Fault Diagnosis for Timed Automata[*]

Stavros Tripakis
VERIMAG (www-verimag.imag.fr)

1 Introduction

In this paper we study the problem of *fault diagnosis* in the context of *dense-time automata*. Our work is inspired from [SSL$^+$95, SSL$^+$96], who have studied the problem in the context of *discrete event systems* (DES) [RW87].

We stick to the terminology used in the above papers, although we find the term fault *detection*, rather than diagnosis, more appropriate. Indeed, the objective is to design an observer for a given plant, such that this observer can detects errors in the behavior of the plant. The method is model-based: it is assumed that the plant behaves according to a known model. The observer will be based on this model. Essentially, the observer will trace all possible states that the plant can be in, given the current sequence of observations. More details follow.

In the DES framework, the fault diagnosis problem is as follows. We are given the description of the behavior of a plant, in the form of a finite-state automaton. A behavior of the plant corresponds to a run of the automaton, that is, a sequence of events. An event is either *observable* or *unobservable*. One or more special unobservable events model *faults* that may occur during the operation of the plant. The objective is to design a *diagnoser*. The diagnoser is just a function which takes sequences of observable events and decides whether the original behavior contained a fault or not. The diagnoser should not announce a fault if no fault has occurred. The diagnoser should announce a fault at most n steps after the fault occurred. Once a fault is announced, the diagnoser cannot stop announcing it (i.e., on-line fault repairs are not modeled).

Not every plant is *diagnosable*. For example, a plant with two behaviors, a, f, b and a, u, b, is not diagnosable if f, u are unobservable and f is the fault. Indeed, the diagnoser, observing only a, b, has no way to know which of the two behaviors happened. On the other hand, a plant with behaviors a, f, b, c and a, u, b, d is diagnosable: after seeing c or d, the diagnoser can distinguish what happened.

Our motivation has been to extend the above framework to dense-time automata [AD94]. Such an extension is useful, since it permits us to model plants with timed behaviors, for example, "a followed by b within a delay of 7 time units". It also allows for diagnosers to base their decisions not only on the sequences of events observed, but also on the time delays between these events. That is, the diagnoser not only observes events, but can also measure the time elapsed between two successive events and, consequently, between any two events.

For example, consider the plant modeled by the timed automaton of Figure 1. The plant has two sets of behaviors: *faulty* behaviors (where f occurs) and non-faulty behaviors. If events a and b are observable, then the plant is diagnosable. Indeed, in all behaviors, a and b occur, in that order. In every faulty behavior, the delay between a and b is greater than 3 time units, while in every non-faulty behavior, the delay is at most 3. Thus, a diagnoser observing a and b, and measuring their interarrival delay can tell whether a fault occurred or not.

The contributions of this paper are as follows. First, we propose a notion of diagnosability for timed automata and give necessary and sufficient conditions for a timed automaton to be diagnosable. Second, we show how diagnosability can be algorithmically checked, by reducing the problem to checking whether a

[*]This is a modified version of a paper that appeared in FTRTFT'02 [Tri02]. This work has been partially supported by the European IST project "Next TTA" under project No IST-2001-32111.

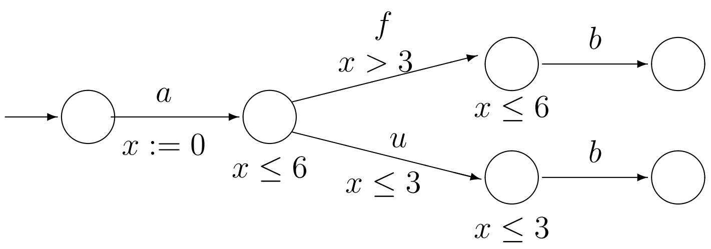


Figure 1: A diagnosable timed automaton.

certain timed automaton has a non-zeno run. It follows that diagnosability is in PSPACE. Third, we show how to effectively build a diagnoser for a diagnosable timed automaton. Although the set of observations is infinite (and in fact, non-enumerable), the diagnoser function is computable, using standard technology used in many timed automata verification tools.

2 Timed automata with faults and unobservable events

Let $\mathcal{X}$ be a finite set of variables taking values in the set of non-negative rational numbers, denoted $\mathcal{Q}$.[1] We call these variables *clocks*. A *valuation* on $\mathcal{X}$ is a function $v : \mathcal{X} \to \mathcal{Q}$ which assigns a value to each clock in $\mathcal{X}$. Given a valuation v and a *delay* $\delta \in \mathcal{Q}$, $v + \delta$ denotes the valuation v' such that for all $x \in \mathcal{X}$, $v'(x) = v(x) + \delta$. Given a valuation v and a subset of clocks $X \subseteq \mathcal{X}$, $v[X := 0]$ denotes the valuation v' such that for all $x \in X$, $v'(x) = 0$ and for all $y \in \mathcal{X} - X$, $v'(y) = v(y)$.

A *polyhedron* on $\mathcal{X}$ is a set of valuations which can be represented as a boolean expression with atomic constraints of the form $x \leq k$ or $x - y \leq k$, where $x, y \in \mathcal{X}$ and k is an integer constant. For example, $x = 0 \wedge y > 3$ is a polyhedron. By definition, polyhedra are closed by boolean operations $\wedge, \vee, \neg$, which correspond to set intersection, union and complementation. Polyhedra are also closed by existential quantification: for $x \in \mathcal{X}$, $\exists x \,.\, \zeta$ denotes the polyhedron $\{v \mid \exists v' \in \zeta, \forall y \in \mathcal{X}, y \neq x \Rightarrow v(y) = v'(y)\}$. For example, $\exists x \,.\, (x \leq 3 \wedge y \leq x)$ is the polyhedron $y \leq 3$. We use true to denote the polyhedron $\bigwedge_{x \in \mathcal{X}} x \geq 0$ and false to denote the empty polyhedron. We also use $\vec{0}$ to denote the singleton $\bigwedge_{x \in \mathcal{X}} x = 0$.

A *timed automaton* [AD94, HNSY94] is a tuple $A = (Q, \mathcal{X}, \Sigma, E, I)$, where:

- Q is a finite set of *discrete states*; $q^0 \in Q$ is the *initial* discrete state.
- $\mathcal{X}$ is a finite set of clocks.
- Σ is a finite set of *events*. Σ is the union of two disjoint sets $\Sigma = \Sigma_o \cup \Sigma_u$, and $f \in \Sigma_u$ is a distinguished event, called the *fault* event. An event in Σ_o is called *observable*, otherwise, it is called *unobservable*.
- E is a finite set of *transitions*. Each transition is a tuple $e = (q, q', a, \zeta, X)$, where $q, q' \in Q$, $a \in \Sigma$, ζ is a polyhedron on $\mathcal{X}$ and $X \subseteq \mathcal{X}$. We use $\mathsf{source}(e)$ to denote q, $\mathsf{dest}(e)$ for q', $\mathsf{event}(e)$ for a, $\mathsf{guard}(e)$ for ζ, and $\mathsf{reset}(e)$ for X. Given an event $a \in \Sigma$, $E(a)$ denotes the set of all transitions $e \in E$ such that $\mathsf{event}(e) = a$.
- I is the *invariant function* which associates with each discrete state $q \in Q$ a polyhedron on $\mathcal{X}$, $I(q)$. We require that $\vec{0} \in I(q^0)$.

A *state* of A is a pair $s = (q, v)$, where $q \in Q$ and v is a valuation on $\mathcal{X}$, such that $v \in I(q)$. We denote q by $\mathsf{discrete}(s)$. The *initial state* of A is $s^0 = (q^0, \vec{0})$. Each delay $\delta \in \mathcal{Q}$ defines a partial function on the states of A: if $s = (q, v)$ is a state of A, and $\forall \delta' \leq \delta$, $v + \delta' \in I(q)$, then $\delta(s) = (q, v + \delta)$, otherwise, $\delta(s)$ is

[1]We consider a rational time domain instead of a real time domain, because we are interested in implementing diagnosers that take as input delay values, and real numbers cannot be finitely represented in computers.

undefined. Each transition $e = (q, q', a, \zeta, X) \in E$ defines a partial function on the states of A: if $s = (q, v)$ is a state of A such that $v \in \zeta$ and $v[X := 0] \in I(q')$, then $e(s) = (q', v[X := 0])$, otherwise, $e(s)$ is undefined.

A *timed sequence* over a set of events Σ is a finite or infinite sequence $\gamma_1, \gamma_2, \cdots$, where each γ_i is either an event in Σ or a delay in $\mathcal{Q}$. We require that between any two events in ρ there is exactly one delay (possibly 0). For example, if a and b are events, $a, 0, b, 3, c$ and $a, 1, 1, 1, ...$ are valid timed sequences, while a, b and $a, 1, 2, b$ are not.

If ρ is a finite timed sequence, $\mathsf{time}(\rho)$ denotes the sum of all delays in ρ. If ρ is infinite, then $\mathsf{time}(\rho)$ denotes the limit of the sum (possibly ∞). We say that ρ is *non-zeno* if $\mathsf{time}(\rho) = \infty$. Note that a non-zeno timed sequence is necessarily infinite, although it might contain only a finite number of events.

We define a *projection* operator P as follows. Given a (finite or infinite) timed sequence ρ and a set of events $\Sigma' \subseteq \Sigma$, $P(\rho, \Sigma')$ is the timed sequence obtained by erasing from ρ all events in Σ' and summing the delays between successive events in the resulting sequence. For example, if $\rho = 1, a, 4, b, 1, c, 0, d, 3, e$, then $P(\rho, \{b, d\}) = 1, a, 5, c, 3, e$. Note that, in the definition of $P(\rho, \Sigma')$, Σ' is the set of events to be erased. Also notice that, $\mathsf{time}(\rho) = \mathsf{time}(P(\rho, \Sigma'))$, for any ρ and Σ'.

Given a state s of A, a *run of A starting at s* (or simply *a run of A*, if $s = s^0$) is a (finite or infinite) timed sequence $\rho = \gamma_1, \gamma_2, \cdots$, for which there exists a sequence of states $s_0, s_1, s_2, \cdots$, such that $s_0 = s$ and for each $i = 1, 2, ...$, if γ_i is a delay $\delta \in \mathcal{Q}$ then $s_i = \delta(s_{i-1})$, whereas if γ_i is an event $a \in \Sigma$, then $s_i = e(s_{i-1})$, for some $e \in E(a)$. If ρ is a finite run $\gamma_1, \gamma_2, \cdots, \gamma_n$ starting from s, we say that s_n is *reachable from s via ρ*. A finite run ρ defines a function on the states of A as follows. If s is a state of A, $\rho(s)$ is the set of all states reachable from s via ρ (note that $\rho(s)$ might be empty). The set of all states of A reachable from s^0 via some run is denoted R_A.

A is *well-timed* if for all $s \in R_A$, there is a non-zeno run of A starting at s.

Definition 1 (Faulty and δ-faulty runs) *A run $\rho = \gamma_1, \gamma_2, \cdots$ is called* faulty *if for some $i = 1, 2, ...$, $\gamma_i = f$. Let j be the smallest i such that $\gamma_i = f$, and let $\rho' = \gamma_j, \gamma_{j+1}, \cdots$. Given $\delta \in \mathcal{Q}$, if $\mathsf{time}(\rho') \geq \delta$, 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.

The following lemma states an important property of the model, which will be used in the sequel.

Lemma 2 *If for all $\Delta \in \mathsf{N}$, A has a Δ-faulty run, then A has a non-zeno faulty run.*

Proof Our proof relies on the *region graph* [AD94] of A, call it G. G is a finite quotient graph with respect to a *time-abstracting bisimulation* [TY01]. Each node of G (called a region) contains a set of bisimilar states of A. The edges of G correspond either to transitions of A or to symbolic passage of time. We refer the reader to timed-automata papers for more details on the region graph. What is important for our proof is that every run of A is inscribed in a path of G and, vice-versa, every path of G contains a set of runs of A.

Let R_f be the set of regions of G which are reachable by a faulty path. Note that for every $r \in R_f$, all successors of r are also in R_f. Let G_f be the restriction of G to R_f. We claim that G_f has a strongly-connected component (SCC) Λ, such that for every clock x, x is either reset and can grow strictly greater than zero in Λ, or remains unbounded in Λ: this implies the existence of a faulty non-zeno run [Alu91]. Suppose our claim is false, that is, for every SCC Λ in G_f, there is a clock x which remains upper bounded in Λ and is never reset or never grows above zero. In both cases, x never grows in Λ above some constant Γ_Λ (in the last case, $\Gamma_\Lambda = 0$). Then, for every run ρ inscribed in Λ, $\mathsf{time}(\rho) \leq \Gamma_\Lambda$. Since there is a finite number of SCCs in a finite graph, the time spent in any faulty run is bounded by some Γ (obtained as the maximum of Γ_Λ^x, plus the times spent in the finite paths linking the SCCs). But this contradicts the hypothesis. ■

3 Diagnosers and diagnosability

In this section we define diagnosability as the existence of diagnosers for a given plant. Informally, a diagnoser is a function that, given an observation, detects whether the original behavior of the plant was faulty or not. As we illustrated in the introduction, not all plants are diagnosable. We given necessary and sufficient conditions for diagnosability. Informally, a plant is diagnosable iff any pair of faulty/non-faulty behaviors can be distinguished by their projections to observable behaviors.

Let $\mathcal{FTS}_\Sigma$ denote the set of all finite timed sequences over Σ.

Definition 3 (Diagnosers) *Given a TA A over Σ with sets of observable/unobservable events $\Sigma_o, \Sigma_u \subseteq \Sigma$, and $\Delta \in \mathsf{N}$, a Δ-diagnoser for A is a function*

$$D : \mathcal{FTS}_{\Sigma_o} \to \{0,1\}$$

such that

1. *For any non-faulty finite run ρ of A, $D(P(\rho, \Sigma_u)) = 0$.*
2. *For any Δ-faulty finite run ρ of A, $D(P(\rho, \Sigma_u)) = 1$.*

If a Δ-diagnoser exists for A then we say that A is Δ-diagnosable. We say that A is diagnosable if there exists some $\Delta \in \mathsf{N}$ such that A is Δ-diagnosable.

Lemma 4 (Necessary and sufficient conditions for Δ-diagnosability) *Let A be a TA over Σ with sets of observable/unobservable events $\Sigma_o, \Sigma_u \subseteq \Sigma$. For any $\Delta \in \mathsf{N}$, A is Δ-diagnosable iff for any two finite runs ρ_1, ρ_2 of A, if ρ_1 is Δ-faulty and ρ_2 is non-faulty, then $P(\rho_1, \Sigma_u) \neq P(\rho_2, \Sigma_u)$.*

Proof Assume A is Δ-diagnosable and D is a diagnoser for A. Suppose ρ_1 is Δ-faulty and ρ_2 is non-faulty. Then, $D(P(\rho_1, \Sigma_u)) = 1$ and $D(P(\rho_1, \Sigma_u)) = 0$. Since D is a function, it must be that $P(\rho_1, \Sigma_u) \neq P(\rho_2, \Sigma_u)$.

In the opposite direction, assume the condition holds. We define function D as follows

$$D(\pi) = \begin{cases} 1, & \text{if there exists } \rho \text{ s.t. } P(\rho, \Sigma_u) = \pi \text{ and } \rho \text{ is } \Delta\text{-faulty}, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Then, by definition, if ρ is a Δ-faulty finite run of A, $D(P(\rho, \Sigma_u)) = 1$. Now, suppose ρ is a non-faulty finite run of A and let $\pi = P(\rho, \Sigma_u)$. If $D(\pi) = 1$ then there must exist some Δ-faulty finite run ρ' of A such that $\pi = P(\rho', \Sigma_u)$. But this would contradict the condition. Thus, $D(\pi) = 0$. ■

Example 5 *Assuming that events a and b are observable, f and u are unobservable and f is the fault, the timed automaton of Figure 1 is 3-diagnosable. On the other hand, the slightly modified automaton shown in Figure 2 is not diagnosable. Indeed, the two runs $a, 2.5, f, 0.1, b$ and $a, 2.5, u, 0.1, b$ have the same projection $a, 2.6, b$, but only the first one is faulty. Moreover, an arbitrary amount of time can elapse after b in both runs, and their projections will remain identical.*

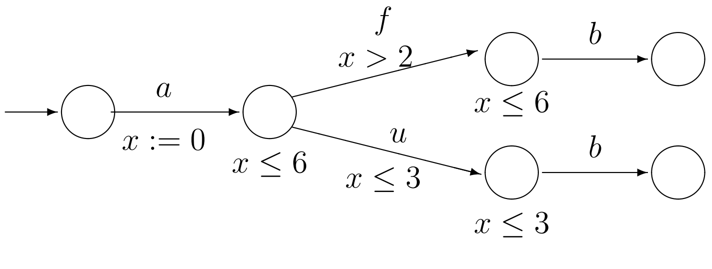


Figure 2: A non-diagnosable timed automaton.

We make some remarks about Definition 3.

For the sake of simplicity, we consider only one type of fault. The framework can be extended to k different faults, $f_1, ..., f_k$, in various ways. For instance, the diagnoser could be a function $D : \mathcal{FTS}_{\Sigma_o} \to \{0, 1, ..., k\}$

and we could require that it yields 0 for non-faulty runs and i if the first fault was f_i. We could also require that the diagnoser catches all faults. Then the range of D would have to be something like 2^k, that is, D yields a vector of k bits and the i-th bit is set iff f_i has occurred. We could require the same detection delay Δ for all faults or a separate delay Δ_i for each $i = 1, ..., k$. Lemma 4 can be extended to deal with k faults easily: it suffices to check the condition individually for each fault.

Another remark: we do not model on-line "repairs" of faults, that is, we assume that faults cannot be "undone". This means that, once a fault has occurred, we consider the behavior erroneous and we would like to detect the fault, no matter what the plant does afterwards.

A final remark: we define diagnosability with respect to a natural constant Δ, rather than, say, a real number. This allows us to speak of Δ_{min} in the lemma that follows. If Δ is taken to be real, we can find plants which are diagnosable for all $\Delta > 3$, say, but not for $\Delta = 3$. Assuming Δ natural also gives a simple enumerative procedure to find Δ_{min}, as we show in Proposition 10.

The following is a direct corollary of Lemma 4 and the fact that a $(\Delta + 1)$-faulty run is also Δ-faulty.

Lemma 6 *Let A be Δ-diagnosable. Then, for any $\Delta' > \Delta$, A is Δ'-diagnosable. Also, there exists Δ_{min} such that A is Δ_{min}-diagnosable and for all $\Delta' < \Delta_{min}$, A is not Δ'-diagnosable.*

4 Checking diagnosability

Checking diagnosability and building diagnosers are well-known problems for finite-state models. Diagnosability can be decided in polynomial time, whereas building a diagnoser relies on a *subset construction* and is exponential in the worst case [Tsi89].

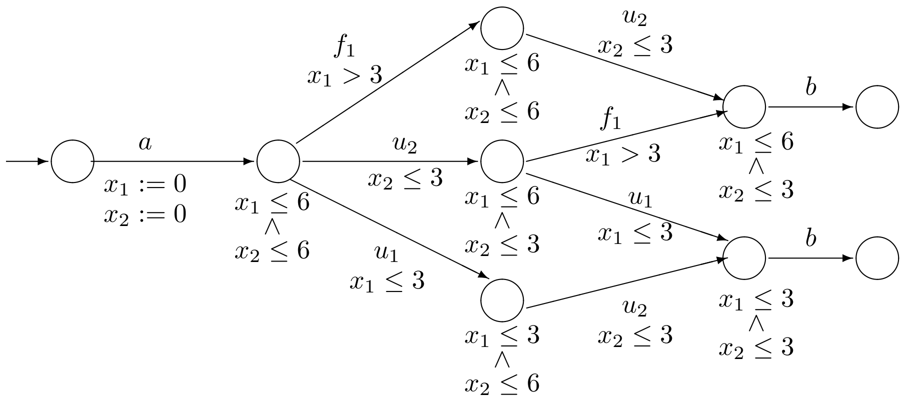
In this section, we show how to effectively check diagnosability in the dense-time case. We will assume that the plant is modeled as a well-timed automaton A. This is a reasonable assumption, since real time advances without upper bound and a faithful model must capture this fact.

Informally, the algorithm works as follows. We first build a special parallel product of A with itself. This product generates all pairs of runs of A that yield the same observations, yet one is faulty while the other one is not. Then, we will show that A is diagnosable iff the product cannot generate a non-zeno pair of runs. Indeed, this would mean that for any Δ we can find prefixes of the non-zeno pair which contradict the necessary condition for Δ-diagnosability.

The product, denoted $(A\|_{\Sigma_o}A)^{-f_2}$, is obtained in two phases. First, we build a product $A\|_{\Sigma_o}A$ as follows:

1. We make two "copies" of A, A_1 and A_2, by renaming unobservable events, discrete states and clocks of A:
   - Each discrete state q of A is renamed q_1 in A_1 and q_2 in A_2. The initial state q^0 is copied into q_1^0 and q_2^0.
   - Each clock x of A is renamed x_1 in A_1 and x_2 in A_2.
   - Each unobservable event u of A is renamed u_1 in A_1 and u_2 in A_2. Let Σ_u^1 and Σ_u^2 denote the corresponding sets of renamed unobservable events. Observable events are not renamed.
   - The transitions are copied and renamed accordingly. For example, $e = (q, q', u, x \leq 3, \{y\})$ becomes $e_1 = (q_1, q_1', u_1, x_1 \leq 3, \{y_1\})$ in A_1 (assuming the event u is unobservable, otherwise it would not be renamed).
2. $A\|_{\Sigma_o}A$ is obtained as the usual parallel composition of A_1 and A_2, where transitions of A_1 and A_2 labeled with the same (observable) event are forced to *synchronize*. For example, if $e_i = (q_i, q_i', a, \zeta_i, X_i)$ are transitions of A_i, for $i = 1, 2$, and a is an observable event, then $e = ((q_1, q_2), (q_1', q_2'), a, \zeta_1 \wedge \zeta_2, X_1 \cup X_2)$ is the synchronized transition of $A\|_{\Sigma_o}A$. All other transitions *interleave*. The invariant of a product state (q_1, q_2) is $I(q_1) \wedge I(q_2)$.

Now, let $(A\|_{\Sigma_o}A)^{-f_2}$ be the timed automaton obtained from $A\|_{\Sigma_o}A$ by removing all transitions labeled f_2 from the latter. An example is shown in Figure 3.



a
$x_1 := 0$
$x_2 := 0$
$x_1 \leq 6 \wedge x_2 \leq 6$
f_1
$x_1 > 3$
u_2
$x_2 \leq 3$
u_1
$x_1 \leq 3$
$x_1 \leq 6 \wedge x_2 \leq 6$
$x_1 \leq 6 \wedge x_2 \leq 3$
$x_1 \leq 3 \wedge x_2 \leq 6$
u_2
$x_2 \leq 3$
f_1
$x_1 > 3$
u_1
$x_1 \leq 3$
u_2
$x_2 \leq 3$
$x_1 \leq 6 \wedge x_2 \leq 3$
$x_1 \leq 3 \wedge x_2 \leq 3$
b
b

Figure 3: The product $(A\|_{\Sigma_o}A)^{-f_2}$ for the timed automaton of Figure 1.

The intuition is that every run of $(A\|_{\Sigma_o}A)^{-f_2}$ corresponds to two runs of A which yield the same observation, that is, the same projection to observable events. We obtain this property by synchronizing the two copies in all observable events. (Note that time advances synchronously in both copies.)

To prove this, we need some notation. Let ρ be a run of $(A\|_{\Sigma_o}A)^{-f_2}$. ρ is called faulty if f_1 appears in it. We denote by ρ^1 (resp., ρ^2) the timed sequence obtained by taking the projection $P(\rho, \Sigma_u^2)$ (resp., $P(\rho, \Sigma_u^1)$) and then renaming each event $u_1 \in \Sigma_u^1$ (resp., $u_2 \in \Sigma_u^2$) back into u. That is, ρ^1 and ρ^2 are timed sequences over Σ. For example, if $\rho = a, 1, u_2, 3, u_1$, then $\rho^1 = a, 4, u$ and $\rho^2 = a, 1, u, 3$.

Lemma 7 *ρ is a run of $(A\|_{\Sigma_o}A)^{-f_2}$ iff ρ^1 and ρ^2 are runs of A, ρ^2 is not faulty and $P(\rho^1, \Sigma_u) = P(\rho^2, \Sigma_u)$. For such ρ, ρ^1, ρ^2, the following also hold:*

1. *ρ is faulty iff ρ^1 is faulty.*
2. *$\mathsf{time}(\rho) = \mathsf{time}(\rho^1) = \mathsf{time}(\rho^2)$.*

Proposition 8 (Checking diagnosability) *A is diagnosable iff every faulty run of $(A\|_{\Sigma_o}A)^{-f_2}$ is zeno.*

Proof Let ρ be a non-zeno faulty run of $(A\|_{\Sigma_o}A)^{-f_2}$. Pick some $\Delta \in \mathsf{N}$. Let ρ_Δ be a Δ-faulty finite prefix of ρ: since ρ is non-zeno and faulty, such a ρ_Δ exists and it is clearly a run of $(A\|_{\Sigma_o}A)^{-f_2}$. Thus, by Lemma 7, ρ_Δ^1 and ρ_Δ^2 are both runs of A, ρ_Δ^2 is not faulty, and $P(\rho_\Delta^1, \Sigma_u) = P(\rho_\Delta^2, \Sigma_u)$. Moreover, ρ_Δ^1 is Δ-faulty: this is because the time elapsing after f_1 in ρ_Δ is equal to the time elapsing after f in ρ_Δ^1. By Lemma 4, A is not Δ-diagnosable. Since such runs can be found for any Δ, A is not diagnosable.

In the opposite direction, suppose A is not diagnosable. By Lemma 4, this means that for any $\Delta \in \mathsf{N}$, there exist two finite runs ρ_Δ^1 and ρ_Δ^2 of A, such that ρ_Δ^1 is Δ-faulty, ρ_Δ^2 is non-faulty and $P(\rho_\Delta^1, \Sigma_u) = P(\rho_\Delta^2, \Sigma_u)$. Therefore, by Lemma 7, for any $\Delta \in \mathsf{N}$, there exists a run ρ_Δ of $(A\|_{\Sigma_o}A)^{-f_2}$ such that ρ_Δ is Δ-faulty. By Lemma 2, A has a non-zeno faulty run. ■

From Proposition 8, it follows that checking diagnosability for timed automata is decidable. Indeed, $(A\|_{\Sigma_o}A)^{-f_2}$ can be automatically generated from A using simple copying and renaming operations, and the standard syntactic parallel composition of timed automata. Finding non-zeno runs of a timed automaton was first shown to be decidable (PSPACE-complete) in [Alu91] using the region graph construction. Since the size of $(A\|_{\Sigma_o}A)^{-f_2}$ is polynomial in the size of A, it follows that checking diagnosability is also in PSPACE. Timed automata reachability can be reduced to diagnosability.[2] which implies that checking diagnosability is also PSPACE-hard.

[2]Given timed automaton A and target discrete state q_f, add two transitions from q_f, one labeled by a fault event f and another by an unobservable event u. Call the new automaton A'. All other events of A' (the original events of A) are observable. Then, it can be seen that if q_f is reachable in A then A' is not diagnosable, whereas if q_f is not reachable in A then A' is diagnosable (A' never performs f).

Proposition 9 (Complexity) *Diagnosability for timed-automata is PSPACE-complete.*

In practice, non-zeno runs can be found more efficiently, using the algorithms proposed in [BTY97]. These algorithms work on the *simulation graph*, which is a much coarser graph than the region graph, and can be constructed on-the-fly using, for instance, a *depth-first search.* The above algorithms have been implemented in the model-checking tool Kronos [Tri98, BDM+98].

Checking whether A is diagnosable is not enough. We would also like to find a Δ such that A is Δ-diagnosable. Even better, we would like to find the minimum such Δ, Δ_{min}, which exists, as we know from Lemma 6. We will find Δ_{min} by trying out different values for Δ, using a binary search. This is based on the fact that, for a given Δ, we can effectively check whether A is Δ-diagnosable, using the construction explained below.

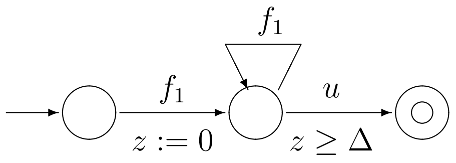


Figure 4: Observer automaton $\mathsf{Obs}(\Delta)$.

Consider the *observer* automaton shown in Figure 4. The automaton is parameterized by the constant $\Delta \in \mathsf{N}$, that is, for each given Δ, there is a different automaton, which will be denoted $\mathsf{Obs}(\Delta)$. The clock z of $\mathsf{Obs}(\Delta)$ is a new clock, different from all clocks in A or $(A\|_{\Sigma_o}A)^{-f_2}$. The event u is a new unobservable event, different from all events in A or $(A\|_{\Sigma_o}A)^{-f_2}$. The rightmost discrete state of $\mathsf{Obs}(\Delta)$ (drawn with two concentric circles) is its *accepting* state. Let $(A\|_{\Sigma_o}A)^{-f_2}\|_{f_1}\mathsf{Obs}(\Delta)$ be the parallel product of $(A\|_{\Sigma_o}A)^{-f_2}$ and $\mathsf{Obs}(\Delta)$, where the two automata synchronize only on the transitions labeled f_1. Then, we have the following result.

Proposition 10 (Checking Δ-diagnosability) *For any timed automaton A and any $\Delta \in N$, A is Δ-diagnosable iff the accepting state of $(A\|_{\Sigma_o}A)^{-f_2}\|_{f_1}\mathsf{Obs}(\Delta)$ is unreachable.*

If we know that a given automaton A is diagnosable, then we can use Proposition 10 in the following way. We check repeatedly, for $\Delta = 0, 1, 2, ...$, whether the accepting state of $(A\|_{\Sigma_o}A)^{-f_2}\|_{f_1}\mathsf{Obs}(\Delta)$ is reachable. Since A is diagnosable, reachability will eventually fail. This will happen for the first time when $\Delta = \Delta_{min}$.

The above method is simple, but not very efficient (especially when Δ_{min} is large), since it requires $\Delta_{min}+1$ reachability tests. An alternative way is to use the well-known *binary search* technique, which involves $O(\log \Delta_{min})$ reachability tests. The binary search starts by performing the reachability test repeatedly for $\Delta = 0, 1, 2, 4, 8, ...$, until the first time the test fails. Assume this happens for $\Delta = 2^k$. Then, we know that A is 2^k-diagnosable but not 2^{k-1}-diagnosable, so, Δ_{min} must lie in the interval $[2^{k-1} + 1, 2^k]$. We search this interval by "splitting" it in two, $[2^{k-1} + 1, M]$ and $[M, 2^k]$, checking reachability for the middle value M, and repeating the procedure recursively, for $[2^{k-1} + 1, M]$, if the test fails, and for $[M, 2^k]$, if it succeeds.

5 Building diagnosers

In this section we show how to effectively construct a diagnoser. Since the domain of a diagnoser D is infinite (the set of all observations, $\mathcal{FTS}_{\Sigma_o}$) we must find a way to finitely represent a diagnoser. The idea is to represent D as an algorithm, which takes as input an observation π and produces the diagnosis decision $D(\pi)$. That is, although the domain of D is infinite, D itself is computable.

In the rest of the section, we fix a timed automaton A which is Δ-diagnosable. Moreover, we will assume that the set of discrete states Q of A is partitioned in two disjoint sets: $Q = Q_f \cup (Q - Q_f)$, such that, for every run ρ of A, $\mathsf{discrete}(\rho(s^0)) \in Q_f$ iff ρ is faulty. In other words, once a fault occurs, A moves to Q_f

and never exits this set of discrete states, and while no fault occurs, A moves inside $Q - Q_f$. It is easy to transform any automaton to an automaton satisfying the above condition, possibly by having to duplicate some discrete states and transitions (the transformed automaton will have at most twice as many discrete states as the original automaton). An example of such a transformation is shown in Figure 5. The motivation for the transformation is to reduce the fault detection problem into a state estimation problem: a fault has been detected once the diagnoser is certain that the plant is in some state with discrete part in Q_f.

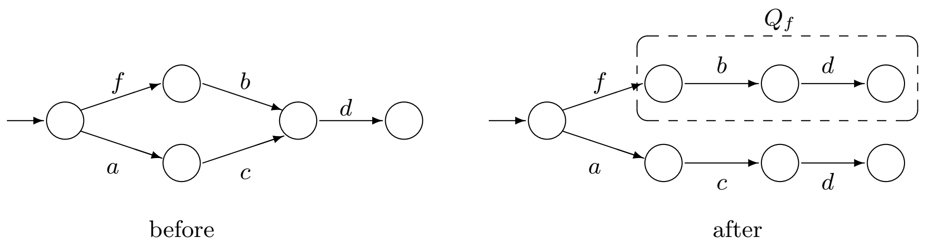


Figure 5: Transforming an automaton.

5.1 Defining the diagnoser

Recall that R_A is the set of reachable states of A. Also, for a given event $a \in \Sigma$, $E(a)$ denotes the set of transitions of A labeled by a. Given a set of events $\Sigma' \subseteq \Sigma$, let $\mathsf{Runs}(A, \Sigma')$ be the set of finite runs of A containing only events in Σ'.

Let $\mathsf{R_o} : 2^{R_A} \times \Sigma_o \to 2^{R_A}$ be the function defined as follows:

$$\mathsf{R_o}(S, a) = \{e(s) \mid s \in S, e \in E(a)\} \tag{2}$$

That is, $\mathsf{R_o}(S, a)$ contains all states that can be reached from a state in S by taking a discrete transition labeled a.

Let $\mathsf{R_u} : 2^{R_A} \times \mathcal{Q} \to 2^{R_A}$ be the function defined as follows:

$$\mathsf{R_u}(S, \delta) = \{\rho(s) \mid s \in S, \rho \in \mathsf{Runs}(A, \Sigma_u), \mathsf{time}(\rho) = \delta\}. \tag{3}$$

That is, $\mathsf{R_u}(S, \delta)$ contains all states that can be reached from a state in S in exactly δ time units by following only unobservable transitions.

Let s^0 be the initial state of A and $S_0 = \mathsf{R_u}(\{s^0\}, 0)$. That is, S_0 contains all states that can be reached from the initial state of A in zero time by following only unobservable transitions.

Now, let $F_D : 2^{R_A} \times \mathcal{FTS}_{\Sigma_o} \to 2^{R_A}$ be the function defined recursively as follows:

$$\begin{aligned} F_D(S, \epsilon) &= S, & (4)\\ F_D(S, a\pi) &= F_D(\mathsf{R_o}(S, a), \pi), & (5)\\ F_D(S, \delta\pi) &= F_D(\mathsf{R_u}(S, \delta), \pi). & (6) \end{aligned}$$

The following lemma states that the function $F_D(S_0, \cdot) : \mathcal{FTS}_{\Sigma_o} \to 2^{R_A}$ acts as a *state estimator* for A.

Lemma 11 *(1) Let ρ be a finite run of A and $\pi = P(\rho, \Sigma_u)$. Then, if $s \in \rho(s^0)$ then $s \in F_D(S_0, \pi)$.*
(2) If $s \in F_D(S_0, \pi)$ then there exists a finite run ρ of A such that $s \in \rho(s^0)$ and $\pi = P(\rho, \Sigma_u)$.

Finally, let $H_D : 2^{R_A} \to \{0, 1\}$ the function defined as follows:

$$H_D(S) = \begin{cases} 1, & \text{if } \forall s \in S, \mathsf{discrete}(s) \in Q_f \\ 0, & \text{otherwise.} \end{cases} \tag{7}$$

Then, the diagnoser $D : \mathcal{FTS}_{\Sigma_o} \to \{0, 1\}$ is defined as follows:

$$D(\pi) = H_D(\, F_D(S_0, \pi)\,). \tag{8}$$

Proposition 12 *If A is Δ-diagnosable then D defined as above is a Δ-diagnoser for A.*

Proof Let ρ be a finite run of A and let $\pi = P(\rho, \Sigma_u)$. Also let $s \in \rho(s^0)$ ($\rho(s^0)$ is non-empty because ρ is a run of A). By part (1) of Lemma 11, $s \in F_D(S_0, \pi)$.

Assume first that ρ is non-faulty. From the assumption about the structure of A, it must be that $\mathsf{discrete}(s) \notin Q_f$. Thus, by definition of H_D, $H_D(F_D(S_0, \pi)) = D(\pi) = 0$.

Now, assume that ρ is Δ-faulty. From the assumption about the structure of A, it must be that $\mathsf{discrete}(s) \in Q_f$. Suppose that $H_D(F_D(S_0, \pi)) = 0$. This means that there exists $s' \in F_D(S_0, \pi)$ such that $\mathsf{discrete}(s') \notin Q_f$. By part (2) of Lemma 11, $s' \in \rho'(s^0)$, where ρ' is a finite run of A such that $\pi = P(\rho', \Sigma_u)$. From the assumption about the structure of A, ρ' must be non-faulty. But this contradicts the fact that A is Δ-diagnosable. Thus, $H_D(F_D(S_0, \pi)) = D(\pi) = 1$. ■

5.2 Diagnoser implementation and run-time considerations

In this section, we show how a diagnoser defined as above can be effectively computed. In fact, we will use technology not much different from that used in timed-automata model-checkers such as Kronos [BDM+98] or Uppaal [Upp].

To be able to compute $D(\pi)$, given a finite observation π, we need to be able to compute functions $H_D, F_D, \mathsf{R_o}, \mathsf{R_u}$. These functions take as inputs discrete events in Σ_o, delays in $\mathcal{Q}$, and subsets of states of A in 2^{R_A}. Thus, we first need to be able to represent these inputs effectively. This can be easily done for discrete events, since Σ_o is finite. It can also be done for delays, although their granularity will be restricted by the numerical accuracy of our computer.

states can be represented using finitary data structures and how the decision and transition functions can be effectively computed on these structures.

A set $S \subseteq R_A$ can sometimes be represented as a list $[(q_1, \zeta_1), ..., (q_k, \zeta_k)]$, where $q_i \in Q$ and ζ_i is a polyhedron on $\mathcal{X}$, the set of clocks of A. Note that not all subsets of R_A can be represented this way. However, it can be shown that we need not exit the class of subsets of R_A that admit indeed such a representation. A polyhedron ζ_i can be effectively represented using well-known data structures called *difference bound matrices* (DBMs) [Dil89]. Set-theoretic operations on such polyhedra, such as union, intersection, test for emptiness, and so on, can be conducted on the corresponding DBMs. The initial state s^0 of A can be represented by the list $[(q_0, \bigwedge_{x \in X} x = 0)]$.

The decision function $H_D(S)$ can be easily computed by scanning the list $[(q_1, \zeta_1), ..., (q_k, \zeta_k)]$ representing S: if the list contains some pair (q_i, ζ_i) such that $q_i \notin Q_f$, then $H_D(S) = 0$, otherwise $H_D(S) = 1$.

The function $\mathsf{R_o}(S, a)$ can be computed as follows. If S is represented by $[(q_1, \zeta_1), ..., (q_k, \zeta_k)]$, start with a new empty list for $\mathsf{R_o}(S, a)$. Then, for each (q_i, ζ_i) and for each $e \in E(a)$ such that $\mathsf{source}(e) = q_i$, if $\zeta_i \cap \mathsf{guard}(e) \neq \emptyset$, then $(\mathsf{dest}(e), \zeta')$ is added to the new list, where ζ' is the polyhedron

$$\Big((\exists x \in \mathsf{reset}(e) \,.\, \zeta_i) \cap (\bigwedge_{x \in \mathsf{reset}(e)} x = 0) \Big) \cap I(\mathsf{dest}(e)).$$

The function $\mathsf{R_u}(S, \delta)$ can be computed using a *reachability* procedure, similar to standard procedures using in the timed automata verification tools mentioned above. There are two differences between the standard procedure and the one for computing $\mathsf{R_u}$. First, for $\mathsf{R_u}$, reachability is restricted only to unobservable transitions of A (i.e., transitions labeled with events in Σ_u). Second, standard reachability computes the set of states reachable at any time, whereas reachability for $\mathsf{R_u}$ must compute the set of states reachable in *exactly* δ time units. This condition can be satisfied as follows.

First, compute the set of states reachable from S in *at most δ time units*. Call this set $S_{\leq \delta}$. $S_{\leq \delta}$ can be computed by adding to A a new clock z initialized to 0 and exploring during reachability only the states satisfying $z \leq \delta$. Once $S_{\leq \delta}$ is computed, take the intersection $S_{\leq \delta} \cap (z = \delta)$: this set contains all states reachable from S in exactly δ time units. Finally, the clock z is eliminated by existential quantification:

$$\mathsf{R_u}(S, \delta) \quad = \quad \exists z \,.\, \Big(S_{\leq \delta} \cap (z = \delta) \Big). \tag{9}$$

```
initialize diagnoser state $S = S_0$ ;
set a timer $T = 0$ and an alarm for $T = \mathsf{TO}$ ;
loop
   if $(H_D(S) = 1)$ then
      announce fault ;
   end if ;
   await event or alarm interrupt ;
   if (event $a$ interrupt) then
      read the current value of $T$, call it $\delta$ ;
      set $S$ to $\mathsf{R_o}(\mathsf{R_u}(S, \delta), a)$ ;
   else
      set $S$ to $\mathsf{R_u}(S, \mathsf{TO})$ ;
   end if ;
   set timer $T = 0$ and an alarm for $T = \mathsf{TO}$ ;
end loop.
```

Figure 6: Diagnoser implementation loop.

Figure 6 shows how the implementation of a diagnoser might look like in pseudo-code. Notice the use of a timeout parameter TO. The timeout "wakes up" the diagnoser after TO time, assuming no observable event has occurred meanwhile. Thus, the diagnoser can detect errors even if no observable event occurs for a long time.

An implementation like the one shown in Figure 6 requires an execution platform which provides event interrupts and one timer which can be set to 0, read, and send an alarm when it reaches a specified value. Such an implementation will function correctly, provided the loop can be executed sufficiently fast. In practice, this means that the maximum time to compute the loop should not be greater than the minimum delay between two observable events (nor greater than TO, of course). This requirement is similar to the *synchrony hypothesis*, which implies the correct execution of programs written in *synchronous languages* such as Esterel [Est] or Lustre [HCRP91].

6 Related work and discussion

Fault diagnosis has been studied by different communities and in different contexts, e.g., see [RM92, SSL$^+$95, SM96, BLPZ99, BBBSV02], and citations therein. We restrict our discussion to work closely related to timed systems.

[CP97, ZKW99] study fault-diagnosis on a discrete-time model, called *timed discrete-event systems* (TDES). In TDES, time passing is modeled by a special (observable) event "tick" and the problem of diagnosis can be easily reduced to the untimed case and solved using untimed techniques. Discretization of time is also used in [SCM93], to reduce the problem into a finite-state diagnosis problem.

[HC94] use a timed automaton model without clocks, but where time intervals are associated with discrete states. They propose *template monitoring* as a technique for distributed fault diagnosis, where templates are sets of constraints on the occurrence times of events.

Fault diagnosis is closely related to observation and state estimation problems. Such problems are considered in the context of hybrid automata in [BBBSV02, NB02]. These methods rely on an observable part of (or function of) the plant's continuous variables. Based on the observable continuous variables (and possibly discrete observations as well), the dynamics of the unobservable variables must be determined. This approach cannot be used to solve our problem, because we assume that *no* clock of the timed automaton is directly observable. Instead, the diagnoser must infer the values of clocks based only on the events it observes. This is a reasonable assumption, since the plant model is often an abstraction of a physical process, which has no clocks anyway.

Fault-diagnosis is also related to the *controller-synthesis* problem, introduced for discrete-event systems in [RW87]. The problem has been studied for timed and hybrid models as well (e.g., see [WTH91, BW94, CG96, AMPS98, RO96, WT97, TLS98]). Some of these works are restricted to a discrete-time framework, for example [BW94, RO96]. The rest make a major assumption, namely, that the state of the plant (including the values of all clocks) is *fully observable*. This is unrealistic, except for the case where the plant is deterministic and all its events are observable.[3]

[WT97] discusses how partial observability of plant states can be taken into account, by assuming the existence of a function $vis(\cdot)$ from the state space of the plant to a domain of possible observations: when the plant is at state s, the controller observes $vis(s)$. Then, [WT97] shows how to synthesize *memoryless* controllers in the above framework. The controllers are memoryless in the sense that their decision depends only on the current observation and not past ones. This is why the algorithm is incomplete: it might fail to synthesize a controller, even though one exists. Another drawback is that the function $vis(\cdot)$ is not always easy to come up with, for example, when starting with an observation framework based on events, as we do here.

Acknowledgements. I would like to thank Eugene Asarin, Oded Maler and Peter Niebert.

References

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[Alu91] R. Alur. *Techniques for Automatic Verification of Real-Time Systems.* PhD thesis, Department of Computer Science, Stanford University, 1991.

[AMPS98] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *Proc. IFAC Symposium on System Structure and Control.* Elsevier, 1998.

[BBBSV02] A. Balluchi, L. Benvenuti, M. Di Benedetto, and A. Sangiovanni-Vincentelli. Design of observers for hybrid systems. In *Hybrid Systems: Computation and Control*, 2002.

[BDM+98] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: a model-checking tool for real-time systems. In *10th Conference on Computer-Aided Verification (CAV'98)*, volume 1427 of *LNCS*. Springer, 1998.

[BLPZ99] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella. Diagnosis of large active systems. *Artificial Intelligence*, 110, 1999.

[BTY97] A. Bouajjani, S. Tripakis, and S. Yovine. On-the-fly symbolic model checking for real-time systems. In *18th IEEE Real-Time Systems Symposium (RTSS'97)*, pages 25–34. IEEE, December 1997.

[BW94] B.A. Brandin and W. Wonham. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39(2), 1994.

[CG96] D.D. Cofer and V.K. Garg. On controlling timed discrete event systems. In *Hybrid Systems III: Verification and Control.* LNCS 1066, Springer-Verlag, 1996.

[CP97] Yi-Liang Chen and Gregory Provan. Modeling and diagnosis of timed discrete event systems – a factory automation example. In *ACC*, 1997.

[3]In this case, the controller can replicate the clocks of the plant and reset them whenever it sees the corresponding observable event.

[Dil89] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 197–212. Springer–Verlag, 1989.

[Est] Esterel web-site: `http://www-sop.inria.fr/esterel.org/`.

[HC94] L.E. Holloway and S. Chand. Time templates for discrete event fault monitoring in manufacturing systems. In *Proc. of the 1994 American Control Conference*, 1994.

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language Lustre. *Proceedings of the IEEE*, 79(9), September 1991.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[NB02] S. Narasimhan and G. Biswas. An approach to model-based diagnosis of hybrid systems. In *Hybrid Systems: Computation and Control*, 2002.

[RM92] Amit Kumar Ray and R. B. Misra. Real-time fault diagnosis - using occupancy grids and neural network techniques. In *Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE.* LNCS 604, Springer-Verlag, 1992.

[RO96] J. Raisch and S. O'Young. A DES approach to control of hybrid dynamical systems. In *Hybrid Systems III: Verification and Control.* LNCS 1066, Springer-Verlag, 1996.

[RW87] P. Ramadge and W. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1), January 1987.

[SCM93] J. Sztipanovits, R. Carnes, and A. Misra. Finite state temporal automata modeling for fault diagnosis. In *9th AIAA Conference on Computing in Aerospace*, 1993.

[SM96] J. Sztipanovits and A. Misra. Diagnosis of discrete event systems using ordered binary decision diagrams. In *7th Intl. Workshop on Principles of Diagnosis*, 1996.

[SSL$^+$95] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 40(9), September 1995.

[SSL$^+$96] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Failure diagnosis using discrete event models. *IEEE Transactions on Control Systems Technology*, 4(2), March 1996.

[TLS98] C. Tomlin, J. Lygeros, and S. Sastry. Synthesizing controllers for nonlinear hybrid systems. In *Hybrid Systems: Computation and Control.* LNCS 1386, Springer-Verlag, 1998.

[Tri98] S. Tripakis. *The formal analysis of timed systems in practice.* PhD thesis, Université Joseph Fourrier de Grenoble, 1998.

[Tri02] S. Tripakis. Fault diagnosis for timed automata. In *Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT'02)*, volume 2469 of *LNCS*. Springer, 2002.

[Tsi89] J.N. Tsitsiklis. On the control of discrete event dynamical systems. *Mathematics of Control, Signals and Systems*, 2(2), 1989.

[TY01] S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, January 2001.

[Upp] Uppaal web-site: `www.docs.uu.se/docs/rtmv/uppaal/`.

[WT97] H. Wong-Toi. The synthesis of controllers for linear hybrid automata. In *Proc. of IEEE Conference on Decision and Control*, 1997.

[WTH91] H. Wong-Toi and G. Hoffmann. The control of dense real-time discrete event systems. In *Proc. of the 30th IEEE Conference on Decision and Control*, 1991.

[ZKW99] S. Hashtrudi Zad, R. Kwong, and W. Wonham. Fault diagnosis in finite-state automata and timed discrete-event systems. In *38th IEEE Conference on Decision and Control*, 1999.