis (MBD) of timed automata [Alur and Dill, 1994]. A timed automaton (TA) is a modelling framework for dynamic systems with finite discrete state space and time constraints between events. Existing approaches developed for discrete event systems are not suitable because of the infinite state space of TA (see the Related Work section).

In this paper we propose to use model checkers to compute the diagnosis. Model checkers are tools that were developed to determine whether certain systems satisfy specific properties. After presenting the background knowledge (Section 2) the contributions of this work are: In Section 3 we show that deciding whether the diagnosis intersects a given set of hypotheses can be expressed as a Computational Tree Logic (CTL) formula over the synchronous product of three timed automata (one that represents the model, one the observations, and one the faults). Then we show how to choose the sets of hypotheses in order to retrieve the minimal diagnosis. In Section 4 we show how to use the conflicts to improve the search time. In Section 5 we show how to adapt this algorithm for diagnosable systems. Finally we compare our work with the existing approaches (Section 6) and provide some experimental results (Section 7).

2 Background

2.1 Timed Automaton

Discrete event systems (DES) are a modelling framework for dynamic systems whose behaviour is described through discrete changes (the *events*). A timed automaton is an extension of DES proposed by Alur and Dill [1994] that includes information about durations between events.

Timed automata rely on the notion of *clocks* whose values increase at constant rate over time. Given a finite set $\mathcal{X}$ of clocks, a *clock constraint* is a finite, possibly empty, conjunction of *atomic constraints* of the form $x \bowtie c$ or $x - y \bowtie c$ where x and y are clocks, c is a constant, and $\bowtie$ is an operator from $\{<, \leq, =, \geq, >\}$. $\mathcal{C}(\mathcal{X})$ is the set of all constraints that can be defined over the clocks $\mathcal{X}$. Given a function ν that assigns a value to each clock and a constraint $\phi \in \mathcal{C}(\mathcal{X})$ on these clocks, $\nu \models \phi$ indicates that the constraint is satisfied by the assignment ν. The symbol $\top$ represents the empty constraint and is satisfied by any function ν.

A timed automaton is then a finite state machine where transitions are labelled by events, and clocks constrain the current state and the transitions. Formally a *timed automaton* is a tuple $TA = \langle S, \Sigma, \mathcal{X}, T, I, s_0 \rangle$ where

- S is a finite set of *discrete states* and s_0 is the *initial discrete state*;
- Σ is a finite set of *events*; $\mathcal{X}$ is a finite set of clocks;
- $T \subseteq S \times \mathcal{C}(\mathcal{X}) \times \Sigma \times 2^{\mathcal{X}} \times S$ is a finite set of *discrete transitions*; and
- $I : S \to \mathcal{C}(\mathcal{X})$ is the *invariant function* that associates each discrete state with a constraint.

A *state* of a timed automaton is a pair $q = \langle s, \nu \rangle$ where $s \in S$ is a discrete state and $\nu : \mathcal{X} \to \mathbb{Q}^+ \cup \{0\}$ is a function that assigns a non-negative rational value to each clock. $disc(q)$ refers to the discrete state s of q and $time(q)$ refers to the clock assignment ν. A state must always satisfy its invariant, i.e., $time(q) \models I(disc(q))$.

The timed automaton defines two types of transitions: event occurrence and time passing. An *event occurrence* leads the system from state q to state q' if there exists a discrete transition $\langle s, G, e, R, s' \rangle \in T$ such that i) s and s' are the discrete states of q and q' ($disc(q) = s$ and $disc(q') = s'$), ii) the *guard* G of the transition is satisfied in q ($time(q) \models G$), and iii) the time assignment of q' is the same as that of q except for the resets specified by R:

$$\forall x \in \mathcal{X}.\ time(q')(x) = \begin{cases} 0 & \text{if } x \in R, \\ time(q)(x) & \text{otherwise.} \end{cases}$$

This is denoted $q \xrightarrow{e} q'$.

A *time passing* leads the system from state q to state q' if i) their discrete states are identical ($disc(q) = disc(q')$) and ii) the clock assignments of q' are postponed by the same positive amount $\tau \in \mathbb{Q}^+$ compared to q ($\forall x \in \mathcal{X}.\ time(q')(x) = time(q)(x) + \tau$). This is denoted $q \xrightarrow{\tau} q'$.

The *initial state* is $init(TA) = \langle s_0, \nu_0 \rangle$ where ν_0 is the function that assigns 0 to each clock. A run ρ is a finite sequence of transitions $q_0 \xrightarrow{\ell_1} \dots \xrightarrow{\ell_n} q_n$. $Runs(TA, q)$ is the set of runs of TA that start from state q, and $Runs(TA)$ the set of runs that start from the initial state. The duration of a run, $|\rho|$, is the sum of all ℓ_i that are numbers (not events).

Given a run $\rho = q_0 \xrightarrow{\ell_1} \dots \xrightarrow{\ell_n} q_n$, ρ *goes through* state q ($q \in \rho$) if

- either $q = q_i$ for some $i \in \{0, \dots, n\}$,
- or $q_{i-1} \xrightarrow{\tau} q \xrightarrow{\ell_i - \tau} q_i$ for some $i \in \{1, \dots, n\}$ and some $\tau \in \mathbb{Q}^+$, (i.e., the transition $q_{i-1} \xrightarrow{\ell_i} q_i$ passes through the state q).

A *distributed set of timed automata* is a set $\{TA^1, \dots, TA^p\}$ of p timed automata whose sets of clocks are disjoint ($\mathcal{X}^i \cap \mathcal{X}^j = \emptyset$ whenever $i \neq j$). It implicitly represents the time automaton $TA = TA^1 || \dots || TA^p = \langle S, \Sigma, \mathcal{X}, T, I, s_0 \rangle$ defined by

- $S = S^1 \times \dots \times S^p$, $s_0 = \langle s_0^1, \dots, s_0^p \rangle$,
- $\Sigma = \bigcup_{i \in \{1,\dots,p\}} \Sigma^i$, $\mathcal{X} = \bigcup_{i \in \{1,\dots,p\}} \mathcal{X}^i$,
- T is the set of transitions defined below, and
- $I(\langle s_1, \dots, s_p \rangle) = \bigwedge_{i \in \{1,\dots,p\}} I^i(s_i)$.

The transitions of T are precisely those transitions $\langle \langle s_1, \dots, s_p \rangle, \bigwedge_{i \in \{1,\dots,p\}} G_i, e, \bigcup_{i \in \{1,\dots,p\}} R_i, \langle s'_1, \dots, s'_p \rangle \rangle$ such that for every $i \in \{1, \dots, p\}$,

- either $e \in \Sigma^i$ and then $\langle s_i, G_i, e, R_i, s'_i \rangle \in T^i$ is a transition;
- or $e \notin \Sigma^i$ and then $s_i = s'_i$, $G_i = \top$, and $R_i = \emptyset$.

Essentially the timed automaton takes a transition on all timed automata TA^i that mention event e.

2.2 Diagnosis

The system is modelled by the timed automaton TA^{M} (or a distributed set of timed automata) and is partially observed. In traditional DES fashion, partial observability is modelled by the partitioning of the events into *observable* and *unobservable* events, Σ_{o} and Σ_{u}. The *trace* of a run $\rho = q_0 \xrightarrow{\ell_1} \dots \xrightarrow{\ell_n} q_n$ is the list of transition labels that compose it ($[\ell_1, \dots, \ell_n]$) in which consecutive durations are added up. For instance, a run $q_0 \xrightarrow{e_1} q_1 \xrightarrow{2} q_2 \xrightarrow{1.5} q_3 \xrightarrow{e_2} q_4$ yields the trace $[e_1, 3.5, e_2]$. The *observable trace* $obs(\rho)$ of the run is the trace of the run in which the unobservable events are stripped out and the duration are, again, added up. For instance, a run $q_0 \xrightarrow{e_1} q_1 \xrightarrow{2} q_2 \xrightarrow{e_2} q_3 \xrightarrow{1.5} q_4 \xrightarrow{e_3} q_5$, in which e_1 and e_3 are observable and e_2 is not, yields the trace $[e_1, 2, e_2, 1.5, e_3]$ and the observable trace $[e_1, 3.5, e_3]$.

Diagnosis is concerned with deciding whether the system is experiencing faults, and if so which ones. This is modelled by assuming that the unobservable events are further partitioned into *faulty* and *non-faulty events* (Σ_{n} and Σ_{f}). The *diagnosis hypothesis* associated with a run $\rho = q_0 \xrightarrow{\ell_1} \dots \xrightarrow{\ell_n} q_n$ is then the set of faults that take place in the run: $hypo(\rho) = \Sigma_{\mathrm{f}} \cap \{\ell_i \mid i \in \{1, \dots, n\}\}$.

The problem of diagnosis then consists in finding all the hypotheses that are consistent with the observation made on the system's run. The minimal diagnosis restricts this set to all subset-minimal ones.

Definition 1. *A* diagnostic problem *is a tuple $\mathcal{P} = \langle TA^{\mathrm{M}}, \Sigma_{\mathrm{o}}, \Sigma_{\mathrm{f}}, \sigma \rangle$ where TA^{M} is the model, Σ_{o} is the set of observable events, Σ_{f} is the set of faulty events, and σ is the observed sequence.*

Given the diagnostic problem $\mathcal{P} = \langle TA^{\mathrm{M}}, \Sigma_{\mathrm{o}}, \Sigma_{\mathrm{f}}, \sigma \rangle$, a diagnosis candidate *is an hypothesis δ that is consistent with the observed sequence: $\exists \rho \in Runs(TA^{\mathrm{M}})$. $hypo(\rho) = \delta \wedge obs(\rho) = \sigma$. The* diagnosis $\Delta(\mathcal{P})$ *is the set of diagnosis candidates. The* minimal diagnosis $\Delta_{\min}(\mathcal{P})$ *is the set of all subset-minimal candidates: $\{ \delta \in \Delta(\mathcal{P}) \mid \forall \delta' \in \Delta(\mathcal{P}).\ \delta' \subseteq \delta \Rightarrow \delta' = \delta \}$.*

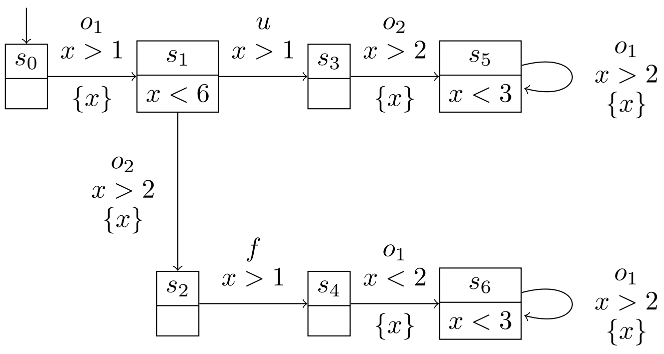

Figure 1: Example model.

Consider the example of Fig. 1 [He *et al.*, 2018] with the observed sequence $[1.5, o_1, 3.5, o_2, 1.5, o_1]$. One possible run that matches this sequence is $q_0 \xrightarrow{1.5} q_1 \xrightarrow{o_1} q_2 \xrightarrow{3.5} q_3 \xrightarrow{o_2} q_4 \xrightarrow{1.2} q_5 \xrightarrow{f} q_6 \xrightarrow{0.3} q_7 \xrightarrow{o_1} q_8$. There are infinitely many other consistent runs (with various delays for the transitions between q_4 and q_5) but they all include the fault f. Therefore the diagnosis is $\Delta = \{ \{f\} \}$.

2.3 Computation Tree Logic and Model Checking

Model checking is the problem of deciding whether an artifact satisfies a given specification [Clarke *et al.*, 2000b]. The specification language we use in this paper is Computational Tree Logic (CTL). This language is sophisticated, but we limit ourselves to specifications of the form $\varphi = EF\ Q$, where each Q is a set of states, that asks the question "does the model exhibit a run that ends in a state from Q. In practice, Q will be replaced with $Q_1 \wedge \dots \wedge Q_k$ where each Q_i is a subset of one of the S^j, with the meaning that the final state of the run should be in Q_i for the right TA.

3 Diagnosis with Model Checkers

In this section we show how MBD of timed automata can be performed using model checkers. The basic idea consists in building a system that includes three components (the model, an "observation automaton" that represents the observed sequence, and a "faulty automaton" that records the faulty events that occurred on the system) together with a query in CTL that is satisfied iff there exists a run that matches the observations and that ends in a specific state of the faulty automaton. This framework is refined in the next section for a more sophisticated algorithm.

3.1 Observation Automaton

The observation automaton is defined in such a way that the runs that the model checker will consider are precisely those that match the observed sequence. To this end, we

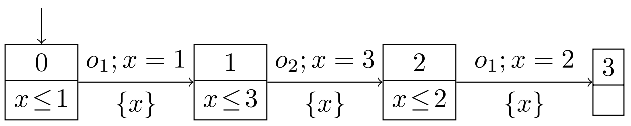

Figure 2: Observation automaton for the observation sequence $[1, o_1, 3, o_2, 2, o_1]$.

assume wlog that $\sigma = [\tau_0, o_1, \tau_1, o_2, \ldots, \tau_{k-1}, o_k]$ is such that the final element of the trace is an observable event and the initial one is not. $|\sigma| = k$ (not to be confused with $|\rho|$) is the number of observed events.

Definition 2. *Let Σ_o be the set of observable events and $\sigma = [\tau_0, o_1, \ldots, \tau_{k-1}, o_k]$ be the observed sequence. The* observation automaton *is the timed automaton $TA^O(\Sigma_o, \sigma) = \langle S^O, \Sigma^O, \mathcal{X}^O, T^O, I^O, s_0^O\rangle$ defined by:*

- $S^O = \{0, 1, \ldots, k\}$ *includes $k+1$ states, $s_0^O = 0$,*
- $\Sigma^O = \Sigma_o$ *includes all observable events,*
- $\mathcal{X}^O = \{x\}$ *includes one fresh clock,*
- $T^O = \{t_i\}_{i\in\{1,\ldots,k\}}$ *where for each $i \in \{1, \ldots, k\}$, $t_i = \langle i-1, (x = \tau_{i-1}), o_i, \{x\}, i\rangle$, and*
- $I^O(i) = \begin{cases} (x \leq t_i) & \text{for } i < k \\ \top & \text{for } i = k. \end{cases}$

We illustrate Definition 2 with an example on Figure 2. It should be clear that any run that ends up in state k generates precisely the observed sequence.

3.2 Faulty Automaton

Similarly, we define an automaton that records the set of faults that occurred on the system. This automaton does not use any clock.

Definition 3. *Given a set of faults Σ_f the corresponding* faulty automaton *is the timed automaton $TA^F(\Sigma_f) = \langle S^F, \Sigma^F, \mathcal{X}^F, T^F, I^F, s_0^F\rangle$ defined by:*

- $S^F = 2^{\Sigma_f}$ *is the powerset of Σ_f; $s_0^F = \emptyset$;*
- $\Sigma^F = \Sigma_f$*; $\mathcal{X}^F = \emptyset$;*
- $T^F = \{\langle h, \top, f, h \cup \{f\}\rangle \mid h \subseteq \Sigma_f,\ f \in \Sigma_f\}$*; and*
- $I^F(h) = \top$ *for all $h \in S^F$.*

We illustrate Definition 3 with an example on Figure 3. In practice the faulty automaton can be defined as the synchronous product of a collection of faulty automata, each of which built from a different event: $TA^F(f_1)|| \ldots || TA^F(f_m)$ where $\Sigma_f = \{f_1, \ldots, f_m\}$.

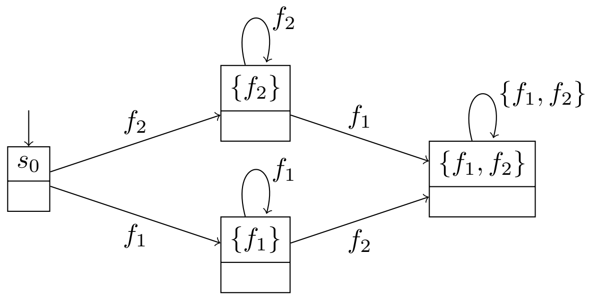

Figure 3: Faulty automaton for the set of faulty events $\{f_1, f_2\}$.

Algorithm 1 Naïve diagnosis algorithm

```
1: input: diagnostic problem $\mathcal{P} = \langle TA^M, \Sigma_o, \Sigma_f, \sigma\rangle$
2: output: minimal diagnosis $\Delta_{\min}(\mathcal{P})$
3: $O := \{\emptyset\}$; $R := \{\}$
4: while $O \neq \emptyset$ do
5:     $h := pop(O)$
6:     if $\exists\delta \in R.\ \delta \subseteq h$ then
7:         continue
8:     if test of $\{h\}$ is positive then
9:         $R := R \cup \{h\}$
10:    else
11:        for all $f \in \Sigma_f \setminus h$ do
12:            if $\nexists h' \in O.\ h' \subseteq h \cup \{f\}$ then
13:                $O := O \cup \{h \cup \{f\}\}$
14: return $R$
```

3.3 Testing a Set of Hypotheses

We now present the basic ingredient that we use to compute MBD of TA^M. This basic ingredient is a consistency testing between the model, the observed sequence, and a set of hypotheses.

Definition 4. *Given the* diagnostic problem *is a tuple $\mathcal{P} = \langle TA^M, \Sigma_o, \Sigma_f, \sigma\rangle$ and given the set of hypotheses $H \subseteq 2^{\Sigma_f}$, a* (diagnostic) test *verifies the validity of the statement*

$$\Delta(\mathcal{P}) \cap H = \emptyset.$$

The test is positive *if the intersection is non-empty.*

We use a model checker to perform the test.

Definition 5. *Given the* diagnostic problem $\mathcal{P} = \langle TA^M, \Sigma_o, \Sigma_f, \sigma\rangle$ *and given the set of hypotheses $H \subseteq 2^{\Sigma_f}$, the* diagnostic CTL query *is* $\text{CTL}(H) = EF\ (|\sigma| \wedge H)$.

Lemma 1. *The diagnosis test for H is positive iff TA^M, $TA^O(\Sigma_o, \sigma)$, $TA^F(\Sigma_f) \models \text{CTL}(H)$ holds.*

Proof sketch: $\text{CTL}(H)$ asks the model checker to search for a specific run from the model: this run should match the observed sequence in order to end up in discrete state $|\sigma|$; it should also match one of the hypotheses in H in order to end up in one of the discrete states H. The test is indeed positive iff such a run exists. QED

3.4 A Simple Diagnosis Algorithm

Lemma 1 allows us to design a diagnostic algorithm by carefully choosing which sets H of hypotheses to test. A naïve implementation is proposed on Algo. 1. Starting with the minimal hypothesis ($\emptyset$) the algorithm maintains a list of hypotheses in the open list O. Whenever an hypothesis h from O is proved to be consistent with the observation sequence it is added to the result R; by construction, it is guaranteed that no strict subset of h is also a result. If h does not belong to R all supersets of h that consists in adding a single fault to h are added to O, assuming there is no strict subset of these in O.

4 A Conflict-Based Approach to Diagnosis

Model-based diagnosis has traditionally been solved through the use of *conflicts*, a notion developped and exploited by De Kleer and Williams [1987] and Reiter [1987]. Conflicts were initially proposed for diagnosis of circuit systems, but their use has been adapted for the diagnosis of

Algorithm 2 Computing a minimal conflict

```
input: Diagnostic problem $\mathcal{P}$, conflict $F \subseteq \Sigma_f$
output Minimal conflict $\mathcal{C}$ such that $\mathcal{C} \subseteq F$
$\mathcal{C} := F$
for all $f \in F$ do
    if $\mathcal{C} \setminus \{f\}$ is a conflict then
        $\mathcal{C} := \mathcal{C} \setminus \{f\}$
return $\mathcal{C}$
```

dynamic systems too [Cordier *et al.*, 2004; Grastien *et al.*, 2012].

In our context, given a diagnostic problem $\mathcal{P}$ a *conflict* is a subset $\mathcal{C} \subseteq \Sigma_f$ of faults such that any diagnosis candidate contains one of these faults:

$$\forall \delta \in \Delta(\mathcal{P}). \quad \delta \cap \mathcal{C} \neq \emptyset.$$

A conflict is *minimal* if none of its strict subsets is a conflict. Given a collection $X = \{C_1, \ldots, C_k\}$ of sets a *hitting set* is a set S that intersects with all sets of the collection:

$$\forall C \in X. \quad S \cap C \neq \emptyset;$$

a hitting set is *minimal* if none of its strict subsets is also a hitting set. It is well known that the minimal diagnosis is the set of minimal hitting sets of the collection of minimal conflicts.

The minimal diagnosis can be deduced by first computing all minimal conflicts and then computing their minimal hitting sets. In a different approach, conflicts are discovered during the diagnosis algorithm rather than as a preprocessing step, and these conflicts are used to reduce the search space.

We now show i) how model checkers can be used to determine if a set of faults is a conflict, ii) how to compute minimal conflicts with model checkers, and iii) how to use these conflicts in a revised algorithm.

A set $F \subseteq \Sigma_f$ of faults is a conflict iff the set $H = \{h \subseteq \Sigma_f \mid h \subseteq \Sigma_f \setminus F\}$ of hypotheses does not intersect the diagnosis [Slaney, 2014]. For instance in a scenario $\Sigma_f = \{f_1, f_2, f_3, f_4\}$, $F = \{f_1, f_2\}$, the set H is defined as $\{\ \emptyset, \{f_3\}, \{f_4\}, \{f_3, f_4\}\ \}$. We define a dedicated CTL formula to represent this.

Definition 6. *Given the diagnostic problem* $\mathcal{P} = \langle TA^{\mathrm{M}}, \Sigma_o, \Sigma_f, \sigma \rangle$*, the* conflict CTL query *associated with the set F of faults is* $\mathrm{CTL}_c(\mathcal{P}, F) = EF(|\sigma| \wedge \bigwedge_{f \in F} \neg\{f\})$.

Lemma 2. *The set F of faults is a conflict iff* TA^{M}, $TA^{\mathrm{O}}(\Sigma_o, \sigma)$, $TA^{\mathrm{F}}(\Sigma_f) \not\models \mathrm{CTL}_c(F)$ *holds.*

We can use Lemma 2 to devise an algorithm that computes a minimal conflict by using the fact that the quality of being a conflict is monotonic, i.e., it satisfies the following property:

$$S_1 \subseteq S_2 \ \wedge S_1 \text{ is a conflict } \Rightarrow S_2 \text{ is a conflict.}$$

Therefore it is possible to remove each fault from F greedily, and to make sure that the set of faults remains inconsistent with the diagnostic problem. This is summarised on Algo. 2.

It is now possible to incorporate the conflicts into the diagnostic algorithm itself. Conflicts are useful in two aspects: first they help identify early that some hypotheses are not diagnosis candidates because they do not intersect these hypotheses; second they reduce the number of new hypotheses that need to be included into the open list.

Algorithm 3 Diagnostic algorithm using conflicts

```
1: input: Diagnostic problem $\mathcal{P}$
2: output: minimal diagnosis $\Delta_{\min}(\mathcal{P})$
3: $O := \{\emptyset\}$; $R := \{\}$; $Conflicts = \{\}$
4: while $O \neq \emptyset$ do
5:     $h := pop(O)$
6:     if $\exists \delta \in R.\ \delta \subseteq h$ then
7:         continue
8:     $\mathcal{C} := \bot$                    // Will differ from $\bot$ if $h \notin \Delta$
9:     for all $\mathcal{C}' \in Conflicts$ do
10:        if $\mathcal{C}' \cap h = \emptyset$ then
11:            $\mathcal{C} := \mathcal{C}'$
12:    if $\mathcal{C} = \bot\ \wedge$ test of $\{h\}$ is positive then
13:        $R := R \cup \{h\}$
14:    else
15:        $\mathcal{C} :=$ compute-min-conflict$(\mathcal{P}, \Sigma_f \setminus h)$
16:        $Conflicts := Conflicts \cup \{\mathcal{C}\}$
17:    if $\mathcal{C} \neq \bot$ then
18:        for all $f \in \mathcal{C}$ do
19:            if $\nexists h' \in O.\ h' \subseteq h \cup \{f\}$ then
20:                $O := O \cup \{h \cup \{f\}\}$
21: return $R$
```

It can be shown that Algo. 3 terminates and returns the minimal diagnosis. The proof relies on the following remarks: for each minimal diagnosis candidate δ and at the beginning of every loop, there is always an hypothesis $h \in R \cup O$ such that $h \subseteq \delta$ (this is guaranteed in particular by the fact that O is updated with faults from the conflict). Furthermore since O is empty at the end and since R only contains minimal candidates, all minimal candidates are in R at the end of the algorithm. Finally we can show that the set O never stays the same from one iteration to the next, which proves the termination of the algorithm.

As an example to show the benefits of Algo. 3 over Algo. 1, consider a situation where the set of faulty events contains ten elements ($\Sigma_f = \{f_1, \ldots, f_{10}\}$) and the only minimal candidate contains only fault f_1: $\Delta_{\min} = \{\ \{f_1\}\ \}$. In this scenario Algo. 1 will test set $\{f_1\}$ as well as all subsets of $\Sigma_f \setminus \{f_1\}$, i.e., a total of 513 tests. In comparison, Algo. 2 will first test hypothesis $\emptyset$, will then test what happens when each fault is removed (ten more tests), and will finally test hypothesis $\{f_1\}$, for a total of only twelve tests.

5 Diagnosable Models

We now briefly discuss our method in a diagnosable context, and how this context allows for a more efficient procedure.

Diagnosability is the property that whenever a fault occurs it leads to the production of an observed sequence that betrays the fault unmistakably. Diagnosability is generally desirable from a design perspective because it means that faults can be detected and precisely identified, which allows the observer to take appropriate action.

Formally a *τ-diagnoser D* (where $\tau \in \mathbb{Q}^+$ is called the *delay*) is a function that reads an observed trace and returns a single hypothesis such that

1. for all run $\rho \in Runs(TA^{\mathrm{M}})$, $D(obs(\rho)) \subseteq hypo(\rho)$, i.e., the diagnoser does not pretend to identify faults that did not happen;

2. for all run $\rho\rho' \in Runs(TA^{\mathrm{M}})$ such that $|\rho'| \geq \tau$, $D(obs(\rho\rho')) \supseteq hypo(\rho)$, i.e., the diagnoser should identify faults that occurred at least τ ago.

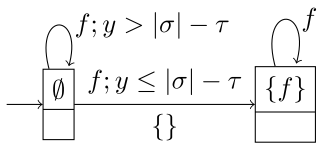

Figure 4: Faulty automaton for the set of faulty events $\{f_1, f_2\}$.

τ-diagnosability is the property that a τ-diagnoser exists. Such a diagnoser may not exist because different runs with different hypotheses may yield the same observable trace. A τ-diagnoser $D(\sigma)$ does not always match the real diagnostic hypothesis $hypo(\rho)$ or even the best diagnosis (i.e., the following may hold: $D(\sigma) \notin \Delta(\sigma)$), but this imprecision is accepted because it only involves faults that recently occurred (less than τ ago), and it is assumed that the negative repercussion of delaying their identification is therefore limited. In this section we therefore focus on the problem of building of τ-diagnoser, assuming the model is τ-diagnosable.

Notice that a τ-diagnoser for a set $\Sigma_{\mathrm{f}} = \{f_1, \ldots, f_m\}$ of faulty events can be split into m independent τ-diagnosers for each fault f_i. This is beneficial because i) it reduces the number of hypotheses that need to be considered from 2^m to $2 \times m$, and ii) each diagnoser can be tailored for its specific fault [Pencolé, 2004]. Importantly this means that we can assume that there is only one fault f.

The main contribution is this section is to show that it is possible to define two different CTL queries in order to build a τ-diagnoser. One can choose either query or both; in the latter case both queries can be answered in parallel so that as soon as one of them is solved, the other one can be terminated. We have the guarantee from the diagnosability property that at least one of them will be negative, and we consequently call this approach "diagnosis by contradiction."

The first query is similar to the one proposed earlier and can be simply written $\varphi_1 = EF\,(k \wedge \neg\{f\})$. Then $D_1(\sigma) = \emptyset$ iff φ_1 is positive is a τ-diagnoser.

The second query asks whether it is possible to find a run that matches the observed sequence and that does not exhibit a fault up to τ before the end of the observed sequence. To this end we redefine the faulty timed automaton TA^{F} as illustrated on Fig. 4. Essentially this automaton records the event f only if the clock y is below $|\sigma| - \tau$. This time the second query is $\varphi_2 = EF\,(k \wedge \{f\})$. Then $D_2(\sigma) = \{f\}$ iff φ_2 is positive (with the new faulty automaton) is a τ-diagnoser.

6 Related Works

The use of model checking for MBD was first proposed by Cordier and Largouët [2001]. They encode the observed sequence directly in the CTL query $EF(o_1 \wedge EF(o_2 \wedge \ldots))$, which makes it more cumbersome to use than our method.

Diagnosis of discrete event system is often performed by computing explicitly a machine that reads a flow of observed events and updates the current *belief state*, i.e., the set of states that the system could be in [Sampath *et al.*, 1995]. This method however is not very efficient (the machine is exponential in the number of system states) and cannot be applied for timed automata where the set of states is infinite [Tripakis, 2003].

Another approach consists in computing all runs that match the observed sequence, representing these runs compactly (e.g., with an automaton), and finally reasoning about this structure directly [Lamperti and Zanella, 2003; Pencolé and Cordier, 2005; Benveniste *et al.*, 2005; Su and Wonham, 2005; Kan John and Grastien, 2008]. Again this method is not quite applicable here because, e.g., in the timed automaton obtained by synchronous product of the model, the observation automaton, and the fault automaton, it is not obvious which discrete states are reachable from the start state. An additional procedure is required, which in our case is the purpose of the model checker.

Tripakis [2002] proposed to track the belief state with the use of data structures such as difference bound matrices (DBM) [Alur *et al.*, 1993]. While this approach is indeed a possible solution and is often used in model checkers, we moved away from it for the following three reasons. First it does not give the solver any flexibility as to how the problem should be solved. As we mentioned before, some of the queries are negative and this can be proved from a small subset of timed automata. In other situations, late observations can help discriminate early uncertainties [Pencolé and Cordier, 2005]. Model checkers can be clever in how they handle queries, e.g., with the use of counter-example guided abstraction refinement [Clarke *et al.*, 2000a]. Tracking the belief state goes against this type of approach. Second the approach by Tripakis is applicable as long as the observed sequence is certain. We did not expand further on this aspect, but it should be clear that the observation automaton could be modified to allow uncertainty on the observed sequence. This cannot be trivially done in the approach from Tripakis. Finally the flexibility of the observation automaton allows us to abstract this automaton, which can be used to identify which specific observed events are instrumental to finding the diagnosis [Christopher and Grastien, 2015].

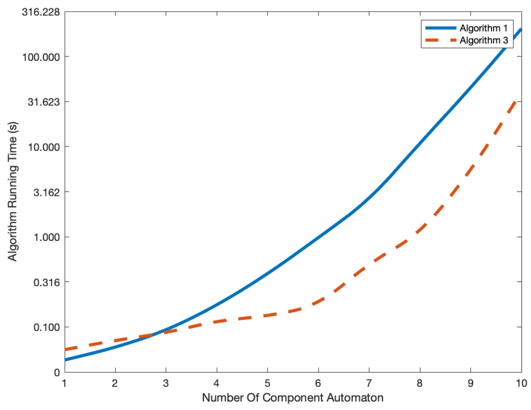

Figure 5: Performance comparison between Algo. 1 and Algo. 3.

7 Experiments

We use the experiment to compare the diagnosis efficiency between Algorithm 1 and the conflict-based approach of Algorithm 3. Our algorithms are implemented in Java. We used the model checker UPPAAL (www.uppaal.org/).

To this end we built a set of benchmarks that consists of $n + 2$ synchronised timed automata with n clocks for $n \in \{1, \ldots, 10\}$. These benchmarks model messages being carried from one component to the next in a timely manner; faults affect the duration of the transmission, with different delay for each component, which allows to narrow down the list of suspects.

In order to evaluate these two diagnosis algorithms, we built scenarios for the various systems. The results are summarised in Figure 5. We see that as soon as the number of faults becomes non-trivial (4 and more) the conflict-based approach becomes significantly more efficient (notice that the y-axis is logscale). Importantly our scenarios include several minimal diagnosis candidates of cardinality (number of faults) higher than one: these problems are not as trivial as the example we provided to illustrate the benefits of Algo. 3.

8 Conclusion

In this paper we show how to use model checkers in order to perform diagnosis of timed automata. We reduce deciding whether a set of hypotheses intersects the diagnosis to a reachability property in CTL, and show how to choose these sets of hypotheses so that the diagnosis can be deduced. We also show how to use conflicts to accelerate the process.

There are many issues that we ignored because of space restrictions but that are relevant. It should be easy to handle uncertainty on the observations [Jiang and Kumar, 2006; Lamperti and Zanella, 2003]. We also want to address the incremental aspect of diagnosis: as new observations are available, the diagnosis needs to be updated. There has been work on this issue [Su *et al.*, 2014; Bouziat *et al.*, 2019], and adapting these techniques to our approach is not obvious.

References

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

[Alur *et al.*, 1993] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[Benveniste *et al.*, 2005] A. Benveniste, S. Haar, É. Fabre, and C. Jard. Distributed monitoring of concurrent and asynchronous systems. *Journal of Discrete Event Dynamical Systems (JDEDS)*, 15(1):33–84, 2005.

[Bouziat *et al.*, 2019] V. Bouziat, X. Pucel, S. Roussel, and L. Travé-Massuyès. Single state trackability of discrete event systems. In *International Workshop on Principles of Diagnosis (DX-19)*, 2019.

[Christopher and Grastien, 2015] C. Christopher and A. Grastien. Formulating event-based critical observations in diagnostic problems. In *IEEE Conference on Decision and Control (CDC-15)*, pages 4462–4467, 2015.

[Clarke *et al.*, 2000a] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *International Conference on Computer-Aided Verification (CAV-00)*, pages 154–169, 2000.

[Clarke *et al.*, 2000b] E. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT Press, 2000.

[Cordier and Largouët, 2001] M.-O. Cordier and C. Largouët. Using model-checking techniques for diagnosing discrete-event systems. In *International Workshop on Principles of Diagnosis (DX-01)*, pages 39–46, 2001.

[Cordier *et al.*, 2004] M.-O. Cordier, P. Dague, F. Lévy, J. Montmain, M. Staroswiecki, and L. Travé-Massuyès. Conflicts versus analytical redundancy relations: a comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5):2163–2177, 2004.

[de Kleer and Williams, 1987] J. de Kleer and B. Williams. Diagnosing multiple faults. *Artificial Intelligence (AIJ)*, 32(1):97–130, 1987.

[Grastien *et al.*, 2012] A. Grastien, P. Haslum, and S. Thiébaux. Conflict-based diagnosis of discrete event systems: Theory and practice. In *Principles of Knowledge Representation and Reasoning*, 2012.

[He *et al.*, 2018] L. He, L. Ye, and P. Dague. SMT-based diagnosability analysis of real-time systems. *IFAC-PapersOnLine*, 2018.

[Jiang and Kumar, 2006] S. Jiang and R. Kumar. Diagnosis of dense-time systems using digital clocks. In *American Control Conference (ACC-06)*, 2006.

[Kan John and Grastien, 2008] P. Kan John and A. Grastien. Local consistency and junction tree for diagnosis of discrete-event systems. In *European Conference on Artificial Intelligence (ECAI-08)*, pages 209–213, 2008.

[Lamperti and Zanella, 2003] G. Lamperti and M. Zanella. *Diagnosis of active systems*. Kluwer Academic Publishers, 2003.

[Pencolé and Cordier, 2005] Y. Pencolé and M.-O. Cordier. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence (AIJ)*, 164(1–2):121–170, 2005.

[Pencolé, 2004] Y. Pencolé. Diagnosability analysis of distributed discrete event systems. In *European Conference on Artificial Intelligence (ECAI-04)*, pages 173–178, 2004.

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence (AIJ)*, 32(1):57–95, 1987.

[Sampath *et al.*, 1995] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control (TAC)*, 40(9):1555–1575, 1995.

[Slaney, 2014] J. Slaney. Set-theoretic duality: a fundamental feature of combinatorial optimisation. In *European Conference on Artificial Intelligence (ECAI-14)*, pages 843–848, 2014.

[Su and Wonham, 2005] R. Su and W. Wonham. Global and local consistencies in distributed fault diagnosis for discrete-event systems. *IEEE Transactions on Automatic Control (TAC)*, 50(12):1923–1935, 2005.

[Su *et al.*, 2014] X. Su, A. Grastien, and Y. Pencolé. Window-based diagnostic algorithms for discrete event systems: what information to remember. In *International Workshop on Principles of Diagnosis (DX-14)*, 2014.

[Tripakis, 2002] S. Tripakis. Fault diagnosis for timed automata. In *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT-02)*, pages 205–224, 2002.

[Tripakis, 2003] S. Tripakis. Folk theorems on the determinization and minimization of timed automata. In *International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS-03)*, pages 182–188, 2003.