HAL
archives-ouvertes.fr

Fault Diagnosis of Timed Systems

Franck Cassez, Stavros Tripakis

▶ **To cite this version:**

Franck Cassez, Stavros Tripakis. Fault Diagnosis of Timed Systems. Roux, Olivier H. and Jard, Claude. Communicating Embedded Systems – Software and Design, ISTE Publishing Ltd. – John Wiley & Sons, Ltd., 2009. <inria-00493633>

HAL Id: inria-00493633
https://hal.inria.fr/inria-00493633

Submitted on 21 Jun 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Chapter 4

Fault Diagnosis of Timed Systems

In this Chapter, we review the main results pertaining to the problem of fault diagnosis of timed automata. Timed automata are introduced in Chapter 1 and Chapter 2 in this book, and the reader not familiar with this model is invited to read them first.

4.1. Introduction

Many computerized systems (sometimes embedded) supervise/control devices or appliances the failure of which can be life-threatening (airplanes, railways, nuclear plants, medical devices, ...) or extremely expensive (Ariane 5 rocket, telephone networks, ...).

For more than two decades, researchers have tried to develop tools and algorithms to increase the dependability of computerized systems. All the verification techniques can be gathered under the generic name of *formal methods*, the most famous and successful one being certainly *model checking* [SCH 99], which consists in building a mathematical model of a system S, and then check that it satisfies a given property ϕ. It is sometimes even possible to *synthesize* a *controller* C (Cf. Chapter 3) for S, in order to ensure that the controlled system $C(S)$ satisfies a property ϕ.

The model checking and controller synthesis techniques assume that we can build a complete model of the system (and the controller), and everything is observable (controller synthesis under partial observation can also be solved for some particular properties for timed automata.) Also it might be that no controller exists or it is not possible to compute one, to ensure that a system $C(S)$ satisfies a property ϕ.

Chapter written by Franck CASSEZ and Stavros TRIPAKIS.

Still we might be able to detect when property ϕ is violated. An alternative approach to verification is thus to try and detect erroneous or *faulty* behaviours, and in case a fault occurs, to trigger a fault handler (shut down the system, restart from a particular state, ...).

The notion of *fault diagnosis* for discrete event systems (DES) was formalized in the mid 90's in [SAM 95, SAM 96]. The two previous papers introduced the formal definitions of fault diagnosis for DES and have been followed by numerous papers on the subject. The framework of fault diagnosis for DES is the following:

– we assume that a formal model S of the system to monitor is available. The *faults* are specified in S by special events $f_1, \ldots, f_n$. Thus we have a formal model of S with the faults, but the faults cannot be observed at runtime (sensors can only observe a subset of the events of the system, and faults are not observable); formally the events in S are partionned into a set of observable events and a set of unobservable events (which includes $f_1, \ldots, f_n$).

– fault diagnosis consists in detecting faulty behaviours —which are behaviours in which a fault event occurs— by observing only the observable events (and of course assuming we have a complete knowledge of the formal model of S). A fault should be detected in a bounded amount of time after it occurred and the bounded fault diagnosis problem asks the following: "Is it possible to detect a fault f_i within a maximum of Δ time units after it occurred?". For DES, one time unit corresponds to firing a discrete transition.

When timing constraints are part of the system description, S can be modelled as a timed automaton [ALU 94a]. The fault diagnosis problem for timed automata has been introduced in [TRI 02] and further studied in [BOU 05]. Some open problems still remain and are listed in Section 4.6.

In Section 4.2, we briefly recall the basic notions of timed languages and timed automata. For a detailed introduction to these concepts we refer the reader to Chapter 1 and Chapter 2 in this book.

In Section 4.3, we define the notion of *diagnoser* which is an observer the purpose of which is to detect faults. We also introduce the fault diagnosis problems studied in the following sections, together with a necessary and sufficient condition for *diagnosability*.

Section 4.4 is devoted to fault diagnosis of DES. The algorithms presented in this section are extended to timed automata in Section 4.5.

4.2. Notations

$\mathbb{B} = \{\top, \bot\}$ is the set of boolean values, $\mathbb{N}$ the set of natural numbers, $\mathbb{Z}$ the set of integers and $\mathbb{Q}$ the set of rational numbers. $\mathbb{R}$ is the set of real numbers and $\mathbb{R}_{\geq 0}$ is the set of non-negative real numbers. Let X be a finite set of variables called *clocks*. A *clock valuation* is a mapping $v : X \to \mathbb{R}_{\geq 0}$. $\mathbb{R}_{\geq 0}^X$ is the set of clock valuations over X. We write $\mathbf{0}_X$ for the *zero* valuation where all the clocks in X are set to 0 (we use $\mathbf{0}$ when X is clear from the context). Given $\delta \in \mathbb{R}$, $v + \delta$ denotes the valuation defined by $(v + \delta)(x) = v(x) + \delta$. We let $\mathcal{C}(X)$ be the set of *convex constraints* on X, i.e, the set of conjunctions of constraints of the form $x \bowtie c$ with $c \in \mathbb{Z}$ and $\bowtie \in \{\leq, <, =, >, \geq\}$. Given a constraint $g \in \mathcal{C}(X)$ and a valuation v, we write $v \models g$ if g is satisfied by the valuation v. Given a set $R \subseteq X$ and a valuation v of the clocks in X, $v[R]$ is the valuation defined by $v[R](x) = v(x)$ if $x \notin R$ and $v[R](x) = 0$ otherwise.

4.2.1. *Timed Words and Timed Languages*

Let Σ be a finite alphabet. The set of finite (resp. infinite) words over Σ is Σ^* (resp. Σ^ω) and we let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. A *language* L is any subset of Σ^∞. A finite (resp. infinite) *timed word* over Σ is a word in $(\mathbb{R}_{\geq 0}.\Sigma)^*.\mathbb{R}_{\geq 0}$ (resp. $(\mathbb{R}_{\geq 0}.\Sigma)^\omega$). In this chapter we write timed words as $0.4\ a\ 1.0\ b\ 2.7\ c \cdots$ where the real values are the durations elapsed between two letters: thus c occurs at global time 4.1. We let *Duration*(w) be the duration of a timed word w which is defined to be the sum of the durations (in $\mathbb{R}_{\geq 0}$) which appear in w; if this sum is infinite, the duration is ∞. Note that the duration of an infinite word can be finite, and such words which contain an infinite number of letters, are called *Zeno* words. We let *Unt*(w) be the *untimed* version of w obtained by erasing all the durations in w. An example of untiming is *Unt*$(0.4\ a\ 1.0\ b\ 2.7\ c) = abc$.

$TW^*(\Sigma)$ is the set of finite timed words over Σ, $TW^\omega(\Sigma)$, the set of infinite timed words and $TW^\infty(\Sigma) = TW^*(\Sigma) \cup TW^\omega(\Sigma)$. A *timed language* is any subset of $TW^\infty(\Sigma)$.

Let $\boldsymbol{\pi}_{/\Sigma'}$ be the projection of timed words of $TW^\infty(\Sigma)$ over timed words of $TW^\infty(\Sigma')$. When projecting a timed word w on a sub-alphabet $\Sigma' \subseteq \Sigma$, the durations elapsed between two events are set accordingly: for instance $\boldsymbol{\pi}_{/\{a,c\}}(0.4\ a\ 1.0\ b\ 2.7\ c) = 0.4\ a\ 3.7\ c$ (note that projection erases some letters but updates the time elapsed between two letters). Given a timed language L, we let *Unt*$(L) = \{$*Unt*$(w) \mid w \in L\}$. Given $\Sigma' \subseteq \Sigma$, $\boldsymbol{\pi}_{/\Sigma'}(L) = \{\boldsymbol{\pi}_{/\Sigma'}(w) \mid w \in L\}$.

4.2.2. *Timed Automata*

Timed automata are finite automata extended with real-valued clocks to specify timing constraints between occurrences of events. For a detailed presentation of the fundamental results for timed automata, the reader is referred to the seminal paper of R. Alur and D. Dill [ALU 94a]. In this chapter, we consider timed automata with a *silent* or *invisible* action, denoted τ, and we let $\Sigma_\tau = \Sigma \cup \{\tau\}$.

DEFINITION 4.1 (TIMED AUTOMATON, TA).– A *Timed Automaton* A is a tuple $(L, l_0, X, \Sigma_\tau, E, \mathit{Inv}, F, R)$ where:

– L is a finite set of *locations*;

– l_0 is the *initial location*;

– X is a finite set of *clocks*;

– Σ is a finite set of *actions*;

– $E \subseteq L \times \mathcal{C}(X) \times \Sigma_\tau \times 2^X \times L$ is a finite set of *transitions*; in a transition (ℓ, g, a, r, ℓ'), g is the *guard*, a the *action*, and r the *reset* set;

– $\mathit{Inv} \in \mathcal{C}(X)^L$ associates with each location an *invariant*; as usual we require the invariants to be conjunctions of constraints of the form $x \preceq c$ with $\preceq \in \{<, \leq\}$.

– $F \subseteq L$ and $R \subseteq L$ are respectively the *final* and *repeated* sets of locations. □

In the sequel we often omit the sets R and F in TA and this implicitly means $F = L$ and $R = \varnothing$.

The semantics of a timed automaton is a timed transition system as described in Chapter 1 and Chapter 2. A *run* ρ of A from (q_0, v_0) is a sequence of the form:

$$\rho = (q_0, v_0) \xrightarrow{\delta_0} (q_0, v_0 + \delta_0) \xrightarrow{a_1} (q_1, v_1) \cdots \xrightarrow{a_n} (q_n, v_n) \xrightarrow{\delta_n} (q_n, v_n + \delta_n) \cdots$$

where $q_i \in L, v_i : X \to \mathbb{R}_{\geq 0}$ is a clock valuation, $a_i \in \Sigma$ and $\delta_i \in \mathbb{R}_{\geq 0}$. The set of finite (resp. infinite) runs in A from a state s is denoted $\mathit{Runs}^*(s, A)$ (resp. $\mathit{Runs}^\omega(s, A)$) and we let $\mathit{Runs}^*(A) = \mathit{Runs}^*((l_0, \mathbf{0}), A)$ and $\mathit{Runs}^\omega(A) = \mathit{Runs}^\omega((l_0, \mathbf{0}), A)$. The set of runs of A is $\mathit{Runs}(A) = \mathit{Runs}^*(A) \cup \mathit{Runs}^\omega(A)$. If ρ is finite and ends in s_n, we define $\mathit{last}(\rho) = s_n$. Because of the denseness of the time domain, the unfolding of A as a graph is infinite (uncountable number of states and delay edges).

The *trace*, $\mathit{tr}(\rho)$, of a run ρ is the timed word $\delta_0 a_0 \delta_1 a_1 \cdots a_n \delta_n \cdots$. The duration of the run ρ is $\mathit{Duration}(\rho) = \mathit{Duration}(\mathit{tr}(\rho))$. For $V \subseteq \mathit{Runs}(A)$, we let $\mathit{Tr}(V) = \{\mathit{tr}(\rho) \mid \varrho \in V\}$, which is the set of traces of the runs in V.

A finite (resp. infinite) timed word w is *accepted* by A if $w = \boldsymbol{\pi}_{/\Sigma}(\mathit{tr}(\rho))$ for some finite run ρ of A that ends in an F-location (resp. a infinite run that reaches infinitely often an R-location). $\mathcal{L}^*(A)$ (resp. $\mathcal{L}^\omega(A)$) is the set of traces of finite (resp. infinite)

timed words accepted by A, and $\mathcal{L}(A) = \mathcal{L}^*(A) \cup \mathcal{L}^\omega(A)$ is the set of timed words accepted by A.

4.2.3. *Region Graph of a TA*

The *region graph* $RG(A)$ of a TA A is a finite quotient of the infinite graph of A which is time-abstract bisimilar to A [ALU 94a] (see Chapter 2 for a formal definition of the region graph). It is a finite automaton on the alphabet $E' = E \cup \{\tau\}$. The states of $RG(A)$ are pairs (ℓ, r) where $\ell \in L$ is a location of A and r is a *region* of $\mathbb{R}^X_{\geq 0}$. More generally, the edges of the graph are tuples (s, t, s') where s, s' are states of $RG(A)$ and $t \in E'$. Genuine unobservable moves of A labelled τ are labelled by tuples of the form $(s, (g, \tau, R), s')$ in $RG(A)$. An edge (g, λ, R) in the region graph corresponds to a discrete transition of A with guard g, action λ and reset set R. A τ move in $RG(A)$ stands for a delay move to the time-successor region. The initial state of $RG(A)$ is $(l_0, \mathbf{0})$. A final (resp. repeated) state of $RG(A)$ is a state (ℓ, r) with $\ell \in F$ (resp. $\ell \in R$). A fundamental property of the region graph [ALU 94a] is:

THEOREM 4.1 ([ALU 94A]).– $\mathcal{L}(RG(A)) = \mathit{Unt}(\mathcal{L}(A))$.

In other words:

1) if w is accepted by $RG(A)$, then there is a timed word v with $\mathit{Unt}(v) = w$ s.t. v is accepted by A.

2) if v is accepted by A, then $\mathit{Unt}(w)$ is accepted by $RG(A)$.

The (maximum) size of the region graph is exponential in the number of clocks and in the maximum constant of the automaton A (see [ALU 94a]): $|RG(A)| = |L| \cdot |X|! \cdot 2^{|X|} \cdot K^{|X|}$ where K is the largest constant used in A.

4.2.4. *Product of TA*

The product of TA is defined in a standard way, keeping in mind that the τ actions do not synchronize.

DEFINITION 4.2 (PRODUCT OF TA).–Let $A_i = (L_i, l_0^i, X_i, \Sigma_\tau^i, E_i, \mathit{Inv}_i), i \in \{1, 2\}$, be two TA s.t. $X_1 \cap X_2 = \varnothing$. The *product* of A_1 and A_2 is the TA $A_1 \times A_2 = (L, l_0, X, \Sigma_\tau, E, \mathit{Inv})$ given by:

– $L = L_1 \times L_2$;

– $l_0 = (l_0^1, l_0^2)$;

– $\Sigma = \Sigma^1 \cup \Sigma^2$;

– $X = X_1 \cup X_2$; and

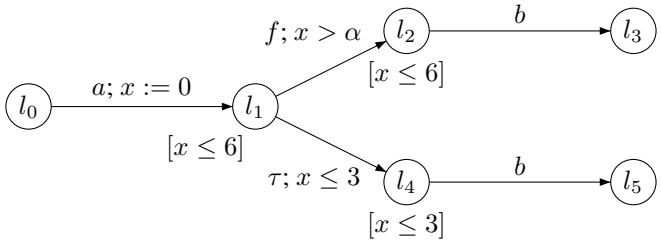


Figure 4.1. *Automaton* $\mathcal{A}(\alpha)$

– $E \subseteq L \times \mathcal{C}(X) \times \Sigma_\tau \times 2^X \times L$ and $((\ell_1, \ell_2), g_{1,2}, \sigma, r, (\ell'_1, \ell'_2)) \in E$ if:

- either $\sigma \in (\Sigma_1 \cap \Sigma_2) \setminus \{\tau\}$, and (i) $(\ell_k, g_k, \sigma, r_k, \ell'_k) \in E_k$ for $k = 1$ and $k = 2$; (ii) $g_{1,2} = g_1 \wedge g_2$ and (iii) $r = r_1 \cup r_2$;

- or for $k = 1$ or $k = 2, \sigma \in (\Sigma_k \setminus \Sigma_{3-k}) \cup \{\tau\}$, and (i) $(\ell_k, g_k, \sigma, r_k, \ell'_k) \in E_k$; (ii) $g_{1,2} = g_k$ and (iii) $r = r_k$;

– and finally $\mathit{Inv}(\ell_1, \ell_2) = \mathit{Inv}(\ell_1) \wedge \mathit{Inv}(\ell_2)$. □

4.2.5. *Timed Automata with Faults*

To model timed systems with faults, we use timed automata on the alphabet $\Sigma_{\tau,f} = \Sigma_\tau \cup \{f\}$ where f is the *faulty* (unobservable) event. We only consider one type of fault here, but the results we give are valid for many types of faults $\{f_1, f_2, \cdots, f_n\}$: indeed solving the many-types diagnosability problem amounts to solving n one-type diagnosability problems [YOO 02]. Other unobservable events are abstracted as a τ action (one τ suffices as these events are all unobservable).

The system we want to supervise is given by a timed automaton $A = (L, l_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$. Figure 4.1 gives an example of such a system (α is a positive integer parameter). Invariants in the automaton $\mathcal{A}(\alpha)$ are written within square brackets as in $[x \leq 3]$.

Let $\Delta \in \mathbb{N}$. A run ρ of A such that

$$\begin{aligned}\rho \quad = \quad & (\ell_0, v_0) \xrightarrow{\delta_0} (\ell_0, v_0 + \delta_0) \xrightarrow{a_1} (\ell_1, v_1) \cdots \\ & \cdots \xrightarrow{a_{n-1}} (\ell_n, v_n) \xrightarrow{\delta_n} (\ell_n, v_n + \delta_n) \cdots\end{aligned}$$

is Δ-faulty if: (1) there is an index i s.t. $a_i = f$ and (2) the duration of the run $\rho' = (\ell_i, v_i) \xrightarrow{\delta_i} \cdots \xrightarrow{\delta_n} (\ell_n, v_n + \delta_n) \cdots$ is larger than Δ. We let $\mathit{Faulty}_{\geq\Delta}(A)$ be

the set of Δ-faulty runs of A. Note that by definition, if $\Delta' \geq \Delta$ then $\mathit{Faulty}_{\geq\Delta'}(A) \subseteq \mathit{Faulty}_{\geq\Delta}(A)$. We let $\mathit{Faulty}(A) = \cup_{\Delta\geq 0}\mathit{Faulty}_{\geq\Delta}(A) = \mathit{Faulty}_{\geq 0}(A)$ be the set of faulty runs of A, and $\mathit{NonFaulty}(A) = \mathit{Runs}(A) \setminus \mathit{Faulty}(A)$ be the set of non-faulty runs of A. Finally, we let

$$\mathit{Faulty}^{tr}_{\geq\Delta}(A) = \mathit{Tr}(\mathit{Faulty}_{\geq\Delta}(A)) \quad \textit{and} \quad \mathit{NonFaulty}^{tr}(A) = \mathit{Tr}(\mathit{NonFaulty}(A))$$

which are respectively the traces of Δ-faulty and non-faulty runs of A.

4.3. Fault Diagnosis Problems

In this section, we give the definition of a *diagnoser*. We then introduce the formal definitions of the fault diagnosis problems we are interested in. We end this section by giving a necessary and sufficient condition for *diagnosability* that we use in the next sections.

4.3.1. *Diagnoser*

The purpose of fault diagnosis is to detect a fault as soon as possible. Faults (f event) are unobservable and only the events in Σ can be observed as well as the time elapsed between these events. Whenever the system generates a timed word w, the observer can only see $\boldsymbol{\pi}_{/\Sigma}(w)$. If an observer can detect faults in this way it is called a *diagnoser*. A diagnoser must detect a fault within a given delay $\Delta \in \mathbb{N}$.

DEFINITION 4.3 ((Σ, Δ)-DIAGNOSER).– Let A be a timed automaton over the alphabet $\Sigma_{\tau,f}$ and $\Delta \in \mathbb{N}$. A (Σ, Δ)-*diagnoser* for A is a mapping $D : \mathit{TW}^*(\Sigma) \to \{0, 1\}$ such that:

– for each $\varrho \in \mathit{NonFaulty}(A)$, $D(\boldsymbol{\pi}_{/\Sigma}(\mathit{tr}(\rho))) = 0$,

– for each $\varrho \in \mathit{Faulty}_{\geq\Delta}(A)$, $D(\boldsymbol{\pi}_{/\Sigma}(\mathit{tr}(\rho))) = 1$. □

A is (Σ, Δ)-*diagnosable* if there exists a (Σ, Δ)-diagnoser for A. A is Σ-*diagnosable* if there is some $\Delta \in \mathbb{N}$ s.t. A is (Σ, Δ)-diagnosable. In the sequel, we often omit the paramater Σ when it is clear from the context.

REMARK 4.1 .– Nothing is required for the Δ'-faulty words with $\Delta' < \Delta$. Thus a diagnoser could change its mind and answers 1 for a Δ'-faulty word, and 0 for a Δ''-faulty word with $\Delta' < \Delta'' < \Delta$.

EXAMPLE 4.1 .– The timed automaton $\mathcal{A}(3)$ in Figure 4.1 taken from [TRI 02] is 3-diagnosable. For the timed words of the form $t.a.\delta.b.t'$ with $\delta \leq 3$, no fault has occurred, whereas when $\delta > 3$ a fault must have occurred. A diagnoser can then be

easily constructed. As we have to wait for a "b" action to detect a fault, D cannot detect a fault in stricly less than 3 time units. If $\alpha = 2$, in $\mathcal{A}(2)$ there are two runs (we write (ℓ, v) a state with v the value of clock x):

$$\begin{aligned}
\rho_1(\delta) &= (l_0,0) \xrightarrow{a} (l_1,0) \xrightarrow{2.5} (l_1,2.5) \xrightarrow{f} (l_2,2.5) \\
&\qquad \xrightarrow{0.2} (l_2,2.7) \xrightarrow{b} (l_3,2.7) \xrightarrow{\delta} (l_2,2.7+\delta) \\
\rho_2(\delta) &= (l_0,0) \xrightarrow{a} (l_1,0) \xrightarrow{2.5} (l_1,2.5) \xrightarrow{\tau} (l_4,2.5) \\
&\qquad \xrightarrow{0.2} (l_4,2.7) \xrightarrow{b} (l_5,2.7) \xrightarrow{\delta} (l_5,2.7+\delta)
\end{aligned}$$

that satisfy $\boldsymbol{\pi}_{/\Sigma}(tr(\rho_1(\delta))) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho_2(\delta)))$, and this for every $\delta \geq 0$. For each $\Delta \in \mathbb{N}$, there are two runs $\rho_1(\Delta)$ and $\rho_2(\Delta)$ which produce the same observations and thus no diagnoser can exist. $\mathcal{A}(2)$ is not diagnosable.

4.3.2. *The Problems*

The classical fault diagnosis problems we are interested in are the following:

PROBLEM 4.1 ((Σ, Δ)-DIAGNOSABILITY).–
INPUT: A timed automaton $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$ and $\Delta \in \mathbb{N}$.
PROBLEM: Is A (Σ, Δ)-diagnosable?

PROBLEM 4.2 (Σ-DIAGNOSABILITY).–
INPUT: A timed automaton $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$.
PROBLEM: Is A Σ-diagnosable?

PROBLEM 4.3 (MAXIMUM DELAY TO ANNOUNCE A FAULT).–
INPUT: A timed automaton $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$.
PROBLEM: If A is Σ-diagnosable, what is the minimum Δ s.t. A is (Σ, Δ)-diagnosable ?

PROBLEM 4.4 (DIAGNOSER SYNTHESIS).–
INPUT: A timed automaton $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$.
PROBLEM: If A is (Σ, Δ)-diagnosable, compute a witness (Σ, Δ)-diagnoser D.

4.3.3. *Necessary and Sufficient Condition for Diagnosability*

According to Definition 4.3, A is Σ-diagnosable, if and only if, there is some $\Delta \in \mathbb{N}$ s.t. A is (Σ, Δ)-diagnosable. Thus:

$$A \textbf{ is not } \Sigma\text{-diagnosable} \iff \forall \Delta \in \mathbb{N}, A \text{ is not } (\Sigma, \Delta)\text{-diagnosable}.$$

Moreover a trace based definition of (Σ, Δ)-diagnosability can be stated as: A is (Σ, Δ)-diagnosable if and only if

$$\boldsymbol{\pi}_{/\Sigma}(\mathit{Faulty}^{tr}_{\geq \Delta}(A)) \cap \boldsymbol{\pi}_{/\Sigma}(\mathit{NonFaulty}^{tr}(A)) = \varnothing. \quad [4.1]$$

Indeed, if equation [4.1] is satisfied, we can define a diagnoser D by: $D(\rho) = 1$ if and only if $\boldsymbol{\pi}_{/\Sigma}(tr(\rho)) \in \boldsymbol{\pi}_{/\Sigma}(\mathit{Faulty}^{tr}_{\geq \Delta}(A))$. If equation [4.1] is not satisfied, we can find two runs ρ_1 and ρ_2 with $\rho_1 \in \mathit{Faulty}_{\geq \Delta}(A)$, $\rho_2 \in \mathit{NonFaulty}(A)$ and such that $\boldsymbol{\pi}_{/\Sigma}(tr(\rho_1)) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho_2))$. Thus no (Σ, Δ)-diagnoser can exist because it should announce at the same time 1 and 0. This gives a necessary and sufficient condition for non-diagnosability and thus diagnosability:

$$A \textit{ is not } \text{diagnosable} \iff \forall \Delta \in \mathbb{N}, \begin{cases} \exists \rho_1 \in \mathit{Faulty}_{\geq \Delta}(A) \\ \exists \rho_2 \in \mathit{NonFaulty}(A) \\ \textit{such that } \boldsymbol{\pi}_{/\Sigma}(tr(\rho_1)) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho_2)). \end{cases} \quad [4.2]$$

In the next section, we show how to solve Problems 4.1–4.4 for discrete event systems. In section 4.5 we extend the solutions for systems given by timed automata.

4.4. Fault Diagnosis for Discrete Event Systems

In this section we review the main results about diagnosability of discrete-event systems (DES) introduced in [SAM 95, SAM 96] and give an alternative proof of Theorem 4.3 which originally appeared in [CAS 08]. In this framework, we assume to have a formal model of the system to observe. This model defines the runs of the system including faulty runs.

4.4.1. *Discrete Event Systems for Fault Diagnosis*

We consider here that the DES is given by a finite automaton (with no clock) $A = (Q, q_0, \Sigma_{\tau,f}, \rightarrow)$ where Q is a finite set of states, and $\rightarrow \subseteq Q \times \Sigma_{\tau,f} \times Q$. A

finite automaton is a particular TA with $X = \varnothing$. Consequently guards and invariants are vacuously true and time elapsing transitions do not exist. Notice that the language accepted by such DES is prefix closed[1].

A run from ℓ_0 of the finite automaton A is thus a sequence of the form:

$$\varrho \quad = \quad \ell_0 \xrightarrow{a_1} \ell_1 \xrightarrow{a_2} \ell_2 \quad \cdots \quad \ell_j \quad \cdots \quad \xrightarrow{a_n} \ell_n \cdots$$

where for each $i \geq 0$, $(\ell_i, a_{i+1}, \ell_{i+1}) \in E$. In this case, the duration of a run ϱ is the number of steps (including τ-steps) of ϱ: if ϱ is finite and ends in ℓ_n, *Duration*$(\varrho) = n$ and otherwise *Duration*$(\varrho) = \infty$.

Definitions of traces and languages, and k-faulty runs given in section 4.3 extend straighforwardly using the duration function defined above.

REMARK 4.2 .– Using a timed automaton where discrete actions are separated by one time unit is not equivalent to using a finite automaton when solving a fault diagnosis problem. For instance, a timed automaton can generate the timed words $1.f.1.a$ and $1.\tau.1.\tau.1.a$. In this case, it is 1-diagnosable: after reading the timed word $2.a$ we announce a fault. If we do not see the 1-time unit durations, the timed words $f.a$ and $\tau^2.a$ give the same observation. And thus it is not diagnosable if we cannot measure time. Using a timed automaton where discrete actions are separated by one time unit gives to the diagnoser the ability to count/measure time and this is not equivalent to the fault diagnosis problem for untimed systems.

Moreover we assume that the automaton A is such that every faulty run of length n can be extended to a run of length $n + 1$; this assumption simplifies the proofs and if A does not satisfy it, it is easy to add τ loops to deadlock states of A to ensure it holds. It does not modify the observation made by the external observer and thus does not modify the diagnosability status of A.

4.4.2. *Checking Δ-Diagnosability and Diagnosability*

4.4.2.1. *Checking Δ-Diagnosability*

To check Problem 4.1 we have to decide whether there is a $(\Delta + 1)$-faulty run ρ_1 and a non-faulty run ρ_2 that give the same observations when projected on Σ.

An easy way to do this is to build a finite automaton $\mathcal{B}$ which accepts exactly those runs, and check whether $\mathcal{L}(\mathcal{B})$ is empty or not.

1. L is prefix closed if $\forall w \in \Sigma^*$, if there is some $w' \in \Sigma^\infty$ s.t. $w.w' \in L$ then $w \in L$.

Let $A_1 = (Q \times \{-1, 0, 1, \cdots, \Delta, \Delta+1\}, (q_0, -1), \Sigma_\tau, \rightarrow_1)$ be the automaton with $\rightarrow_1$ defined by:

– $(q, n) \xrightarrow{\lambda}_1 (q', n)$ if $q \xrightarrow{\lambda} q'$ and $n = -1$ and $\lambda \in \Sigma \cup \{\tau\}$;

– $(q, n) \xrightarrow{\lambda}_1 (q', \min(n+1, \Delta+1))$ if $q \xrightarrow{\lambda} q'$ and $n \geq 0$ and $\lambda \in \Sigma \cup \{\tau\}$;

– $(q, n) \xrightarrow{\tau}_1 (q', \min(n+1, \Delta+1)$ if $q \xrightarrow{f} q'$.

Let $A_2 = (Q, q_0, \Sigma_\tau, \rightarrow_2)$ with: $q \xrightarrow{\lambda}_2 q'$ if $q \xrightarrow{\lambda} q'$ and $\lambda \in \Sigma \cup \{\tau\}$. Define $\mathcal{B} = A_1 \times A_2$ with the final states $F_\mathcal{B}$ of $\mathcal{B}$ given by: $F_\mathcal{B} = \{((\ell, \Delta+1), \ell') \mid (\ell, \ell') \in Q \times Q\}$. We let $R_\mathcal{B} = \varnothing$. It is straightforward to see that

THEOREM 4.2 .– *A is Δ-diagnosable if and only if $\mathcal{L}^*(\mathcal{B}) = \varnothing$.*

As language emptiness for $\mathcal{B}$ amounts to reachability checking, it can be done in linear time in the size of $\mathcal{B}$. Still strictly speaking, the automaton $\mathcal{B}$ has size $(\Delta + 2) \cdot |A|^2$ which is exponential in the size of the inputs of the problem A and Δ because Δ is given in binary. Thus Problem 4.1 can be solved in EXPTIME. However, as storing Δ requires only polynomial space Problem 4.1 is in PSPACE.

4.4.2.2. *Checking Diagnosability*

To check whether A is diagnosable, we build a synchronized product $A_1 \times A_2$, s.t. A_1 behaves exactly as A but records in its state whether a fault has occurred, and A_2 behaves like A without the faulty runs as before. It is then as if $\Delta = 0$ in the previous construction. More precisely we have $A_1 = (Q \times \{0, 1\}, (q_0, 0), \Sigma_\tau, \rightarrow_1)$ with:

– $(q, n) \xrightarrow{l}_1 (q', n)$ if $q \xrightarrow{l} q'$ and $l \in \Sigma \cup \{\tau\}$;

– $(q, n) \xrightarrow{\tau}_1 (q', 1)$ if $q \xrightarrow{f} q'$, (n is set to 1 after a fault);

and $A_2 = (Q, q_0, \Sigma_\tau, \rightarrow_2)$ with: $q \xrightarrow{l}_2 q'$ si $q \xrightarrow{l} q'$ and $l \in \Sigma \cup \{\tau\}$. Let $A_1 \times A_2$ be the synchronized product of A_1 and A_2 (recall they only synchronise on common actions except τ, see Definition 4.2).

Assume we have a predicate A_1*Move*(t) (resp. A_2*Move*) which is true when A_1 (resp. A_2) participates in a transition t of the product $A_1 \times A_2$. For instance for A_1-moves, and $t : (s_1, s_2) \xrightarrow{\lambda} (s_1', s_2')$, A_1*Move*(t) is true if either $\lambda \in \Sigma$ or $s_1 \xrightarrow{\tau} s_1'$ and $s_2 = s_2'$. Given a run $\rho \in$ *Runs*$(A_1 \times A_2)$, we let $\rho_{|1}$ be the run of A_1 which consists of the sequence of transitions $t_1 t_2 \cdots t_k$ that satisfy A_1*Move*$(\cdot)$, and $\rho_{|2}$ be the run of A_2 which consists of the sequence of transitions $t_1' t_2' \cdots t_l'$ that satisfy A_2*Move*$(\cdot)$. By definition of $A_1 \times A_2$, we have $\boldsymbol{\pi}_{/\Sigma}(tr(\rho_{|1})) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho_{|2}))$.

Let *Runs*$_{\geq k}(A_1 \times A_2)$ be the set of runs in $A_1 \times A_2$ s.t. a (faulty) state $((q_1, 1), q_2)$ is followed by at least k A_1-actions. Then we have the following lemmas:

LEMMA 4.1 .– *Let* $\rho \in$ *Runs*$_{\geq k}(A_1 \times A_2)$. *Then* $\rho_{|1} \in$ *Faulty*$_{\geq k}(A)$ *and* $\rho_{|2} \in$ *NonFaulty*(A). *Moreover* $\boldsymbol{\pi}_{/\Sigma}(tr(\rho_{|1})) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho_{|2}))$.

LEMMA 4.2 .– *Let* $\rho_1 \in$ *Faulty*$_{\geq k}(A)$ *and* $\rho_2 \in$ *NonFaulty*(A) *s.t.* $\boldsymbol{\pi}_{/\Sigma}(tr(\rho_1)) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho_2))$. *Then there is some* $\rho \in$ *Runs*$_{\geq k}(A_1 \times A_2)$ *s.t.* $\rho_{|1} = \rho_1$ *and* $\rho_{|2} = \rho_2$.

PROOF 4.1 .– By definition of $A_1 \times A_2$, it suffices to notice that a run in *Runs*$_{\geq k}(A_1 \times A_2)$ can be split into a run of A_1 and a run of A_2 having the same projection on Σ. □

To check Problem 4.2, we build an extended automaton $\mathcal{B}$ s.t. $\mathcal{B}$ accepts an infinite word if and only if A is not diagnosable.

$\mathcal{B}$ is the extended version of $A_1 \times A_2$ built as follows: we add a boolean variable z that is set to 0 in the initial state of $\mathcal{B}$. An extended state of $\mathcal{B}$ is a pair (s, z) with s a state of $A_1 \times A_2$. Whenever A_1 participates in an $A_1 \times A_2$-action, z is re-set to 1, and when only A_2 makes a move in $A_1 \times A_2$, z is set 0. We denote $\longrightarrow_{\mathcal{B}}$ the new transition relation between extended states. The Büchi automaton $\mathcal{B}$ is the tuple $((Q \times \{0,1\}) \times Q \times \{0,1\}, ((q_0, 0), q_0, 0), \Sigma^\tau, \longrightarrow_{\mathcal{B}}, \varnothing, R_{\mathcal{B}})$ defined by:

– $(s, z) \xrightarrow{\lambda}_{\mathcal{B}} (s', z')$ if (i) there exists a transition $t : s \xrightarrow{\lambda}_{1,2} s'$ in $A_1 \times A_2$, and (ii) $z' = 1$ if A_1*Move*(t) and $z' = 0$ otherwise;

– $R_{\mathcal{B}} = \{((q, 1), q', 1) \mid (q, q') \in Q \times Q\}$ and $F_{\mathcal{B}} = \varnothing$.

$\mathcal{B}$ accepts a langage of infinite words $\mathcal{L}(\mathcal{B}) = \mathcal{L}^\omega(\mathcal{B}) \subseteq \Sigma^\omega$ and the following theorem holds:

THEOREM 4.3 .– $\mathcal{L}^\omega(\mathcal{B}) \neq \varnothing \iff$ *A is not* Σ*-diagnosable*.

PROOF 4.2 .–
Proof of $\Longrightarrow$. Assume $\mathcal{L}^\omega(\mathcal{B}) \neq \varnothing$. Let $\rho \in \mathcal{L}^\omega(\mathcal{B})$: ρ has infinitely many faulty states and infinitely many A_1-actions because of the definition of $R_{\mathcal{B}}$ which implies that $z = 1$ infinitely often. Let $k \in \mathbb{N}$. Let $\rho[i_k]$ be a finite prefix of ρ that contains more than k A_1-actions after the first faulty state in ρ (for any k this i_k exists because ρ contains infinitely many A_1-actions). $\rho[i_k] \in$ *Runs*$_{\geq k}(A_1 \times A_2)$ and by Lemma 4.1, it follows that $\rho[i_k]_{|1} \in$ *Faulty*$_{\geq k}(A)$ and $\rho[i_k]_{|2} \in$ *NonFaulty*(A) and $\boldsymbol{\pi}_{/\Sigma}(tr(\rho[i_k]_{|1})) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho[i_k]_{|2}))$. Thus equation (4.2) is satisfied for any $k \in \mathbb{N}$ and A is not diagnosable.

Proof of $\Longleftarrow$. Conversely assume A is not Σ-diagnosable. Then by equation (4.2) and Lemma 4.2, for any $k \in \mathbb{N}$, there is a run $\rho_k \in$ *Runs*$_{\geq k}(A_1 \times A_2)$. Consider the tree

which is the unfolding of $A_1 \times A_2$. Restrict this tree to branches where there is only a finite number of consecutive τ-actions fired by A_2. This is not a real restriction, because after a finite number of such moves, A_2 will enter an already visited state. In this tree, there is an infinite number of paths $(\rho_k, k \in \mathbb{N})$, which contain a faulty state of $A_1 \times A_2$. Thus an infinite number of them must contain the same first faulty state. Without loss of generality we can assume that all the ρ_k have the same first faulty state $((q_1, 1), q_2)$. It follows that $((q_1, 1), q_2)$ is the source of a tree having an infinite number of nodes and thus, by König's Lemma, the source of an infinite branch. On this infinite branch we have only a bounded number of consecutive τ actions fired by A_2, and thus it must contain an infinite number of A_1 moves. We then have an infinite path containing infinitely many A_1 moves the source of which is a faulty state which implies that $\mathcal{L}^\omega(\mathcal{B}) \neq \varnothing$. □

REMARK 4.3 .– The set $R_\mathcal{B}$ involves z because A_1 has to move infinitely often. This corresponds to logical time divergence. Without this condition, it could be that A_1 refuses to proceed, and thus prevents the logical time to progress. If A_2 then fires an infinite number of transitions τ, the automaton would be declared not diagnosable. Still they could be no arbitrary large faulty run on A and this does not match Definition [4.2].

Theorem 4.3 gives a procedure to check whether an automaton A is Σ-diagnosable or not and thus to decide Problem 4.2:

COROLLARY 4.1 .– *Problem 4.2 can checked in polynomial time* $O(|A|^2)$.

PROOF 4.3 .– Büchi emptiness can be decided in polynomial (linear) time. For an automaton H with n states and m transitions, deciding whether $\mathcal{L}^\omega(H) = \varnothing$ can de done in $O(n + m)$. As the size of $\mathcal{B}$ is $4 \times |Q|^2$, Problem 4.2 can be checked in $O(|Q|^2)$. □

In the case of a finite number of *failure types*, it suffices to design $A_1 \times A_2$ for each type of faults [YOO 02] and check for diagnosability. Thus checking diagnosability with multiple type of faults is also polynomial.

Polynomial algorithms for checking diagnosability (Problem 4.2) were already reported in [JIA 01, YOO 02]. In these two papers, the plant cannot have unobservable loops i.e, loops that consist of τ actions. Our algorithm does not have this limitation (we even may have to add τ loops to ensure that each faulty run can be extended). Note also that in [JIA 01, YOO 02], the product construction is symmetric in the sense that A_2 is a copy of A as well. Our A_2 does not contain the f transitions, which makes no difference complexity-wise, but in practice this can be useful to reduce the size of the product.

Moreover, reducing Problem 4.2 to emptiness checking of Büchi automata is interesting in many respects:

– the proof of Theorem 4.3 is easy and short; algorithms for checking Büchi emptiness are well-known and correctness follows easily as well;

– this also implies that standard tools from the *model-checking/verification* community can be used to check for diagnosability. There are very efficient tools to check for Büchi emptiness (e.g., SPIN [HOL 05]). Numerous algorithms, like *on-the-fly* algorithms [COU 05] have been designed to improve memory/time consumption (see [SCH 05] for an overview). Also when the DES is not diagnosable a counter-example is provided by these tools. The input languages (like PROMELA for SPIN) that can be used to specify the DES are more expressive than the specification languages of some dedicated tools[2] like DESUMA/UMDES [RIC 06] (notice that the comparison with DESUMA/UMDES concerns only the diagnosability algorithms; DESUMA/UMDES can perform a lot more than checking diagnosability).

From Theorem 4.3, one can also conclude that diagnosability amounts to bounded diagnosability: indeed if A is diagnosable, there can be no accepting cycles of faulty states in $\mathcal{B}$; in this case there cannot be a faulty run of length more than $2 \cdot |Q|^2$ in $\mathcal{B}$. Thus Problem 4.2 reduces to a particular instance of Problem 4.1 as was already stated in [YOO 02]:

THEOREM 4.4 .– *A is Σ-diagnosable if and only if A is $(\Sigma, 2 \cdot |Q|^2)$-diagnosable.*

This appeals from some final remarks on the algorithms we should choose to check diagnosability: for the particular case of $\Delta = 2 \cdot |A|^2$, solving Problem 4.1 (a reachability problem) can be done in time $2 \cdot |A|^2 \cdot |A|^2$ i.e, $O(|A|^4)$ whereas solving directly Problem 4.2 as a Büchi emptiness problem can be done in $O(|A|^2)$. Thus the extra-cost of using a reachability algorithm vs. a Büchi emptiness checking algorithm is still reasonable.

4.4.3. *Computation of the Maximum Delay*

To compute the least k s.t. A is (Σ, k)-diagnosable, we can proceed as follows: if A is Σ-diagnosable, there cannot be any run in $\mathcal{B}$ which is (1) faulty and (2) contains an infinite number of A_1 moves. Thus any run of $\mathcal{B}$ starting in a faulty state is followed by a bounded number of A_1 moves: if not, there would be a faulty run with an infinite number of A_1 moves and A would not be diagnosable. Let k be the maximum number of A_1 transitions that can follow a faulty state of $\mathcal{B}$. Then A is $(\Sigma, k+1)$-diagnosable:

2. UMDES was the only publicly available tool which could be found by a Google search.

otherwise, there would a faulty run with $k+1$ actions of A_1. Moreover A is not (Σ, k)-diagnosable because there is a faulty run in $\mathcal{B}$ which has k A_1 steps; the definition of A_2 and $\mathcal{B}$ implies (Lemma 4.1) that A has a k-faulty run and a non-faulty run with the same observation. Altogether this means that the maximum delay to detect faults is $k+1$. Let $\mathit{Max}_d(A)$ be the value of this maximum delay. Computing $\mathit{Max}_d(A)$ amounts to finding the maximum number of A_1 actions that can follow a faulty (reachable) state of $\mathcal{B}$. This can be computed in polynomial time. Solving Problem 4.3 can also be done by a binary search solving iteratively Δ-diagnosability problems starting with $\Delta = 2 \cdot |A|^2$. As there are at most $O(\log |A|)$ Δ-diagnosability problems to solve we obtain:

THEOREM 4.5 .– *Problem 4.3 can be solved in polynomial time* $O(\log |A| \cdot |A|^4)$.

Using a different approach, Problem 4.3 was reported to be solvable in $O(|A|^3)$ in [YOO 03]. The algorithm is based on the computation of longest paths in a weighted automaton built from $A_1 \times A_1 \times A_2$.

4.4.4. *Synthesis of a Diagnoser*

To compute a witness diagnoser when A is Σ-diagnosable, we just have to determinise A_1. To do this, we use the classical *subset construction*. Let F be the set of faulty states of A_1 i.e, $F = Q \times \{1\}$.

Define $E_a(q)$ to be the set of states q' s.t. there is a path in A_1 from q to q' involving only a finite number of τ steps followed by a:

$$E_a(q) = \{q'' \mid q \xrightarrow{\tau^*} q' \xrightarrow{a} q''\}. \qquad \text{[E]}$$

We then define the deterministic automaton $\mathit{Det}(A_1) = (S, s_0, \Sigma, \delta', F', \varnothing)$ by:
– $S = 2^Q$;
– $s_0 = \{q' \in Q \times \{0,1\} \mid (q_0, 0) \xrightarrow{\tau^*} q'\}$;
– for $s \in S, a \in \Sigma, \delta'(s,a) = \cup_{x \in s} E_a(x)$;
– $F' = \{s \in S \mid s \subseteq F\}$.

$\mathit{Det}(A_1)$ is deterministic by construction. Let w be a word in $\boldsymbol{\pi}_{/\Sigma}(\mathcal{L}^*(A))$, $\mathit{last}(w)$ is thus determined in a unique way.

To obtain a $(\Sigma, \mathit{Max}_d(A))$-diagnoser D for A, we let $D(w) = 1$ if $\mathit{last}(w) \in F'$ and 0 otherwise. This diagnoser detects faults at most $\mathit{Max}_d(A)$ discrete steps after they occurred. It has size exponential in the size of A.

THEOREM 4.6 .– *Problem 4.4 can be solved in time* $O(2^{|A|})$.

4.5. Fault Diagnosis for Timed Systems

In this section, we show how to decide wether a systems given as a timed automaton is diagnosable or not. For the synthesis problem, we take into account the *resources* (clocks and integer constants) that a diagnoser can use. Indeed, there is one major difference between the fault diagnosis problems in discrete time and dense time: if a timed automaton is diagnosable, there is not always a diagnoser which is a (deterministic) timed automaton. This is in contrast to the discrete time case where a finite automaton diagnoser (at most of exponential size) can be built when a system is diagnosable. The results of sections 4.5.1 to 4.5.4 are based on [TRI 02] and the results of section 4.5.5 on [BOU 05, CHE 04].

Throughout this section, we assume $A = (L, l_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$ is a timed automaton with faults.

4.5.1. *Checking Δ-Diagnosability*

Let t be a fresh clock not in X. Let $A_1(\Delta) = ((L \times \{0,1\}) \cup \{Bad\}, (l_0, 0), X \cup \{t\}, \Sigma_\tau, E_1, \mathit{Inv}_1)$ where Bad is a fresh location not in L, and:

– $((\ell, n), g, \lambda, r, (\ell', n)) \in E_1$ if $(\ell, g, \lambda, r, \ell') \in E$, $\lambda \in \Sigma \cup \{\tau\}$;

– $((\ell, 0), g, \tau, r \cup \{t\}, (\ell', 1)) \in E_1$ if $(\ell, g, f, r, \ell') \in E$;

– $\mathit{Inv}_1((\ell, 0)) = \mathit{Inv}(\ell), \mathit{Inv}_1((\ell, 1)) = \mathit{Inv}(\ell) \wedge t \leq \Delta$ and $\mathit{Inv}_1(Bad) = \top$;

– for $\ell \in L$, $((\ell, 1), t \geq \Delta, \tau, \varnothing, Bad) \in E_1$

and $A_2 = (L, l_0, X_2, \Sigma_\tau, E_2, \mathit{Inv}_2)$ with:

– $X_2 = \{x_2 \mid x \in X\}$ (clocks of A are renamed);

– $(\ell, g_2, \lambda, r_2, \ell') \in E_2$ if $(\ell, g, \lambda, r, \ell') \in E$, $\lambda \in \Sigma \cup \{\tau\}$ with: g_2 is g where each clock x is replaced by its counterpart x_2; r_2 is r with the same renaming;

– $\mathit{Inv}_2(\ell)$ is $\mathit{Inv}(\ell)$ with the renaming of x by x_2.

Consider $A_1(\Delta) \times A_2$. A faulty state of $A_1(\Delta) \times A_2$ is a state of the form $(((\ell, 1), v), (\ell', v'))$ i.e, where the state of A_1 is faulty. Let $\mathit{Runs}_{\geq\Delta}(A_1(\Delta) \times A_2)$ be the runs of $A_1(\Delta) \times A_2$ s.t. a faulty state of A_1 is encountered and s.t. at least Δ time units have elapsed after this state. Then there are two runs, one Δ-faulty and one non-faulty which give the same observation. Moreover, because t is reset exactly when the first fault occurs, we have $t \geq \Delta$. Conversely, if a state of the form $(((\ell, 1), v), (\ell', v'))$ with $v(t) \geq \Delta$ is reachable, then there are two runs, one Δ-faulty and one non-faulty which give the same observation. Location Bad in A_1 is thus reachable exactly if A is not Δ-diagnosable. Let $\mathcal{D}$ be $A_1(\Delta) \times A_2$ with the final set of locations $F_\mathcal{D} = \{Bad\}$ and $R_\mathcal{D} = \varnothing$.

THEOREM 4.7 ([TRI 02]).– *A is Δ-diagnosable if and only if $\mathcal{L}^*(\mathcal{D}) = \varnothing$.*

Checking reachability of a location for timed automata is PSPACE-complete [ALU 94a]. More precisely, it can be done in linear time on the region graph. The size of the region graph of $\mathcal{D}$ is $(2 \cdot |L|^2 + |L|) \cdot (2|X| + 1)! \cdot 2^{2|X|+1} \cdot K^{2|X|} \cdot \Delta$ where K is the maximal constant appearing in A. It follows that:

COROLLARY 4.2 .– *Problem 4.1 can be solved in PSPACE for TA.*

4.5.2. *Checking Diagnosability*

As for the untimed case, we build an automaton $\mathcal{D}$, which is a special version of $A_1(\Delta) \times A_2$. Assume A_1 is defined as before omitting the clock t and the location Bad. In the timed case, we have to take care of the following real-time related problems [TRI 02]:

– some runs of A_2 might prevent time from elapsing from a given point in time. In this case, equation (4.1) cannot be satisfied but this is for a spurious reason: for Δ large enough, there will be no Δ faulty run in $A_1 \times A_2$ because A_2 will block the time. In this case we can claim that A is diagnosable but it is not realistic;

– a more tricky thing may happen: A_1 could produce a Zeno run[3] after a fault occurred. This could happen by firing infinitely many τ transitions in a bounded amount of time. If we declare that A is not diagnosable but the only witness run is a Zeno run, it does not have any physical meaning. Thus to declare that A is not diagnosable, we should find a non-Zeno witness run which is realizable, and for which time diverges.

To cope with the previous dense-time related problems we have to ensure that the two following conditions are met:

C_1**:** A_2 is *timelock-free* i.e, A_2 cannot prevent time from elapsing; this implies that every finite non-faulty run of A_2 can be extended in a time divergent run. We can assume that A_2 satisfies this property or check it on A_2 before checking diagnosability;

C_2**:** for A to be non-diagnosable, we must find an infinite run in $A_1 \times A_2$ for which time diverges.

C_2 can be enforced by adding a third timed automaton *Div*(x) and synchronizing it with $A_1 \times A_2$. Let x be a fresh clock not in X. Let *Div*$(x) = (\{0, 1\}, 0, \{x\}, E,$ *Inv*$)$ be the TA given in Figure 4.2.

3. A Zeno run is a run with infinitely many discrete steps the duration of which is bounded.

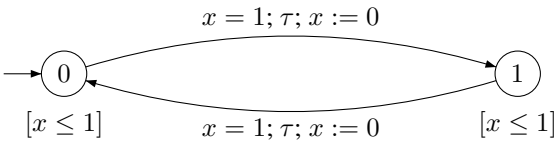


Figure 4.2. *Timed Automaton Div*(x)

If we use $F = \varnothing$ and $R = \{1\}$ for *Div*(x), any infinite accepted run is time divergent. Let $\mathcal{D} = (A_1 \times A_2) \times$ *Div*(x) with the sets $F_{\mathcal{D}} = \varnothing$ and $R_{\mathcal{D}}$ is the set of states where A_1 is in a faulty state and *Div*(x) is in location 1. The following theorem is the counterpart of Theorem 4.3 for timed automata:

THEOREM 4.8 ([TRI 02]).– *A is diagnosable if and only if $\mathcal{L}^{\omega}(\mathcal{D}) = \varnothing$.*

PROOF 4.4 .–
Proof of $\Longrightarrow$. Assume $\mathcal{L}^{\omega}(\mathcal{D}) \neq \varnothing$. Let $\rho \in \mathcal{L}^{\omega}(\mathcal{D})$. Because of the definition of $R_{\mathcal{D}}$, ρ is a time divergent run: $\forall \alpha \in \mathbb{N}$, there is a prefix $\rho[\alpha]$ of ρ such that *Duration*$(\rho[\alpha]) \geq \alpha$. Using Lemma 4.1 (extended for timed automata), there exist two runs $\rho_{\alpha,1} = \rho[\alpha]_{|1}$ and $\rho_{\alpha,2} = \rho[\alpha]_{|2}$ such that $\boldsymbol{\pi}_{/\Sigma}(tr(\rho_{\alpha,1})) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho_{\alpha,2}))$ and $\rho_{\alpha,1} \in$ *Faulty*$_{\geq\alpha}(A)$, $\rho_{\alpha,2} \in$ *NonFaulty*(A). For every α, equation [4.2] is satisfied and thus A is not Σ-diagnosable.

Proof of $\Longleftarrow$. Assume A is not Σ-diagnosable. Then, from Equation [4.2], for every $\alpha \in \mathbb{N}$, there are two runs $\rho_1 \in$ *Faulty*$_{\geq\alpha}(A)$, $\rho_2 \in$ *NonFaulty*(A), $\boldsymbol{\pi}_{/\Sigma}(tr(\rho_1)) = \boldsymbol{\pi}_{/\Sigma}(tr(\rho_2))$. From Lemma 4.2 (again extended to finite automata), the following property (P) holds: for every α, there exists[4] a run $\rho[\alpha] \in$ *Runs*$_{\geq\alpha}((A_1 \times A_2) \times$ *Div*$(x))$ such that $\rho[\alpha]_{|1} = \rho_1$ and $\rho[\alpha]_{|2} = \rho_2$. To replay the proof of Theorem 4.3, we use a property of the region graph introduced in Chapter 2. Let *RG*$(\mathcal{D})$ be the region graph of $\mathcal{D}$. This graph is finite and satisfies: ρ is a run of *RG*$(\mathcal{D})$ if and only if there is a *timed version* of ρ, $\widetilde{\rho}$ which is a run of $\mathcal{D}$. Property (P) above, implies that faulty behaviours of arbitrary length exist in the region graph *RG*$(\mathcal{D})$. As this graph is finite, from König's Lemma, there is an infinite run[5] in *RG*$(\mathcal{D})$ from a faulty state. Hence there is a timed version of this run which is an infinite execution in $\mathcal{D}$. And thus $\mathcal{L}^{\omega}(\mathcal{D}) \neq \varnothing$. □

Deciding whether $\mathcal{L}^{\omega}(A) \neq \varnothing$ for timed automata is PSPACE-complete [ALU 94a]. Thus deciding diagnosability is in PSPACE.

4. *Div*(x) does not constraint A_1 nor A_2 and every run of $A_1 \times A_2$ can be interleaved with a run of *Div*(x) to form a run of $(A_1 \times A_2) \times$ *Div*(x).

5. The reason is the same as for for the proof of Theorem 4.3.

The reachability problem for TA can be reduced to a diagnosability problem [TRI 02]. Let A be a TA on alphabet Σ and *End* a particular location of A. We want to check whether *End* is reachable in A. It suffices to build A' on the alphabet $\Sigma_{\tau,f}$ by adding to A the following transitions: $(\mathit{End}, \top, \lambda, \varnothing, \mathit{End})$ for $\lambda \in \{\tau, f\}$. Then: A' is not diagnosable if and only if *End* is reachable in A. It follows that:

THEOREM 4.9 ([TRI 02]).– *Problem 4.2 is PSPACE-complete for TA.*

4.5.3. *Computation of the Maximal Delay*

To compute the maximim delay, we can proceed as for the untimed case. Assume we have checked that A is Σ-diagnosable. We can then check wether A is (Σ, Δ)-diagnosable starting with $\Delta = 1 = 2^0$, and then $\Delta = 2^k$, increasing k, until A is $(\Sigma, 2^k)$-diagnosable (which is bound to happen as we have previously checked that A is Σ-diagnosable). Let k_0 be the smallest k such that A is $(\Sigma, 2^{k_0})$-diagnosable. We can then do a binary search for the maximal delay k in the interval $[2^{k_0-1}+1, 2^{k_0}]$.

Nevertheless, we can be more effective because we draw another conclusion from the proof of Theorem 4.8: if a TA A is diagnosable, there cannot be any cycle with faulty states in the region graph of $A_1 \times A_2 \times \mathit{Div}(x)$. Indeed, otherwise, by Theorem 4.1, there would be a non-Zeno word in $A_1 \times A_2 \times \mathit{Div}(x)$ itself[6]. Let $\alpha(A)$ denote the size of the region graph $\mathit{RG}(A_1 \times A_2 \times \mathit{Div}(x))$. If A is diagnosable, then (P_1): a faulty state in $\mathit{RG}(A_1 \times A_2 \times \mathit{Div}(x))$ can be followed by at most $\alpha(A)$ (faulty) states. Notice that a faulty state cannot be followed by a state (s, r) where r is an unbounded region of A, as this would give rise to a non-Zeno word in $A_1 \times A_2 \times \mathit{Div}(x)$. Hence ($P_2$): all the regions following a faulty state in $\mathit{RG}(A_1 \times A_2 \times \mathit{Div}(x))$ are bounded. As the amount of time which can elapse within a region is less than 1 time unit[7], this implies that the duration of the longest faulty run in $A_1 \times A_2 \times \mathit{Div}(x)$ is less than $\alpha(A)$. Actually as every other region is a *singular region*[8], it must be less than $(\alpha(A)/2) + 1$. Thus we obtain the following result:

THEOREM 4.10 .– *A is diagnosable if and only if A is $(\alpha(A)/2 + 1)$-diagnosable.*

As diagnosability can be reduced to Δ-diagnosability for timed automata:

---

6. Note that this is true because we add the automaton $\mathit{Div}(x)$. Otherwise an infinite run in the region graph of a TA does not imply a time divergent run in the TA A itself.

7. We assume the constants are integers.

8. A singular region is a region in which time elapsing is not possible e.g., defined by $x = 0 \wedge y \geq 1$.

COROLLARY 4.3 .– *Problem 4.1 is PSPACE-complete for TA.*

PROOF 4.5 .– The size of the binary encoding of $(\alpha(A)/2+1)$ is polynomial in the size of A. □

Although Problem 4.1 and Problem 4.2 are PSPACE-complete for timed automata, the price to pay to solve Problem 4.2 as a reachability problem is much higher than solving it as a Büchi emptiness problem: indeed the size of the region graph of $A_1(\alpha(A)/2+1)\times A_2$ is the square of the size of the region graph of $A_1 \times A_2 \times \mathit{Div}(x)$ which is already exponential in the size of A. Time-wise this means a blow up from 2^n to 2^{n^2} which is not negligible as in the discrete case.

4.5.4. *Synthesis of a Diagnoser*

In case A is Σ-diagnosable, we define a diagnoser as a mapping which performs a state estimate of A after a timed word w is read by A. Indeed, we cannot determinize A (as for the untimed case) because timed automata cannot always be determinized [ALU 94a]. Moreover, testing whether a timed automaton is determinizable is undecidable [FIN 05, TRI 06].

Note that for classes of determinizable timed automata like ERA (see [ALU 94b]), we can do the same construction as for the untimed case.

Nevertheless, there are some non deterministic timed automata which do not admit any deterministic timed automaton as a diagnoser. An example automaton, C, is given in Figure 4.3. The faulty runs of C are of the form $\delta.a.t$ with $\delta \in \mathbb{N}$ and $t \geq 0$, and the non-faulty runs are of the form $\delta.a.t$ with $\delta \notin \mathbb{N}$. Equation [4.1] is thus satisfied for every delay Δ and C is diagnosable. We can build a 0-diagnoser for C which is the mapping $D : \mathit{TW}^*(\{a\}) \rightarrow \{0,1\}$ given by $D(\delta.a.t) = 1$ if $\delta \in \mathbb{N}$ and $D(\delta.a.t) = 0$ otherwise. But there is no deterministic timed automaton which can accept the language $\delta.a$ with $\delta \in \mathbb{N}$ (Cf. [BER 98]). To build a diagnoser in the general case we proceed as follows. The first stage is to add to the states of A a bit which indicates whether a fault occurred or not. This amounts to building A_1 as defined earlier in this section. Then A has tow types of states: faulty states with the bit set to 1 and non-faulty states with the bit set to 0. We then define a mapping D which computes the (symbolic) set of states A can be in after reading a timed word w. This set of states is updated after each occurrence of a new event (δ, a). If the set of states A can be in after reading a timed word w contains only faulty states, then we can announce a fault. If it contains both faulty and non faulty states it is too early to decide wether a fault really occurred.

The formal construction of such a mapping is detailed in [TRI 02]. A diagnoser in the general case is thus an algorithm (a Turing machine) which computes sets of

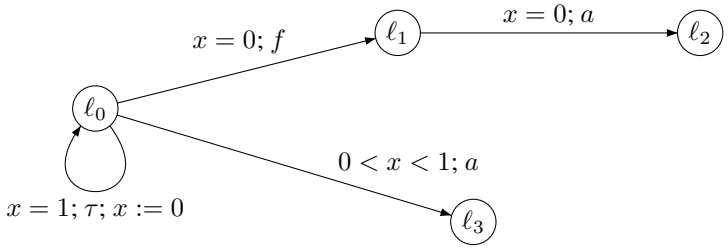


Figure 4.3. *Automaton C (taken from [BOU 05])*

symbolic states. The worst case is obtained when each state is a node of the region graph. Thus computing the set of states A can be in can be done in time exponential in the size of A.

If we have to compute this set of states in real-time, this can be prohibitive. It is thus sensible to address the problem of diagnosability with timed automata, where we compute off-line a diagnoser which is a timed automaton. This is the purpose of the next section.

4.5.5. *Fault Diagnosis with Deterministic Timed Automata*

The fault diagnosis problem with deterministic timed automata (DTA) was introduced and solved in [BOU 05, CHE 04]. A closely related problem was studied in [KRI 04b] for the synthesis of *timed testers* which are deterministic timed automata.

We recall that a timed automaton A is deterministic if there is no τ labelled transition in A, and if, whenever (ℓ, g, a, r, ℓ') and $(\ell, g', a, r', \ell'')$ are transitions of A, $g \wedge g' \equiv \bot$. A is *complete* if from each state (ℓ, v), and for each action a, there is a transition (ℓ, g, a, r, ℓ') such that $v \models g$. Let $\mathcal{C}$ be a class of deterministic timed automata. We note DTA the class of deterministic timed automata.

In the sequel, DTA diagnosers are used as language acceptors and we do not need any invariants on locations. We let $Inv_\top$ (abusing notations) be the invariant which assigns $\top$ to any location of a DTA.

DEFINITION 4.4 ($\mathcal{C}$-DIAGNOSER).– Let A be a timed automaton over $\Sigma_{\tau,f}$ and $\Delta \in \mathbb{N}$. A (Σ, Δ)-$\mathcal{C}$-diagnoser for A is a complete and deterministic timed automaton $\Theta = (N, n_0, C, \Sigma, E_\Theta, Inv_\top, F_\Theta, \varnothing)$ in the class $\mathcal{C}$ such that:

– for each $\rho \in NonFaulty(A)$, $last(\boldsymbol{\pi}_{/\Sigma}(tr(\rho))) \notin F_\Theta$;

– for each $\rho \in Faulty_{\geq\Delta}(A)$, $last(\boldsymbol{\pi}_{/\Sigma}(tr(\rho))) \in F_\Theta$. □

In other words, Θ accepts the Δ-faulty words, but not the non-faulty words. A is (Σ, Δ)-$\mathcal{C}$-diagnosable if there is a (Σ, Δ)-$\mathcal{C}$-diagnoser for A. A is Σ-$\mathcal{C}$-diagnosable if there exists a $\Delta \in \mathbb{N}$ such that A is (Σ, Δ)-$\mathcal{C}$-diagnosable. The fault diagnosis problem using a diagnoser in $\mathcal{C}$ can be formally stated as follows:

PROBLEM 4.5 ($\mathcal{C}$-DIAGNOSABILITY).–
INPUT: A TA $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$.
PROBLEM: Is A Σ-$\mathcal{C}$-diagnosable?

In this problem, we are looking for a delay Δ and a deterministic timed automaton in the class $\mathcal{C}$ which accepts the Δ-faulty words.

EXAMPLE 4.2 .– The timed automaton $\mathcal{A}(3)$ of Figure 4.1 is 3-diagnosable and there is a deterministic timed automaton which is a 3-diagnoser. An example is the automaton of Figure 4.4: when location 3 is reached we announce a fault. Actually this automaton is in location 3 if and only if a fault occurred less than 3 time units ago and was followed by a b. $\mathcal{A}(3)$ fires a transition on the occurrences of observable events (a and b) in a deterministic manner. When it enters location 3 a fault has certainly occurred. Thus $\mathcal{A}(3)$ is 3-DTA-diagnosable.

Problem 4.5 in its general version is still open. A solution for a simpler version has been proposed in [BOU 05] when the resources of the automata of the class $\mathcal{C}$ are fixed and when the maximum delay Δ to announce a fault is given. The resources of a timed automaton are: the number of clocks it can use, the constants the clocks can be compared against; this includes the maximal constant and the granularity of the constants. For instance, the ressources of a class of timed automata can be given as a triple $(\{x\}, 2, \frac{1}{3})$ with the meaning:

– only one clock (the name x is unimportant) can be used to measure time;
– the maximal constant that can be used is 2;
– x can only be compared against multiples of $\frac{1}{3}$ in the interval $[-2, 2]$.

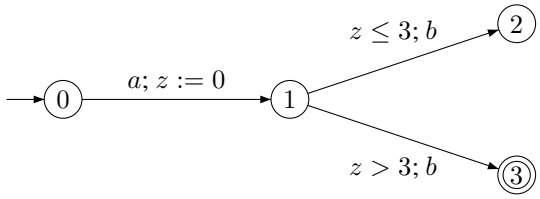


Figure 4.4. *A DTA Diagnoser for* $\mathcal{A}(3)$

EXAMPLE 4.3 .– The automaton $\mathcal{A}(3)$ is diagnosable with the resource $(\{z\}, 3, 1)$.

Let $\mu = (Y, \max, \frac{1}{m})$ be a resource[9], with Y a finite set of clocks (disjoint from X), $\max \in \mathbb{N}$ and $m \in \mathbb{N}^*$. A timed automaton *of resource* μ is an automaton which uses clocks in Y, those clocks are compared to constants the absolute values of which are less than $\max$ and which are multiples of $\frac{1}{m}$. We let DTA_μ be the set of determiistic timed automata of resource μ. In the sequel we use the term « regions of μ » for the set of regions of granularity μ and also the clock constraints which define these regions. The DTA diagnosis problem with fixed resource μ is the following:

PROBLEM 4.6 (DTA_μ-DIAGNOSABILITY).–
INPUT: A TA $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$ and a resource μ.
PROBLEM: Is A Σ-DTA_μ-diagnosable?

The previous problem is still open: the maximum delay Δ to annouce a fault is not an input of this problem. A simpler problem is thus:

PROBLEM 4.7 (Δ-DTA_μ-DIAGNOSABILITY).–
INPUT: A TA $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$ and a delay $\Delta \in \mathbb{N}$.
PROBLEM: Is A (Σ, Δ)-DTA_μ-diagnosable?

A solution to Problem 4.7 has been proposed in [BOU 05]. Before giving the sketch of this solution, we focus on the following problem:

PROBLEM 4.8 (CHECKING DTA-DIAGNOSABILITY).–
INPUT: A TA $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$, a DTA $\Theta = (N, n_0, C, \Sigma, E_\Theta, \mathit{Inv}_\top, F_\Theta, \varnothing)$ and $\Delta \in \mathbb{N}$.
PROBLEM: Is Θ a (Σ, Δ)- diagnoser for A?

To solve the previous problem, we consider the automaton A_1 of paragraph 4.5.1. We let $A(\Delta) = A_1(\Delta)$. In $A(\Delta)$ we can distinguish the Δ-faulty locations, $L_{\Delta f} = \{Bad\}$, and the non-faulty locations, $L_{\neg f}$, of the form $(\ell, 0)$.

To check that Θ is a (Σ, Δ)-diagnoser for A amounts to verifying that each reachable state $((q, v), (n, v'))$ of $A(\Delta) \times \Theta$ satisfies:

1) $q \in L_{\Delta f} \implies n \in F_\Theta$;

2) $n \in F_\Theta \implies q \notin L_{\neg f}$ (i.e, $q = (\ell, k)$ and $k = 1$).

9. Such a triple is also called a *granularity* in the literature.

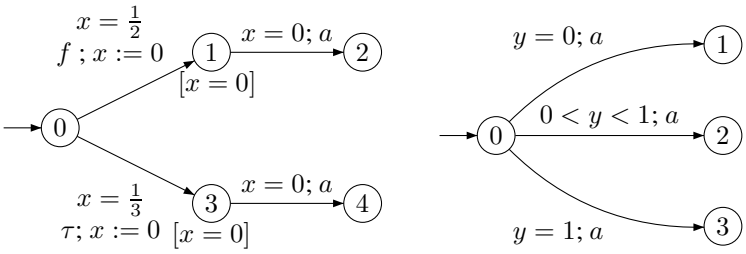
This can be done in PSPACE using the region graph of $A(\Delta) \times \Theta$.

REMARK 4.4 .– In $A(\Delta)$, a Δ-faulty run does not necessarily ends in Bad. However, if it ends in $((\ell, 1), v)$, we must have $v(t) = \Delta$, and thus Bad is reachable in one (immediate) discrete step. In the su-ynchronized product $A(\Delta) \times \Theta$, Θ cannot change its location on this last move of $A(\Delta)$, and this is why condition 1 above is sufficient.

We can now consider a more liberal version of Problem 4.8: let $\Theta^- = (N, n_0, C, \Sigma, E_\Theta, \mathit{Inv}_\top)$ be a DTA, for which the accepting locations can be chosen. Is there a set $F_\Theta \subseteq N$ such that A is (Σ, Δ)-diagnosable with $\Theta = (N, n_0, C, \Sigma, E_\Theta, \mathit{Inv}_\top, F_\Theta, \varnothing)$?

EXAMPLE 4.4 .– As pointed out previously, the automaton of Figure 4.4 can be configured to be a diagnoser for $\mathcal{A}(3)$: it suffices to set the accepting locations to $\{3\}$. In the example of Figure 4.5, we want to diagnose the faults of automaton J of Figure 4.5(a). The automaton O^- of Figure 4.5(b) cannot be configured to diagnose J. Indeed, it cannot make any difference between the runs with traces $\delta.a$ with $0 < \delta < 1$ and thus cannot distinguish faulty and non faulty runs in J. Notice that J is diagnosable in the sense of Definition 4.3, and there is even a DTA which correctly diagnoses J: we just have to take an automaton of resource $(\{y\}, 1, \frac{1}{2})$.

As witnessed by the previous example, to decide whether a given automaton can be configured to be a diagnoser, we have to check whether it is not two "coarse": it must accept all the Δ-faulty words without accepting a non-faulty word.



(a) Automaton J (b) Automaton O^-

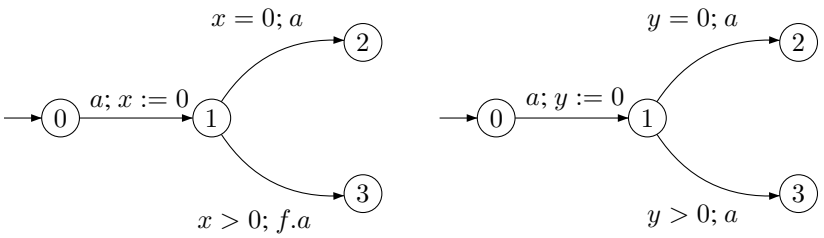
Figure 4.5. *Automata J and O^-*

Given a run ρ of a DTA H such that

$$\rho = (n_0, \mathbf{0}) \xrightarrow{\delta_0} (n_0, v'_0) \xrightarrow{a_1} (n_1, v_1) \cdots (n_{k_1}, v'_{k-1}) \xrightarrow{a_k} (n_k, v_k) \xrightarrow{\delta_k} (n_k, v'_k),$$

we let *Symbtr*(ρ) be the *symbolic trace* of ρ: *Symbtr*(ρ) is the[10] sequence (g_i, a_i, R_i) of symbolic labels of the transitions of H fired in the run ρ. Let *Det*(*RG*(H)) be the automaton obtained by determinization of the region graph *RG*(H) (i.e, we interpret the transitions τ and the transitions (g, τ, R) of *RG*(H) as invisible). By definition of the region graph *RG*(H), if ρ is a run of H then *Symbtr*(ρ) is a run of *Det*(*RG*(H)). Conversely, for each word w of *Det*(*RG*(H)), there is a timed word v which is accepted by H and such that *Symbtr*(v) $= w$. Actually, the automaton H cannot make any difference (considering the reachable locations) between two timed words v_1 and v_2 which have the same symbolic trace.

EXAMPLE 4.5 .– Consider the automaton given in Figure 4.6(a) taken from [BOU 05] and the canditate diagnoser Q of Figure 4.6(b). The transition $f.a$ in P indicates that a is fired immediatly after f. The automaton Q can be configured to diagnose P: it suffices to set localition 3 as accepting. The automaton Q has resource $(\{y\}, 0, 1)$ and Q is thus a 0-diagnoser for P.



(a) Automaton P (b) Automaton Q

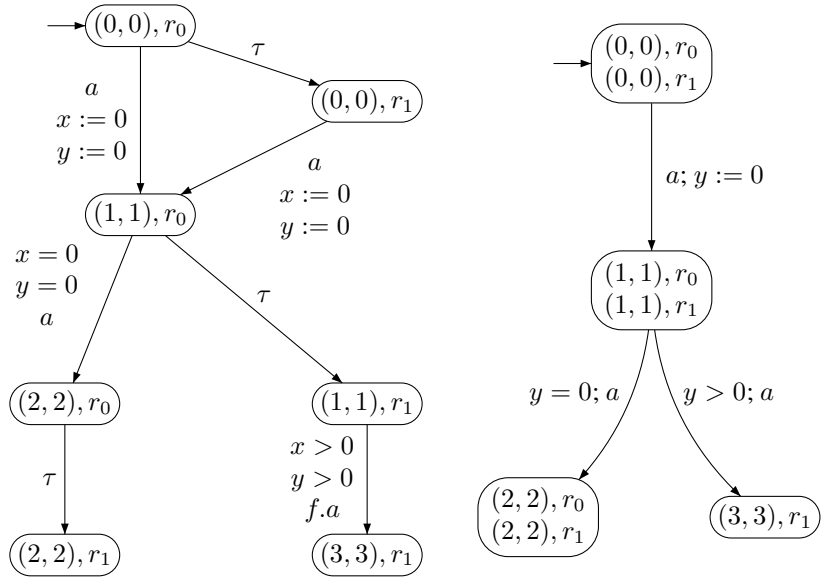
Figure 4.6. *Automata P and Q*

Formaly, an automaton Θ^- can be configured to be a diagnoser for A if and only if the following conditions, (K), is satisfied: we cannot find two runs $\rho_1 \in$ *Faulty*$_{\geq\Delta}(A)$, $\rho_2 \in$ *NonFaulty*(A), such that $\boldsymbol{\pi}_{/\Sigma}$(*tr*($\rho_1$)) and $\boldsymbol{\pi}_{/\Sigma}$(*tr*($\rho_2$)) produce the same symbolic trace when "read" by Θ^-. To check (K), we proceed as follows: let $RG_1 = RG(A(\Delta) \times \Theta^-)$ be the region graph of $A(\Delta) \times \Theta^-$. We "project" the labels of this region graph on Θ^-: each label different from τ (time elapsing to the next region) is of the form $(g_1 \wedge g_2, a, R_1 \cup R_2)$ where (g_2, a, R_2) is a symbolic label of Θ^-. The projection consists in replacing the labels $(g_1 \wedge g_2, a, R_1 \cup R_2)$ by (g_2, a, R_2). Let RG_2 be the graph obtained after these replacements. The next step consists in determinizing RG_2 to obtain *Det*(RG_2) (the transitions τ and (g, τ, R) are the invisible

10. We restrict ourselves to deterministic timed automata and thus there is only one sequence of transitions.

transitions). In *Det*(RG_2) the nodes are of the form $\{((q_1, \ell), r_1), \cdots, ((q_n, \ell), r_n)\}$ with ℓ a location of Θ^- because Θ^- is deterministic. We can thus use an equivalent form (S, ℓ) with $S = \{(q_1, r_1), \cdots, (q_n, r_n)\}$ and each q_i is a location of L and each r_i a region of RG_1. We let *Bad*(ℓ) be the set of nodes $(\{(q_1, r_1), \cdots, (q_n, r_n)\}, \ell)$ such that there exist two indices i, j with $q_i \in L_{\Delta f}$ and $q_j \in L_{\neg f}$.

EXAMPLE 4.6 .– For the automata P and Q of Figure 4.6, the graphs RG_1 and RG_2 are given in Figure 4.7. We assume $\Delta = 0$ and thus $P(\Delta) = P$ and the Δ-faulty location in P is 3. In the region graphs, r_0 is the initial region with $x = y = 0$ and r_1 is $x = y > 0$. There is no location ℓ in Q such that *Bad*(ℓ) $\neq \varnothing$ because only $(3, 3)$ is Δ-faulty.



(a) RG_1 (b) RG_2

Figure 4.7. *Region Graphs RG_1 and RG_2*

Checking whether a set F_Θ exists to make Θ^- a diagnoser amounts to checking whether there is a location $\ell \in N$ such that *Bad*(ℓ) $\neq \varnothing$. If the answer is "yes" then N cannot be partionned so that Θ^- is a diagnoser for A. If the answer is "no" it suffices to set F_Θ to the set of locations ℓ such that, in the region graph, (S, ℓ) contains a location q_i which is in $L_{\Delta f}$. The complexity of this algorithm is doubly exponential in the size of A (one exponential for the region graph, and another for the determinization step).

EXAMPLE 4.7 .– On the example of automaton Q, we set 3 to be accepting and Q is a diagnoser for P and this can be figured out using RG_2 of Figure 4.7.

To solve Problem 4.8, the algorithm given in [BOU 05] uses a construction similar to the one we have introduced.

THEOREM 4.11 ([BOU 05]).– *Problem 4.7 is 2EXPTIME-complete for DTA.*

PROOF 4.6 .– The 2EXPTIME easiness proof is done by reducing Problem 4.7 to a safety game problem. 2EXPTIME hardness consists in reducing the acceptance problem of an alternating Turing machine of exponential space to Problem 4.7. In the sequel we give the sketch of the 2EXPTIME easiness proof. For the hardness proof (very technical) and further details the reader is refered to [CHE 04] (in French). Let $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, Inv)$ be a TA and let $A_1(\Delta)$ be the automaton defined in section 4.5.1. Again we use the notation $A(\Delta)$ for $A_1(\Delta)$. As we do not know the structure of a DTA which could diagnose A, we start with the most powerful TA w.r.t. to the given fixed resource μ.

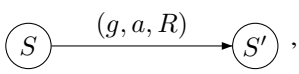
Given a resource $\mu = (Y, \max, \frac{1}{m})$ ($X \cap Y = \varnothing$), a *minimal guard* for μ is a guard which defines a region of μ. We can define the *universal automaton* $\mathcal{U} = (\{0\}, \{0\}, Y, \Sigma, E_\mu, Inv_\mu)$ by:

– $Inv_\mu(0) = \top$,

– $(0, g, a, R, 0) \in E_\mu$ for each (g, a, R) s.t. $a \in \Sigma$, $R \subseteq Y$, and g is a minimal guard for μ.

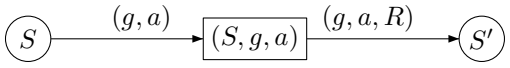
$\mathcal{U}$ is finite because E_μ is finite. Nevertheless $\mathcal{U}$ is not deterministic because it can choose to reset different sets of clocks Y for a pair "guard,letter" (g, a). To diagnose A, we have to find when a set of clocks has to be reset. This can provide enough information to distinguish Δ-faulty words from non-faulty words.

We first thus define the automaton $A(\Delta) \times \mathcal{U}$. Second we build the region graph $RG(A(\Delta) \times \mathcal{U})$ and compute its projection A' on $\mathcal{U}$ (using the same procedure as the one defined for Problem 4.8). Finally, we compute the determinization of A' and we obtain the automaton $H_{A,\Delta,\mu}$. The bad locations of $H_{A,\Delta,\mu}$ are the nodes which contain both location Bad and another location. We let $Bad(H_{A,\Delta,\mu})$ be the set of bad locations. Next we define a game from $H_{A,\Delta,\mu}$ using the following transformation. Each transition of $H_{A,\Delta,\mu}$ of the form



,

is split into two consecutive transitions:



Notice that R is a set of clocks of $\mathcal{U}$ because we have projected the labels on $\mathcal{U}$ in the first step.

This tranformation produces a turn-based game $G_{A,\Delta,\mu}$ where the set of states are partionnoned as follows: Player 1's states (round shape) of type S; Player 2's states (square shape), of the form (S, g, a). The *Bad* states of $G_{A,\Delta,\mu}$ are the states in *Bad*$(H_{A,\Delta,\mu})$. The objective of Player 2 is: "avoid states *Bad*". The key step in the proof is then to prove that: Player 2 has a *winning* startegy if and only if there is a DTA diagnoser for A in DTA_μ. As the game $G_{A,\Delta,\mu}$ is a finite turn-based game, if Player 2 can win, there is a *positional* winning strategy: the choice of actions for Player 2 only depends on the current state (square) of the game. There is even a so-called *most permissive positional strategy*: this strategy gives, for each state, the *set* of actions Player 2 can play to win the game. We thus have an algorithm to decide Problem 4.7: (1) we can check wether there is strategy for Player 2 to win the game $G_{A,\Delta,\mu}$ and (2) if the answer is "yes", compute the most permissive strategy. The previous algorithm runs in 2EXPTIME because (i) the game $G_{A,\Delta,\mu}$ has size doubly exponential in the size of A and (ii) solving a safety finite game can be done in linear time in the size of the game. □

EXAMPLE 4.8 .– The previous construction is examplified on Figure 4.8. The automaton to diagnose is P given in Figure 4.6. We look for DTA of resource $\mu = (\{y\}, 0, 1)$ to diagnose the faults within $\Delta = 0$ time units. As showed before, it suffices to use automaton Q and reset y when the first a occurs; if the second a occurs when $y > 0$ we announce a fault. The game $G_{P,0,\mu}$ obtained using the previous algorithm is given on Figure 4.8. In this graph, r_0 is the region $x = y = 0$, r_1 is $x = y > 0$, r_2 is $y = 0 \wedge x > 0$ and r_3 is $x = 0 \wedge y > 0$.

The states to avoid for Player 2 (who plays from square shaped states) are double circled: they contain a faulty state 3 and a non faulty state 2. In order to win, Player 2 must avoid state $(1, r_2)$ and thus must reset y when a occurs from states $\boxed{2}$. The winning choices for Player 2 are the plain arrows whereas the losing ones are dashd arrows. The accepting states where a fault can be announced are $(3, r_2)$ and $(3, r_1)$: in state $\boxed{4}$ we can choose either to reset y or not, as no fault has occurred.

Another interesting class of deterministic timed automata is the class of *Event Recording Automata* (ERA) introduced in [ALU 94b]. An ERA imposes that clocks be associated with events. Thus, for each event $a \in \Sigma$, there is a clock x_a which is reset exactly when a occurs. It thus measures the time since the last occurrence of a (or since the system started if no a occurred). The class ERA enjoys nice properties: in particular, every ERA A can be determinized in an ERA *Det*(A) which accepts

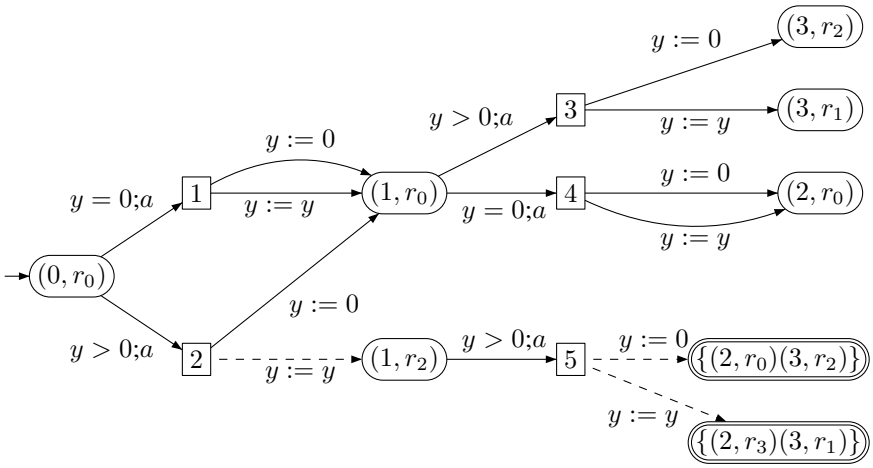


Figure 4.8. *The Game $G_{P,0,\mu}$*

the same language. Another result of [BOU 05] thus concerns ERA and proves that solving the diagnosis problem with ERA diagnosers is less expensive than for DTA :

THEOREM 4.12 ([BOU 05]).– *Problem 4.7 is PSPACE-complete for ERA.*

As the diagnosis problem is already PSPACE-complete, the previous result gives an optimal algorithm.

4.6. Other Results and Open Problems

The previous results extend for *multiple types of faults*. In this case, we want to diagnose several faults $f_1, f_2, \cdots, f_k$. It thus suffices to consider the diagnosis problem for each fault f_i (considering that the other faults $f_j, j \neq i$ are replaced by τ). The fault diagnosis problem is closely related to conformance testing. The relationships between these problems are investigated in [KRI 04a, KRI 04b].

[ALT 06, JIA 06] considers the diagnosis problem with *digital clocks*. In this framework, the diagnoser cannot measure time but can only count the number of ticks generated by a *clock*. The setting is thus a (non-deterministic) timed automaton A to be diagnosed, and a timed automaton *Clock* which generates *tick* events (we assume *tick* is not an event of A). The system to be observed produces timed words w wich are generated by the product $A \times$ *Clock*, but the diagnoser can only observe the untimed version, *Unt*(w), of w. Thus the timing information about the events of A can only be

inferred from the ordering of the events of A and the *tick* event. Given A and *Clock*, checking whether $A \times Clock$ is diagnosable is PSPACE [ALT 06].

Finally, here are some open problems about fault diagnosis of timed systems:

– diagnosability with DTA with no fixed resource or with fixed resource but no bound Δ on the maximal delay; the corresponding problems are Problem 4.5 and 4.6;

– for the digital clock diagnosis problem, the problem of deciding the existence of a digital clock (timed automaton) *Clock* such that $A \times Clock$ is diagnosable.

4.7. Bibliography

[ALT 06] ALTISEN K., CASSEZ F., TRIPAKIS S., "Monitoring and Fault-Diagnosis with Digital Clocks", *Proceedings of the 6th International Conference on Application of Concurrency to System Design (ACSD'06)*, p. 101-110, IEEE Computer Society, June 2006.

[ALU 94a] ALUR R., DILL D., "A theory of timed automata", *Theoretical Computer Science*, vol. 126, p. 183-235, 1994.

[ALU 94b] ALUR R., FIX L., HENZINGER T.A., "A Determinizable Class of Timed Automata", *Proceedings of the 6th International Conference on Computer Aided Verification (CAV'94)*, vol. 818 of *Lecture Notes in Computer Science*, p. 1-13, Springer, 1994.

[BER 98] BÉRARD B., DIEKERT V., GASTIN P., PETIT A., "Characterization of the Expressive Power of Silent Transitions in Timed Automata", *Fundamenta Informaticae*, vol. 36, num. 2-3, p. 145-182, IOS Press, 1998.

[BOU 05] BOUYER P., CHEVALIER F., D'SOUZA D., "Fault Diagnosis Using Timed Automata", *Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, vol. 3441 of *Lecture Notes in Computer Science*, p. 219-233, Springer, April 2005.

[CAS 08] CASSEZ F., TRIPAKIS S., "Fault Diagnosis with Static or Dynamic Diagnosers", *Fundamenta Informaticae*, vol. 88, num. 4, p. 497–540, November 2008.

[CHE 04] CHEVALIER F., Détection d'erreurs dans les systèmes temporisés, Rapport de DEA, DEA Algorithmique, Paris, France, September 2004.

[COU 05] COUVREUR J.M., DURET-LUTZ A., POITRENAUD D., "On-the-Fly Emptiness Checks for Generalized Büchi Automata", GODEFROID P. (dir.), *Proceedings of SPIN*, vol. 3639 of *Lecture Notes in Computer Science*, p. 169-184, Springer, 2005.

[FIN 05] FINKEL O., "On decision problems for timed automata", *Bulletin of the European Association for Theoretical Computer Science*, vol. 87, p. 185-190, 2005.

[HOL 05] HOLZMANN G.J., "Software model checking with SPIN", *Advances in Computers*, vol. 65, p. 78-109, 2005.

[JIA 01] JIANG S., HUANG Z., CHANDRA V., KUMAR R., "A Polynomial Algorithm for Testing Diagnosability of Discrete Event Systems", *IEEE Transactions on Automatic Control*, vol. 46, num. 8, August 2001.

[JIA 06] JIANG S., KUMAR R., "Diagnosis of Dense-Time Systems Using Digital Clocks", *Proceedings of the American Control Conference (ACC'06)*, IEEE Computer Society, June 2006.

[KRI 04a] KRICHEN M., TRIPAKIS S., "Black-box conformance testing for real-time systems", *Proc. of the 11th International SPIN Workshop on Model Checking of Software (SPIN'04)*, vol. 2989 of *Lecture Notes in Computer Science*, p. 109-126, Springer, 2004.

[KRI 04b] KRICHEN M., TRIPAKIS S., "Real-time Testing with Timed Automata Testers and Coverage Criteria", *Proceedings of Formal Techniques, Modelling and Analysis of Timed and Fault Tolerant Systems (FORMATS-FTRTFT'04)*, vol. 3253 of *Lecture Notes in Computer Science*, p. 134-151, Springer, 2004.

[RIC 06] RICKER L., LAFORTUNE S., GENC S., "DESUMA: A Tool Integrating GIDDES and UMDES", *Proc. of the* 8^{th} *Workshop on Discrete Event Systems (WODES'08)*, Ann Arbor, MI, USA, IEEE Computer Society, July 2006.

[SAM 95] SAMPATH M., SENGUPTA R., LAFORTUNE S., SINNAMOHIDEEN K., TENEKETZIS D., "Diagnosability of Discrete Event Systems", *IEEE Transactions on Automatic Control*, vol. 40, num. 9, September 1995.

[SAM 96] SAMPATH M., SENGUPTA R., LAFORTUNE S., SINNAMOHIDEEN K., TENEKETZIS D., "Failure Diagnosis Using Discrete-Event Models", *IEEE Transactions on Control Systems technology*, vol. 4, num. 2, March 1996.

[SCH 99] SCHNOEBELEN P., BÉRARD B., BIDOIT M., LAROUSSINIE F., PETIT A., *Vérification de logiciels : Techniques et outils de model-checking*, Vuibert, Paris, 1999.

[SCH 05] SCHWOON S., ESPARZA J., "A Note on On-the-Fly Verification Algorithms", HALBWACHS N., ZUCK L.D. (dir.), *TACAS*, vol. 3440 of *Lecture Notes in Computer Science*, p. 174-190, Springer, 2005.

[TRI 02] TRIPAKIS S., "Fault Diagnosis for Timed Automata", *Proceedings of the International Conference on Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT'02)*, vol. 2469 of *Lecture Notes in Computer Science*, p. 205-224, Springer, 2002.

[TRI 06] TRIPAKIS S., "Folk theorems on the determinization and minimization of timed automata", *Information Processing Letters*, vol. 99, num. 6, p. 222-226, Elsevier, 2006.

[YOO 02] YOO T.S., LAFORTUNE S., "Polynomial-Time Verification of Diagnosability of Partially-Observed Discrete-Event Systems", *IEEE Transactions on Automatic Control*, vol. 47, num. 9, p. 1491-1495, September 2002.

[YOO 03] YOO T.S., GARCIA H., "Computation of Fault Detection Delay in Discrete-Event Systems", *Proceedings of the 14th International Workshop on Principles of Diagnosis, DX'03*, p. 207-212, Washington, D.C., Etats-Unis, June 2003.