Fault Diagnosis for Timed Automata

Stavros Tripakis

VERIMAG
Centre Equation, 2, avenue de Vignate, 38610 Gières, France
tripakis@imag.fr

Abstract. We study the problem of fault-diagnosis in the context of dense-time automata. The problem is, given the model of a plant as a timed automaton with a set of observable events and a set of unobservable events, including a special event modeling faults, to construct a deterministic machine, the diagnoser, which reacts to observable events and time delays, and announces a fault within a delay of at most Δ time units after the fault occurred. We define what it means for a timed automaton to be diagnosable, and provide algorithms to check diagnosability. The algorithms are based on standard reachability analyses in search of accepting states or non-zeno runs. We also show how to construct a diagnoser for a diagnosable timed automaton, and how the diagnoser can be implemented using data structures and algorithms similar to those used in most timed-automata verification tools.

Keywords: Fault diagnosis, Partial observability, Timed Automata.

1 Introduction

In this paper we study the problem of *fault diagnosis* in the context of *dense-time automata*. Our work is inspired from [21,22], who have studied the problem in the context of *discrete event systems* (DES) [19].

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 a deterministic machine which reacts to observable events by changing state instantaneously. The requirements are as follows. If the plant performs a fault event, the diagnoser should detect it after at most n steps (i.e., moves of the plant), where n has a known upper bound. Detection means, for instance, that the diagnoser enters a special state which announces that the fault has occurred. Another obvious requirement is that the diagnoser does not create any false alarms, that is, whenever it announces a fault, the fault has indeed occurred. Finally, once a fault is announced, the diagnoser cannot stop announcing it (i.e., on-line fault repairs are not modeled).

W. Damm and E.-R. Olderog (Eds.): FTRTFT 2002, LNCS 2469, pp. 205–221, 2002.

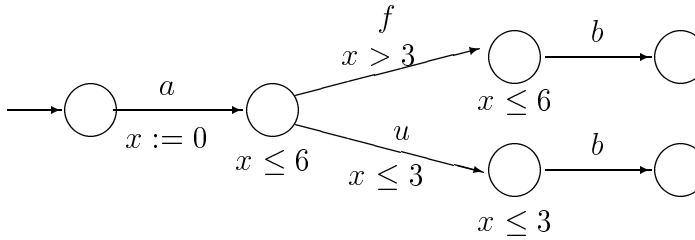

Fig. 1. A diagnosable timed automaton.

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 whether a fault actually occurred or not. 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 [3]. 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. This notion, called Δ-diagnosability, ensures that a fault can always be detected at most Δ time units after the time it occurs. Second, we provide an algorithm to check whether a given automaton A is diagnosable. The algorithm is based on a reachability analysis in search of *non-zeno runs* in a special product automaton of A with itself. Third, we provide an algorithm to find the minimum Δ such that A is Δ-diagnosable (assuming that it is already known that A is diagnosable). The algorithm conducts a *binary search* on the constant Δ, each time performing a reachability analysis of the above product, composed with an observer automaton with one clock.

Finally, we show how to build a diagnoser for A. The diagnoser works as a *state estimator* for A, that is, a state of the diagnoser is a set of all possible states that A could be in, according to what has been observed so far. The diagnoser changes state each time it observes an event a or a delay δ, and it is guaran-

teed not to produce any false alarms and to announce a fault at most Δ time units after the fault occurs. We also show how a diagnoser can be implemented effectively using finitary data structures to represent its current state, and how the transition function of the diagnoser can be effectively computed on these structures.

The rest of the paper is organized as follows. In Section 2, we present our model. In Section 3, we introduce the notion of diagnosability and show how it can be algorithmically checked. In Section 4, we define diagnosers and show how they can be implemented. Related work is discussed in Section 5.

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 reals, denoted R. We call these variables *clocks*. A *valuation* on $\mathcal{X}$ is a function $v : \mathcal{X} \to \mathsf{R}$ which assigns a value to each clock in $\mathcal{X}$. Given a valuation v and a *delay* $\delta \in \mathsf{R}$, $v + \delta$ 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 $\mathbf{0}$ to denote the singleton $\bigwedge_{x \in \mathcal{X}} x = 0$.

A *timed automaton* [3,15] 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[1]. 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 $\mathbf{0} \in I(q^0)$.

[1] For simplicity, we assume a single type of fault. The definitions and results generalize directly to more than one fault types. In Section 3 we discuss how this can be done.

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 $s^0 = (q^0, \mathbf{0})$. Each delay $\delta \in \mathsf{R}$ 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 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 R. 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. Let $\mathcal{TS}_\Sigma$ denote the set of all timed sequences over Σ.

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 $\delta \in \mathsf{R}$ 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.

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 $\delta \in \mathsf{R}$, 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 1. *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* [3] of A, call it G. G is a finite quotient graph with respect to a *time-abstracting bisimulation* [28]. 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 [4]. 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^x_Λ, plus the times spent in the finite paths linking the SCCs). But this contradicts the hypothesis. □

3 Diagnosability

We assume that the behavior of the plant to be diagnosed can be modeled as a timed automaton A. We also assume that A is well-timed. This is a reasonable assumption, since, in real plants, time elapses without upper bound. Therefore, if A can reach a state from which time cannot progress, this is due to a modeling error. Well-timedness can be algorithmically checked using, for instance, the techniques proposed in [27].

3.1 The Notion of Diagnosability for Timed Automata

In discrete-event systems, diagnosability has been defined with respect to a parameter n (a natural constant), representing the maximum delay required for the diagnoser to detect a fault [21,22]. Since time is not an inherent part of the DES model, delays are captured by counting events: the diagnoser must detect a fault after at most n steps of the plant, where a step corresponds to an event (observable or not).

For timed automata, we propose a more natural definition: a timed automaton A is diagnosable with respect to a delay Δ, if a fault can be detected in A at most Δ time units after it occurs. Note that, since A is well-timed, Δ time units will eventually elapse.

Definition 2 (Diagnosability for Timed Automata). *Consider a timed automaton A. We say that A is* Δ-diagnosable*, for a natural number $\Delta \in \mathsf{N}$, if for any two finite runs ρ_1, ρ_2 of A, if ρ_1 is Δ-faulty, then either ρ_2 is faulty or $P(\rho_1, \Sigma_u) \neq P(\rho_2, \Sigma_u)$. We say that A is diagnosable if there exists some $\Delta \in \mathsf{N}$, such that A is Δ-diagnosable.*

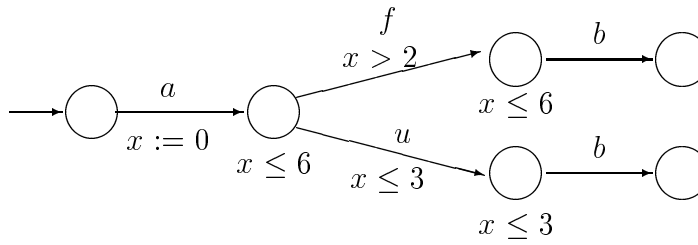

Fig. 2. A non-diagnosable timed automaton.

Example 3. 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.

We make some remarks about Definition 2.

For the sake of simplicity, we consider only one type of fault. This is not an essential assumption. The framework can be extended in a straightforward manner so that a number of different faults are considered, modeled by a set of events $\Sigma_f \subseteq \Sigma_u$. Then, the diagnosability definition would state that for each $f \in \Sigma_f$, there should not be two runs ρ_1 and ρ_2, such that f occurs in ρ_1 and at least Δ time passes afterwards[2], the projections of ρ_1 and ρ_2 to observable events are the same, but f does not occur in ρ_2. Checking diagnosability with multiple faults can be done by checking diagnosability separately for each fault. Building diagnosers which detect multiple faults can also be done by a simple extension of the single-fault construction.

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.

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 Section 3.3.

Lemma 4. *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.*

[2] We could also consider a definition with a different delay Δ_f for each fault $f \in \Sigma_f$.

Lemma 5. *A is Δ-diagnosable iff there exists a function $\phi : \mathcal{TS}_{\Sigma_o} \to \{0,1\}$, such that for every finite run ρ of A, if ρ is not faulty, then $\phi(P(\rho, \Sigma_u)) = 0$, whereas if ρ is Δ-faulty, then $\phi(P(\rho, \Sigma_u)) = 1$.*

Proof Assume A is Δ-diagnosable. Define ϕ as follows. Given $\pi \in \mathcal{TS}_{\Sigma_o}$, if there exists a finite run τ of A, such that $P(\tau, \Sigma_u) = \pi$ and τ is Δ-faulty, then $\phi(\pi) = 1$, otherwise, $\phi(\pi) = 0$. Now, consider some finite run ρ of A and let $\pi = P(\rho, \Sigma_u)$. If ρ is Δ-faulty, then $\phi(\pi) = 1$, by definition of ϕ. If ρ is not faulty, then we claim that $\phi(\pi) = 0$. Suppose not. Then, there exists a finite run τ of A, such that τ is Δ-faulty and $P(\tau, \Sigma_u) = \pi$. But this means that $P(\tau, \Sigma_u) = P(\rho, \Sigma_u)$, which contradicts the hypothesis that A is Δ-diagnosable.

Conversely, assume A is not Δ-diagnosable, that is, there exist two finite runs ρ_1 and ρ_2 of A, such that ρ_1 is Δ-faulty, ρ_2 is not faulty, and $P(\rho_1, \Sigma_u) = P(\rho_2, \Sigma_u) = \pi$. Now, $\phi(\pi)$ cannot be 0, because $\phi(P(\rho_1, \Sigma_u))$ must be 1, and $\phi(\pi)$ cannot be 1 either, because $\phi(P(\rho_1, \Sigma_u))$ must be 0. So, ϕ cannot exist. □

3.2 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 [29].

In the dense-time case, in order to check whether a given timed automaton A is diagnosable or not, we first form a special parallel product of A with itself. This product, denoted $(A\|_{\Sigma_o}A)^{-f_2}$, has as set of states the cartesian product of the states of A and contains twice as many clocks as A. Checking diagnosability of A will be reduced to finding non-zeno faulty paths in $(A\|_{\Sigma_o}A)^{-f_2}$.

$(A\|_{\Sigma_o}A)^{-f_2}$ 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 $I(q_1) \wedge I(q_2)$.

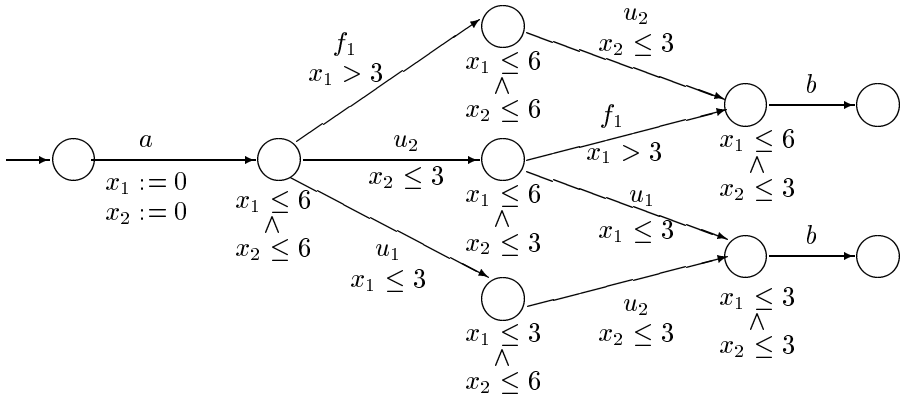

Fig. 3. The product $(A\|_{\Sigma_o}A)^{-f_2}$ for the timed automaton of Figure 1.

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.

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 6. *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 7 (Diagnosability Check). *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 the finite prefix of ρ up to exactly Δ time units after the first occurrence of f_1 in ρ. Since ρ is non-zeno, ρ_Δ is well-defined and it is clearly a run of $(A\|_{\Sigma_o}A)^{-f_2}$. Thus, by Lemma 6, 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. Because of $\rho_\Delta^1, \rho_\Delta^2$, A is not Δ-diagnosable. Since such runs can be found for any Δ, A is not diagnosable.

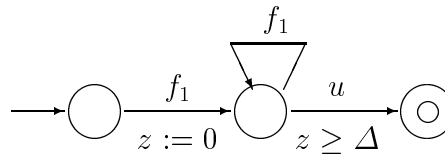

Fig. 4. Observer automaton $\mathsf{Obs}(\Delta)$.

In the other direction, assume A is not diagnosable. This means that for any $\Delta \in \mathsf{N}$, there exist two finite runs ρ^1_Δ and ρ^2_Δ of A, such that ρ^1_Δ contains f, ρ^2_Δ does not, $P(\rho^1_\Delta, \Sigma_u) = P(\rho^2_\Delta, \Sigma_u)$, and at least Δ time units elapse after the first occurrence of f in ρ^1_Δ. Therefore, by Lemma 6, 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 1, A has a non-zeno faulty run. □

From Proposition 7, 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 in [4] using the region graph construction, with a worst-case complexity of PSPACE. Since the size of $(A\|_{\Sigma_o} A)^{-f_2}$ is polynomial in the size of A, it follows that the worst case complexity of checking diagnosability is also PSPACE.

In practice, non-zeno runs can be found more efficiently, using the algorithms proposed in [8]. 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 [26,9].

3.3 Finding the Maximum Delay for Fault Detection

In the previous section we showed how to check whether a given timed automaton A is diagnosable. Now, we show how to find the required delay for fault detection, Δ_{min}, introduced in Lemma 4.

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 8 (Maximum Delay for Fault Detection). *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 8 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.

4 Diagnosers

For this section, we fix a timed automaton A, which is well-timed and Δ-diagnosable.

Our objective is to construct a *diagnoser* for A, as illustrated in Figure 5. The diagnoser is a deterministic machine which instantaneously observes each observable event generated by the plant and measures delays between successive events. It is also realistic to assume that the diagnoser sets a *time-out*, so that, even if no event happens for some time, the diagnoser will still react to the passage of time.

Each time an event or a delay is observed, the diagnoser changes its state. A state of the diagnoser is marked yes (a fault has been detected) or not-yet (no fault has been detected so far). A valid diagnoser should announce a fault only if a fault indeed occurred. Moreover, when a fault occurs, the diagnoser should announce it at most Δ time units later. Finally, a diagnoser should never stop announcing a fault once it has announced it.

Definition 9 (Diagnoser). *A diagnoser for a timed automaton A is a tuple $(\mathcal{W}, W^0, f_e, f_t, f_d)$, where $\mathcal{W}$ is a set of states, $W^0 \in \mathcal{W}$ is the initial state, $f_e : \mathcal{W} \times \Sigma_o \to \mathcal{W}$ is the* event transition function*, $f_t : \mathcal{W} \times \mathsf{R} \to \mathcal{W}$ is the* time transition function*, and $f_d : \mathcal{W} \to \{$not-yet, yes$\}$ is the* decision function*. The time transition function must satisfy the usual* semi-group property*, that is, for all $W \in \mathcal{W}$, for all $\delta, \delta', \delta'' \in \mathsf{R}$, if $\delta = \delta' + \delta''$, then $f_t(W, \delta) = f_t(f_t(W, \delta'), \delta'')$.*

A finite timed sequence $\rho = \gamma_1, \gamma_2, \cdots, \gamma_n$ over Σ_o defines a function on the states $\mathcal{W}$ of the diagnoser. If $W \in \mathcal{W}$, $\rho(W)$ is defined to be the last state W_n in

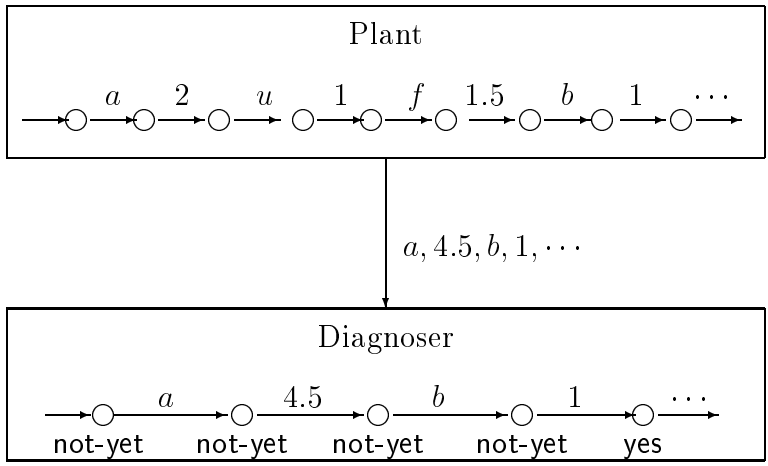

Fig. 5. Illustration of a diagnoser.

a sequence of states $W_0, W_1, \cdots, W_n$, where $W_0 = W$, and for each $i = 1, 2, ...$, if γ_i is a delay $\delta \in \mathsf{R}$, then $W_i = f_t(W_{i-1}, \delta)$, whereas if γ_i is an event $a \in \Sigma_o$, then $W_i = f_t(W_{i-1}, a)$.

A diagnoser is called *valid* if it satisfies the following conditions:

1. If ρ is a non-faulty finite run of A, then $f_d(\ \pi(W^0)\ ) = \mathsf{not\text{-}yet}$, where $\pi = P(\rho, \Sigma_u)$.
2. If ρ is a finite Δ-faulty run of A, then $f_d(\ \pi(W^0)\ ) = \mathsf{yes}$, where $\pi = P(\rho, \Sigma_u)$.
3. If ρ_1, ρ_2 are finite runs of A, ρ_1 is a prefix of ρ_2, and $\pi_i = P(\rho_i, \Sigma_u)$, for $i = 1, 2$, then, if $f_d(\ \pi_1(W^0)\ ) = \mathsf{yes}$, then $f_d(\ \pi_2(W^0)\ ) = \mathsf{yes}$. That is, once the diagnoser has output yes, it can no longer output not-yet.

4.1 Constructing a Diagnoser

We now show how to construct a diagnoser for A. The diagnoser will work as a *state estimator* for A, that is, each state of the diagnoser will correspond to a set of possible states of the timed automaton.

For simplicity of the presentation, 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 6. The motivation for the transformation is to reduce the fault detection problem into a state estimation

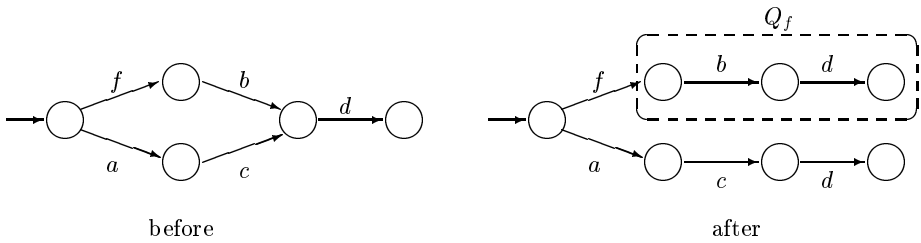

Fig. 6. Transforming an automaton.

problem: a fault has been detected once the diagnoser is certain that the plant is in some state with discrete part in Q_f.

We are now ready to define the diagnoser. Recall that R_A is the set of reachable states of A. The state-space $\mathcal{W}$ of the diagnoser is defined to be $\mathcal{W} = 2^{R_A}$.

The decision function f_d of the diagnoser is defined as follows:

$$f_d(W) = \begin{cases} \mathsf{yes}, & \text{if } \forall s \in W, \mathsf{discrete}(s) \in Q_f \\ \mathsf{not\text{-}yet}, & \text{otherwise.} \end{cases} \tag{1}$$

Recall that, 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'. Then, the transition functions of the diagnoser are defined as follows:

$$f_e(W, a) = \{e(s) \mid s \in W, e \in E(a)\} \tag{2}$$

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

It can be seen that f_t defined as above satisfies the semi-group property.

The initial state of the diagnoser is defined to be

$$W^0 = \{\rho(s^0) \mid \rho \in \mathsf{Runs}(A, \Sigma_u), \mathsf{time}(\rho) = 0\}, \tag{4}$$

that is, W^0 contains all states reached in zero delay from the initial state of the automaton, by performing only unobservable actions.

The following lemma states that a diagnoser acts as a state estimator for A.

Lemma 10. *For every finite run ρ of A, if $s = \rho(s^0)$ and $\pi = P(\rho, \Sigma_u)$, then $s \in \pi(W^0)$. Conversely, for every finite run π of the diagnoser, for all $s \in \pi(W^0)$, there exists a finite run ρ of A, such that $s = \rho(s^0)$ and $P(\rho, \Sigma_u) = \pi$.*

Proposition 11. *If A is a Δ-diagnosable timed automaton, then the tuple $(\mathcal{W}, W^0, f_e, f_t, f_d)$ defined above is a valid diagnoser for A.*

Proof Let ρ be a finite run of A and let $\pi = P(\rho, \Sigma_u)$.

Assume first that ρ is non-faulty. From the assumption about the structure of A, it must be that for all $s \in \rho(s^0)$, we have $\mathsf{discrete}(s) \notin Q_f$. Also, $\rho(s^0) \neq \emptyset$,

since ρ is a run of A. Let $s \in \rho(s^0)$. By Lemma 10, $s \in \pi(W^0)$. Thus, by the definition of f_d, $f_d(\pi(W^0)) = \mathsf{not\text{-}yet}$. This proves the first condition for validity.

Now, assume that ρ is Δ-faulty. By Lemma 10, $s = \rho(s^0) \in \pi(W^0)$ and, since ρ is faulty, $\mathsf{discrete}(s) \in Q_f$. Now, pick some $s' \in \pi(W^0)$. By Lemma 10, there exists a finite run ρ' of A, such that $s' = \rho'(s^0)$ and $P(\rho', \Sigma_u) = \pi$. Since A is Δ-diagnosable, ρ' has to be faulty. Therefore, $\mathsf{discrete}(s') \in Q_f$. That is, for all $s' \in \pi(W^0)$, we have $\mathsf{discrete}(s') \in Q_f$. Thus, by the definition of f_d, $f_d(\pi(W^0)) = \mathsf{yes}$, which proves the second condition for validity.

As for the third condition, recall that, once A enters the set Q_f, it never exits. Thus, if W is a state of the diagnoser such that for all $s \in W$, $\mathsf{discrete}(s) \in Q_f$, then, at any future state W', the same will hold, thus, the decision of the diagnoser will not change. □

4.2 Diagnoser Implementation and Run-Time Considerations

In this section, we show how diagnoser states can be represented using finitary data structures and how the decision and transition functions can be effectively computed on these structures. In fact, we will use technology not much different from that used in timed-automata model-checkers such as Kronos [9] or Uppaal [1].

A state of the diagnoser will be a list $[(q_1, \zeta_1), ..., (q_k, \zeta_k)]$, where $q_i \in Q$ and ζ_i is a polyhedron on X, the set of clocks of A. Such a polyhedron can be effectively represented using well-known data structures called *difference bound matrices* (DBMs) [13]. 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 of the diagnoser can be easily computed by scanning the list: if the list contains some pair (q_i, ζ_i) such that $q_i \notin Q_f$, then the function returns $\mathsf{not\text{-}yet}$, otherwise it returns yes.

The event transition function $f_e(W, a)$ can be computed as follows. If $W = [(q_1, \zeta_1), ..., (q_k, \zeta_k)]$ is the current list, start with a new empty list W', and for each $(q_i, \zeta_i) \in W$, 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 r', where ζ' is the polyhedron

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

which contains all $v[\mathsf{reset}(e) := 0] \in I(\mathsf{dest}(e))$, such that $v \in \zeta_i \cap \mathsf{guard}(e)$. Since the number of pairs in W is finite, and there is a finite number of transitions e, the number of pairs in W' is also finite.

The time transition function $f_t(W, \delta)$ can be computed for any delay δ which is a rational number[3], using a *reachability* procedure. There are two differences

[3] In practice, the granularity of δ will be restricted by the numerical accuracy of the machine.

between the standard reachability procedure for timed automata, and the one for computing $f_t(W, \delta)$. For $f_t(W, \delta)$, reachability is restricted only to unobservable transitions of A. Standard reachability can be easily modified to meet this condition: simply select only transitions labeled with unobservable events. Also, standard reachability computes the set of states reachable at any time, whereas reachability for $f_t(W, \delta)$ computes the set of states reachable in *exactly* δ time units. This condition can be satisfied as follows.

First, we compute the set of states reachable from W in *at most* δ *time units*. Call this set $W_{\leq\delta}$. $W_{\leq\delta}$ can be represented as a finite list of DBMs, and can be computed by extending W with a new clock z initially set to 0 and exploring only the states satisfying $z \leq \delta$. Once $W_{\leq\delta}$ is computed, we take the intersection $W_{\leq\delta} \cap (z = \delta)$: this set contains all states reachable from W in exactly δ time units. Finally, the clock z is eliminated by existential quantification, and the final result is obtained:

$$f_t(W, \delta) = \exists z \,.\, \Big(W_{\leq\delta} \cap (z = \delta)\Big). \tag{5}$$

```
set W to W^0 ;                                  /* initialize diagnoser state */
set timer x to 0 ;                              /* set an incrementing timer */
loop
   if (f_d(W) = yes) then
      announce fault ;
   end if ;
   await (event interrupt) or (x = TO) ; /* await event or timer interrupt */
   if (event a interrupt) then
      set W to f_e(f_t(W, x), a) ;              /* update state by time and discrete step */
   else
      set W to f_t(W, TO) ;                     /* update state by time step */
   end if ;
   set timer x to 0 ;                           /* reset timer for new time interrupt */
end loop.
```

Fig. 7. Diagnoser implementation loop.

Figure 7 shows how the implementation of a diagnoser would look like, in pseudo-code. After initializing its state and setting a timer to 0, the diagnoser enters an infinite loop. Inside the loop, the diagnoser checks whether a fault has occurred and if so it announces it. Then, it waits until an (observable) event is received, or the timer times-out (this happens after TO time units, where TO is a parameter), updates its state accordingly, and repeats the loop. Such an implementation requires an execution platform which provides some kind of event interrupts, time-outs and clock readings[4].

[4] The memory required for such an implementation is generally unbounded, since the set of reachable states of the diagnoser is generally unbounded. This is not surprising,

A diagnoser implemented like the one shown in Figure 7 will function correctly, provided the loop can be executed sufficiently fast. In practice, this means that the maximum time to compute the transition function should not be greater than the minimum delay between two observable events. This requirement is similar to the *synchrony hypothesis*, which implies the correct execution of programs written in *synchronous languages* such as Esterel [2] or Lustre [14].

5 Related Work and Discussion

Fault diagnosis has been studied by different communities and in different contexts, e.g., see [20,21,24,7,6], and citations therein. We restrict our discussion to work closely related to timed systems.

[11,32] 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 [23], to reduce the problem into a finite-state diagnosis problem.

[16] 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 [6,17]. 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 [19]. The problem has been studied for timed and hybrid models as well (e.g., see [31,10,12,5,18,30,25]). Some of these works are restricted to a discrete-time framework, for example [10,18]. 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[5].

[30] 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

since the set of states of a timed automaton is also generally infinite. Thus, even if the diagnoser was represented as a timed automaton, an unbounded memory would be generally needed to implement clocks.

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

the plant to a domain of possible observations: when the plant is at state s, the controller observes $vis(s)$. Then, [30] 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

1. Uppaal web-site: `www.docs.uu.se/docs/rtmv/uppaal/`.
2. Esterel web-site: `www.esterel.org`.
3. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
4. Rajeev Alur. *Techniques for Automatic Verification of Real-Time Systems*. PhD thesis, Department of Computer Science, Stanford University, 1991.
5. 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.
6. A. Balluchi, L. Benvenuti, M. Di Benedetto, and A. Sangiovanni-Vincentelli. Design of observers for hybrid systems. In *Hybrid Systems: Computation and Control*, 2002. To appear.
7. P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella. Diagnosis of large active systems. *Artificial Intelligence*, 110, 1999.
8. A. Bouajjani, S. Tripakis, and S. Yovine. On-the-fly symbolic model checking for real-time systems. In *Proc. of the 18th IEEE Real-Time Systems Symposium, San Francisco, CA*, pages 232–243. IEEE, December 1997.
9. M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: a model-checking tool for real-time systems. In *Proc. of the 10th Conference on Computer-Aided Verification, CAV'98*. LNCS 1427, Springer-Verlag, 1998.
10. B.A. Brandin and W.M. Wonham. Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39(2), 1994.
11. Yi-Liang Chen and Gregory Provan. Modeling and diagnosis of timed discrete event systems – a factory automation example. In *ACC*, 1997.
12. 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.
13. D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, Lecture Notes in Computer Science 407, pages 197–212. Springer–Verlag, 1989.
14. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language Lustre. *Proceedings of the IEEE*, 79(9), September 1991.
15. 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.

16. 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.
17. S. Narasimhan and G. Biswas. An approach to model-based diagnosis of hybrid systems. In *Hybrid Systems: Computation and Control*, 2002. To appear.
18. 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.
19. P. Ramadge and W. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1), January 1987.
20. 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.
21. 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.
22. 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.
23. J. Sztipanovits, R. Carnes, and A. Misra. Finite state temporal automata modeling for fault diagnosis. In *9th AIAA Conference on Computing in Aerospace*, 1993.
24. J. Sztipanovits and A. Misra. Diagnosis of discrete event systems using ordered binary decision diagrams. In *7th Intl. Workshop on Principles of Diagnosis*, 1996.
25. C. Tomlin, J. Lygeros, and S. Sastry. Synthesizing controllers for nonlinear hybrid systems. In *Hybrid Systems: Computation and Control*. LNCS 1386, Springer-Verlag, 1998.
26. S. Tripakis. *The formal analysis of timed systems in practice*. PhD thesis, Université Joseph Fourrier de Grenoble, 1998.
27. S. Tripakis. Verifying progress in timed systems. In *ARTS'99*, LNCS 1601, 1999.
28. S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, January 2001.
29. J.N. Tsitsiklis. On the control of discrete event dynamical systems. *Mathematics of Control, Signals and Systems*, 2(2), 1989.
30. H. Wong-Toi. The synthesis of controllers for linear hybrid automata. In *Proc. of IEEE Conference on Decision and Control*, 1997.
31. 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.
32. S. Hashtrudi Zad, R.H. Kwong, and W.M. Wonham. Fault diagnosis in finite-state automata and timed discrete-event systems. In *38th IEEE Conference on Decision and Control*, 1999.