The Complexity of Codiagnosability for Discrete Event and Timed Systems

Franck Cassez*

National ICT Australia & CNRS
The University of New South Wales
Sydney, Australia

Abstract. In this paper we study the fault codiagnosis problem for discrete event systems given by finite automata (FA) and timed systems given by timed automata (TA). We provide a uniform characterization of codiagnosability for FA and TA which extends the necessary and sufficient condition that characterizes diagnosability. We also settle the complexity of the codiagnosability problems both for FA and TA and show that codiagnosability is PSPACE-complete in both cases. For FA this improves on the previously known bound (EXPTIME) and for TA it is a new result. Finally we address the codiagnosis problem for TA under bounded resources and show it is 2EXPTIME-complete.

1 Introduction

Discrete-event systems [1,2] (DES) can be modelled by finite automata (FA) over an alphabet of *observable* events Σ.

The *fault diagnosis problem* is a typical example of a problem under partial observation. The aim of fault diagnosis is to detect *faulty* sequences of the DES. The assumptions are that the behavior of the DES is known and a model of it is available as a finite automaton over an alphabet $\Sigma \cup \{\tau, f\}$, where Σ is the set of observable events, τ represents the unobservable events, and f is a special unobservable event that corresponds to the faults: this is the original framework introduced by M. Sampath *et al.* [3] and the reader is referred to this paper for a clear and exhaustive introduction to the subject. A *faulty* sequence is a sequence of the DES containing an occurrence of event f. An *observer* which has to detect faults, knows the specification/model of the DES, and it is able to observe sequences of *observable* events. Based on this knowledge, it has to announce whether an observation it makes (in Σ^*) was produced by a faulty sequence (in $(\Sigma \cup \{\tau, f\})^*$) of the DES or not. A *diagnoser* (for a DES) is an observer which observes the sequences of observable events and is able to detect whether a fault event has occurred, although it is not observable. If a diagnoser can detect a fault at most Δ steps[1] after it has occurred, the DES is said to be Δ-diagnosable. It is diagnosable if it is Δ-diagnosable for some $\Delta \in \mathbb{N}$. Checking whether a DES is Δ-diagnosable for

* Author supported by a Marie Curie International Outgoing Fellowship within the 7th European Community Framework Programme.

[1] Steps are measured by the number of transitions in the DES.

a given Δ is called the *bounded diagnosability problem*; checking whether a DES is diagnosable is the *diagnosability problem*.

Checking *diagnosability* for a given DES and a fixed set of observable events can be done in polynomial time using the algorithms of [4,5]. If a diagnoser exists there is a finite state one. Nevertheless the size of the diagnoser can be exponential as it involves a determinization step. The extension of this DES framework to timed automata (TA) has been proposed by S. Tripakis in [6], and he proved that the problem of checking diagnosability of a timed automaton is PSPACE-complete. In the timed case however, the diagnoser may be a Turing machine. The problem of checking whether a timed automaton is diagnosable by a diagnoser which is a *deterministic* timed automaton was studied by P. Bouyer *et al.* [7].

Codiagnosability generalizes diagnosability by considering *decentralized architectures*. Such decentralized architectures have been introduced in [8] and later refined in [9,10]. In these architectures, local diagnosers (with their own partial view of the system) can send some information to a coordinator, summarizing their observations. The coordinator then computes a result from the partial results of the local diagnosers. The goal is to obtain a coordinator that can detect the faults in the system. When local diagnosers do not communicate with each other nor with a coordinator (protocol 3 in [8]), the decentralized diagnosis problem is called *codiagnosis* [10,9]. In this case, codiagnosis means that each fault can be detected by at least one local diagnoser. In the paper [10], codiagnosability is considered and an algorithm to check codiagnosability is presented for discrete event systems (FA). An upper bound for the complexity of the algorithm is EXPTIME. In [9], the authors consider a *hierarchical framework* for decentralized diagnosis. In [11] a notion of *robust* codiagnosability is introduced, which can be thought of as a *fault tolerant* (local diagnosers can fail) version of codiagnosability. None of the previous papers has addressed the codiagnosability problems for timed automata. Moreover, the exact complexity of the codiagnosis problems is left unsettled for discrete event systems (FA).

Our Contribution. In this paper, we study the codiagnosability problems for FA and TA. We settle the complexity of the problems for FA (PSPACE-complete), improving on the best known upper bound (EXPTIME). We also address the codiagnosability problems for TA and provide new results: algorithms to check codiagnosability and also codiagnosability under bounded resources. Our contribution is both of theoretical and practical interests. The algorithms we provide are optimal, and can also be implemented using standard model-checking tools like SPIN [12] for FA, or UPPAAL [13] for TA. This means that very expressive languages can be used to specify the systems to codiagnose and very efficient implementations and data structures are readily available.

Organisation of the Paper. Section 2 recalls the definitions of finite automata and timed automata. We also give some results on the Intersection Emptiness Problems (section 2.6) that will be used in the next sections. Section 3 introduces the fault codiagnosis problems we are interested in, and a necessary and sufficient condition that characterizes codiagnosability for FA and TA. Section 4 contains the first main results: optimal algorithms for the codiagnosability problems for FA and TA. Section 5 describes how

to synthesize the codiagnosers and the limitations of this technique for TA. Section 6 is devoted to the codiagnosability problem under bounded resources for TA and contains the second main result of the paper.

Omitted proofs can be found in the full version of this paper [14].

2 Preliminaries

Σ denotes a finite alphabet and $\Sigma_\tau = \Sigma \cup \{\tau\}$ where $\tau \notin \Sigma$ is the *unobservable* action. $\mathbb{B} = \{\text{TRUE}, \text{FALSE}\}$ 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 $\mathbb{R}_{\geq 0}$ (resp. $\mathbb{R}_{>0}$) is the set of non-negative (resp. positive) real numbers. We denote tuples (or vectors) by $\overline{d} = (d_1, \cdots, d_k)$ and write $\overline{d}[i]$ for d_i.

2.1 Clock Constraints

Let X be a finite set of variables called *clocks*. A *clock valuation* is a mapping $v : X \to \mathbb{R}_{\geq 0}$. We let $\mathbb{R}_{\geq 0}^X$ be the set of clock valuations over X. We let $\overline{0}_X$ be the *zero* valuation where all the clocks in X are set to 0 (we use $\overline{0}$ when X is clear from the context). Given $\delta \in \mathbb{R}$, $v + \delta$ is 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. We also write $[\![g]\!]$ for the set $\{v \mid v \models g\}$. 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.

2.2 Timed Words

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. $(\mathbb{R}_{\geq 0}.\Sigma)^\omega$). *Duration*$(w)$ is 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*$(0.4\ a\ 1.0\ b\ 2.7\ c) = abc$. In this paper 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.

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

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

2.3 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 [15].

Definition 1 (Timed Automaton). *A* Timed Automaton A *is a tuple* $(L, l_0, X, \Sigma_\tau, E, Inv, F, R)$ *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; as usual we often write a transition* $\ell \xrightarrow{g,a,r} \ell'$*;* $Inv \in \mathcal{C}(X)^L$ *associates with each location an* invariant*; as usual we require the invariants to be conjunctions of constraints of the form* $x \preceq c$ *with* $\preceq \in \{<, \leq\}$*;* $F \subseteq L$ *is the set of* final *locations and* $R \subseteq L$ *is the set of* repeated *locations.* ■

The size of a TA A is denoted $|A|$ and is the size of the clock constraints i.e., the size of the transition relation E. A *state* of A is a pair $(\ell, v) \in L \times \mathbb{R}^X_{\geq 0}$. A *run* ϱ of A from (ℓ_0, v_0) is a (finite or infinite) sequence of alternating *delay* and *discrete* moves:

$$\varrho = (\ell_0, v_0) \xrightarrow{\delta_0} (\ell_0, v_0 + \delta_0) \xrightarrow{a_0} (\ell_1, v_1) \cdots \xrightarrow{a_{n-1}} (\ell_n, v_n) \xrightarrow{\delta_n} (\ell_n, v_n + \delta_n) \cdots$$

s.t. for every $i \geq 0$:

- $v_i + \delta \models Inv(\ell_i)$ for $0 \leq \delta \leq \delta_i$;
- there is some transition $(\ell_i, g_i, a_i, r_i, \ell_{i+1}) \in E$ s.t. : (i) $v_i + \delta_i \models g_i$, (ii) $v_{i+1} = (v_i + \delta_i)[r_i]$.

The set of finite (resp. infinite) runs in A from a state s is denoted $Runs^*(s, A)$ (resp. $Runs^\omega(s, A)$). We let $Runs^*(A) = Runs^*(s_0, A)$, $Runs^\omega(A) = Runs^\omega(s_0, A)$ with $s_0 = (l_0, \overline{0})$, and $Runs(A) = Runs^*(A) \cup Runs^\omega(A)$. If ϱ is finite and ends in s_n, we let $last(\varrho) = s_n$. 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*, $tr(\varrho)$, of a run ϱ is the timed word $\boldsymbol{\pi}_\Sigma(\delta_0 a_0 \delta_1 a_1 \cdots a_n \delta_n \cdots)$. The duration of the run ϱ is $Duration(\varrho) = Duration(tr(\varrho))$. For $V \subseteq Runs(A)$, we let $Tr(V) = \{tr(\varrho) \mid \varrho \in V\}$, which is the set of traces of the runs in V.

A finite (resp. infinite) timed word w is *accepted* by A if it is the trace of a run of A that ends in an F-location (resp. a 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.

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

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, $g \wedge g' \equiv$ FALSE. 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 $v \models g$. We note DTA the class of deterministic timed automata.

A finite automaton is a particular TA with $X = \varnothing$. Consequently guards and invariants are vacuously true and time elapsing transitions do not exist. We write $A = (Q,$

$q_0, \Sigma_\tau, E, F, R)$ for a finite automaton. A run is thus a sequence of the form: $\varrho = \ell_0 \xrightarrow{a_0} \ell_1 \cdots\cdots \xrightarrow{a_{n-1}} \ell_n \cdots$ where for each $i \geq 0$, $(\ell_i, a_i, \ell_{i+1}) \in E$. Definitions of traces and languages are the same as for TA. For FA, 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*$(\varrho) = \infty$.

2.4 Region Graph of a Timed Automaton

A *region* of $\mathbb{R}^X_{\geq 0}$ is a conjunction of *atomic* constraints of the form $x \bowtie c$ or $x - y \bowtie c$ with $c \in \mathbb{Z}$, $\bowtie \in \{\leq, <, =, >, \geq\}$ and $x, y \in X$. 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 [15]. 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 $(l_0, \overline{0})$. 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 [15] is:

Theorem 1 (R. Alur and D. Dill, [15]). $\mathcal{L}(RG(A)) = Unt(\mathcal{L}(A))$.

The (maximum) size of the region graph is exponential in the number of clocks and in the maximum constant of the automaton A (see [15]): $|RG(A)| = |L| \cdot |X|! \cdot 2^{|X|} \cdot K^{|X|}$ where K is the largest constant used in A.

2.5 Product of Timed Automata

Given a n locations $\ell_1, \cdots, \ell_n$, we write $\overline{\ell}$ for the tuple $(\ell_1, \cdots, \ell_n)$ and let $\overline{\ell}[i] = \ell_i$. Given a letter $a \in \Sigma^1 \cup \cdots \cup \Sigma^n$, we let $I(a) = \{k \mid a \in \Sigma^k\}$.

Definition 2 (Product of TA). *Let $A_i = (L_i, l_0^i, X_i, \Sigma_\tau^i, E_i, Inv_i)$, $i \in \{1, \cdots, n\}$, be n TA s.t. $X_i \cap X_j = \varnothing$ for $i \neq j$. The* product *of the A_i is the TA $A = A_1 \times \cdots \times A_n = (L, \overline{l_0}, X, \Sigma_\tau, E, Inv)$ given by: $L = L_1 \times \cdots \times L_n$; $\overline{l_0} = (l_0^1, \cdots, l_0^n)$; $\Sigma = \Sigma^1 \cup \cdots \cup \Sigma^n$; $X = X_1 \cup \cdots \cup X_n$; $E \subseteq L \times \mathcal{C}(X) \times \Sigma_\tau \times 2^X \times L$ and $(\overline{\ell}, g, a, r, \overline{\ell}') \in E$ if:*

- *either $a \in \Sigma \setminus \{\tau\}$, and*
  1. *for each $k \in I(a)$, $(\overline{\ell}[k], g_k, a, r_k, \overline{\ell}'[k]) \in E_k$,*
  2. *$g = \wedge_{k \in I(a)} g_k$ and $r = \cup_{k \in I(a)} r_k$;*
  3. *for $k \notin I(a)$, $\overline{\ell}'[k] = \overline{\ell}[k]$;*
- *or $a = \tau$ and $\exists j$ s.t. $(\overline{\ell}[j], g_j, \tau, r_j, \overline{\ell}'[j]) \in E_j$, $g = g_j$, $r = r_j$ and for $k \neq j$, $\overline{\ell}'[k] = \overline{\ell}[k]$.*

$Inv(\overline{\ell}) = \wedge_{k=1}^n Inv(\overline{\ell}[k])$. ■

This definition of product also applies to finite automata (no clock constraints).

If the automaton A_i has the set of final locations F_i then the set of final locations for A is $F_1 \times \cdots \times F_n$. For Büchi acceptance, we add a counter c to A which is incremented every time the product automaton A encounters an R_i-location in A_i, following the standard construction for product of Büchi automata. The automaton constructed with the counter c is A^+. The repeated set of states of A^+ is $L_1 \times \cdots \times L_{n-1} \times L_n \times \{n\}$. As the sets of clocks of the A_i's are disjoint[2], the following holds:

Fact 1 *$\mathcal{L}^*(A) = \cap_{i=1}^n \mathcal{L}^*(A_i)$ and $\mathcal{L}^\omega(A^+) = \cap_{i=1}^n \mathcal{L}^\omega(A_i)$.*

2.6 Intersection Emptiness Problem

In this section we give some complexity results for the emptiness problem on products of FA and TA.
First consider the following problem on deterministic finite automata (DFA):

Problem 1 (Intersection Emptiness for DFA)
INPUTS: *n deterministic finite automata $A_i, 1 \leq i \leq n$, over the alphabet Σ.*
PROBLEM: *Check whether $\cap_{i=1}^n \mathcal{L}^*(A_i) \neq \varnothing$.*

The size of the input for Problem 1 is $\sum_{i=1}^n |A_i|$.

Theorem 2 (D. Kozen, [16]). *Problem 1 is PSPACE-complete.*

D. Kozen's Theorem also holds for Büchi languages:

Theorem 3. *Checking whether $\cap_{i=1}^n \mathcal{L}^\omega(A_i) \neq \varnothing$ is PSPACE-complete.*

Problem 1 is PSPACE-hard even if $A_2, \cdots, A_n$ are automata where all the states are accepting and A_1 is the only automaton with a proper set of accepting states (actually one accepting state is enough).

Proposition 1. *Let $A_i, 1 \leq i \leq n$ be n DFA over the alphabet Σ. If for all $A_i, 2 \leq i \leq n$, all states of A_i are accepting, Problem 1 is already PSPACE-hard.*

The next results are counterparts of D. Kozen's results for TA.

Problem 2 (Intersection Emptiness for TA)
INPUTS: *n TA $A_i = (L_i, l_0^i, X_i, \Sigma_\tau^i, E_i, Inv_i, F_i)$, $1 \leq i \leq n$ with $X_k \cap X_j = \varnothing$ for $k \neq j$.*
PROBLEM: *Check whether $\cap_{i=1}^n \mathcal{L}^*(A_i) \neq \varnothing$.*

Theorem 4. *Problem 2 is PSPACE-complete.*

The previous theorem extends to Büchi languages:

Problem 3 (Büchi Intersection Emptiness for TA)
INPUTS: *n TA $A_i = (L_i, l_0^i, X_i, \Sigma_\tau^i, E_i, Inv_i, R_i)$, $1 \leq i \leq n$ with $X_k \cap X_j = \varnothing$ for $k \neq j$.*
PROBLEM: *Check whether $\cap_{i=1}^n \mathcal{L}^\omega(A_i) \neq \varnothing$.*

Theorem 5. *Problem 3 is PSPACE-complete.*

[2] For finite automata, this is is vacuously true.

3 Fault Codiagnosis Problems

We first recall the basics of *fault diagnosis*. The purpose of fault diagnosis [3] is to detect a fault in a system as soon as possible. The assumption is that the model of the system is known, but only a subset Σ_o of the set of events Σ generated by the system are observable. Faults are also unobservable.

Whenever the system generates a timed word $w \in TW^*(\Sigma)$, an external observer can only see $\boldsymbol{\pi}_{\Sigma_o}(w)$. If an observer can detect faults under this partial observation of the outputs of A, it is called a *diagnoser*. We require a diagnoser to detect a fault within a given delay $\Delta \in \mathbb{N}$.

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* (and unobservable) event. We only consider one type of fault, 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 [5]. The observable events are given by $\Sigma_o \subseteq \Sigma$ and τ is always unobservable.

The idea of *decentralized* or *distributed* diagnosis was introduced in [8]. It is based on decentralized architectures: local diagnosers and a communication protocol. In these architectures, local diagnosers (with their own partial view of the system) can send to a coordinator some information, using a given communication protocol. The coordinator then computes a result from the partial results of the local diagnosers. The goal is to obtain a coordinator that can detect the faults in the system. When local diagnosers do not communicate with each other nor with a coordinator (protocol 3 in [8]), the decentralized diagnosis problem is called *codiagnosis* [10,9]. In this section we formalize the notion of codiagnosability introduced in [10] in a style similar to [17]. This allows us to obtain a necessary and sufficient condition for codiagnosability of FA but also to extend the definition of codiagnosability to *timed automata*.

In the sequel we assume that the model of the system is a TA $A = (L, l_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$ and is fixed.

3.1 Faulty Runs

Let $\Delta \in \mathbb{N}$. A run $\varrho = (\ell_0, v_0) \xrightarrow{\delta_0} (\ell_0, v_0 + \delta_0) \xrightarrow{a_0} (\ell_1, v_1) \cdots \xrightarrow{a_{n-1}} (\ell_n, v_n) \xrightarrow{\delta_n} (\ell_n, v_n + \delta) \cdots$ of A is Δ-faulty if: (1) there is an index i s.t. $a_i = f$ and (2) the duration of $\varrho' = (\ell_i, v_i) \xrightarrow{\delta_i} \cdots \xrightarrow{\delta_n} (\ell_n, v_n + \delta_n) \cdots$ is larger or equal to 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 $\mathit{Faulty}^{tr}_{\geq\Delta}(A) = \mathit{Tr}(\mathit{Faulty}_{\geq\Delta}(A))$ and $\mathit{NonFaulty}^{tr}(A) = \mathit{Tr}(\mathit{NonFaulty}(A))$ which are the traces[3] of Δ-faulty and non-faulty runs of A.

We also make the assumption that the TA A cannot prevent time from elapsing. For FA, this assumption is that from any state, a discrete transition can be taken. If it is not

[3] Notice that $tr(\varrho)$ erases τ and f.

case, τ loop actions can be added with no impact on the (co)diagnosability status of the system. This is a standard assumption in diagnosability and is required to avoid taking into account these cases that are not interesting in practice.

For discrete event systems (FA), the notion of time is the number of transitions (discrete steps) in the system. A Δ-faulty run is thus a run with a fault action f followed by at least Δ discrete steps (some of them can be τ or even f actions). When we consider codiagnosability problems for discrete event systems, this definition of Δ-faulty runs apply. The other definitions are unchanged.

Remark 1. 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 FA (discrete event systems).

3.2 Codiagnosers and Codiagnosability Problems

A *codiagnoser* is a tuple of diagnosers, each of which has its own set of observable events Σ_i, and whenever a fault occurs, at least one diagnoser is able to detect it. In the sequel we write $\boldsymbol{\pi}_i$ in place of $\boldsymbol{\pi}_{\Sigma_i}$ for readability reasons. A codiagnoser can be formally defined as follows:

Definition 3 ($(\Delta, \mathcal{E})$-Codiagnoser). *Let A be a timed automaton over the alphabet $\Sigma_{\tau,f}$, $\Delta \in \mathbb{N}$ and $\mathcal{E} = (\Sigma_i)_{1\leq i\leq n}$ be a family of subsets of Σ. A $(\Delta, \mathcal{E})$-*codiagnoser *for A is a mapping $\overline{D} = (D_1, \cdots, D_n)$ with $D_i : TW^*(\Sigma_i) \to \{0, 1\}$ such that:*

- *for each $\varrho \in NonFaulty(A)$, $\sum_{i=1}^{n} \overline{D}[i](\boldsymbol{\pi}_i(tr(\varrho))) = 0$,*
- *for each $\varrho \in Faulty_{\geq\Delta}(A)$, $\sum_{i=1}^{n} \overline{D}[i](\boldsymbol{\pi}_i(tr(\varrho))) \geq 1$.* ■

As for diagnosability, the intuition of this definition is that (i) the codiagnoser will raise an alarm ($\overline{D}$ outputs a value different from 0) when a Δ-faulty run has been identified, and that (ii) it can identify those Δ-faulty runs unambiguously. The codiagnoser is not required to do anything special for Δ'-faulty runs with $\Delta' < \Delta$ (although it is usually required that once it has announced a fault, it does not change its mind and keep outputting 1).

A is $(\Delta, \mathcal{E})$-codiagnosable if there exists a $(\Delta, \mathcal{E})$-codiagnoser for A. A is $\mathcal{E}$-codiagnosable if there is some $\Delta \in \mathbb{N}$ s.t. A is $(\Delta, \mathcal{E})$-codiagnosable.

The standard notions [3] of Δ-diagnosability and Δ-diagnoser are obtained when the family $\mathcal{E}$ is the singleton $\mathcal{E} = \{\Sigma\}$. The fundamental codiagnosability problems for timed automata are the following:

Problem 4 ($(\Delta, \mathcal{E})$-Codiagnosability)
INPUTS: *A TA $A = (L, l_0, X, \Sigma_{\tau,f}, E, Inv)$, $\Delta \in \mathbb{N}$ and $\mathcal{E} = (\Sigma_i)_{1\leq i\leq n}$.*
PROBLEM: *Is A $(\Delta, \mathcal{E})$-codiagnosable?*

Problem 5 (Codiagnosability)
INPUTS: *A TA* $A = (L, l_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$ *and* $\mathcal{E} = (\Sigma_i)_{1 \leq i \leq n}$.
PROBLEM: *Is* A $\mathcal{E}$*-codiagnosable?*

Problem 6 (Optimal delay)
INPUTS: *A TA* $A = (L, l_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$ *and* $\mathcal{E} = (\Sigma_i)_{1 \leq i \leq n}$.
PROBLEM: *What is the minimum* Δ *s.t.* A *is* $(\Delta, \mathcal{E})$*-codiagnosable?*

The size of the input for Problem 4 is $|A| + \log \Delta + n \cdot |\Sigma|$, and for Problems 5 and 6 it is $|A| + n \cdot |\Sigma|$.

In addition to the previous problems, we will consider the construction of a $(\Delta, \mathcal{E})$-codiagnoser when A is $(\Delta, \mathcal{E})$-codiagnosable in section 5.

3.3 Necessary and Sufficient Condition for Codiagnosability

In this section we generalize the necessary and sufficient condition for diagnosability [6,17] to codiagnosability.

Lemma 1. *A is not $(\Delta, \mathcal{E})$-codiagnosable if and only if $\exists \varrho \in \mathit{Faulty}_{\geq \Delta}(A)$ and*

$$\forall 1 \leq i \leq n, \exists \varrho_i \in \mathit{NonFaulty}(A)\ s.t.\ \boldsymbol{\pi}_i(\mathit{tr}(\varrho)) = \boldsymbol{\pi}_i(\mathit{tr}(\varrho_i)). \tag{1}$$

Using Lemma 1, we obtain a language based characterisation of codiagnosability extending the one given in [6,17]. Let $\boldsymbol{\pi}_i^{-1}(X) = \{w \in \mathit{TW}^*(\Sigma) \mid \boldsymbol{\pi}_i(w) \in X\}$.

Lemma 2. *A is $(\Delta, \mathcal{E})$-codiagnosable if and only if*

$$\mathit{Faulty}^{tr}_{\geq \Delta}(A) \cap \left(\bigcap_{i=1}^{n} \boldsymbol{\pi}_i^{-1}\big(\boldsymbol{\pi}_i(\mathit{NonFaulty}^{tr}(A))\big) \right) = \varnothing. \tag{2}$$

4 Algorithms for Codiagnosability Problems

4.1 $(\Delta, \mathcal{E})$-Codiagnosability (Problem 4)

Deciding Problem 4 amounts to checking whether equation 2 holds or not. Recall that $A = (L, l_0, X, \Sigma_{\tau,f}, E, \mathit{Inv})$. Let t be a fresh clock not in X. Let $A^f(\Delta) = ((L \times \{0,1\}) \cup \{Bad\}, (l_0, 0), X \cup \{t\}, \Sigma_\tau, E_f, \mathit{Inv}_f)$ with:

- $((\ell, n), g, \lambda, r, (\ell', n)) \in E_f$ if $(\ell, g, \lambda, r, \ell') \in E$, $\lambda \in \Sigma \cup \{\tau\}$;
- $((\ell, 0), g, \tau, r \cup \{t\}, (\ell', 1)) \in E_f$ if $(\ell, g, f, r, \ell') \in E$;
- for $\ell \in L$, $((\ell, 1), t \geq \Delta, \tau, \varnothing, Bad) \in E_f$;
- $\mathit{Inv}_f((\ell, n)) = \mathit{Inv}(\ell)$.

$A^f(\Delta)$ is similar to A but when a fault occurs it switches to a copy of A (encoded by $n = 1$). When sufficient time has elapsed in the copy (more than Δ time units), location *Bad* can be reached. The language accepted by $A^f(\Delta)$ with the set of final states $\{\mathit{Bad}\}$ is thus $\mathcal{L}^*(A^f(\Delta)) = \mathit{Faulty}^{tr}_{\geq \Delta}(A)$. Define $A_i = (L, l_0, X_i, \Sigma_\tau, E_i, \mathit{Inv}_i)$ with:

– $X_i = \{x^i \mid x \in X\}$ (create copies of clocks of A);
– $(\ell, g_i, \lambda, r_i, \ell') \in E_i$ if $(\ell, g, \lambda, r, \ell') \in E$, $\lambda \in \Sigma_i \cup \{\tau\}$ with: g_i is g where the clocks x in X are replaced by their counterparts x^i in X_i; r_i is r with the same renaming;
– $(\ell, g_i, \tau, r_i, \ell') \in E_i$ if $(\ell, g, \lambda, r, \ell') \in E$, $\lambda \in \Sigma \setminus \Sigma_i$
– $\mathit{Inv}_i(\ell) = \mathit{Inv}(\ell)$ with clock renaming (x^i in place of x).

Each A_i accepts only non-faulty traces as the f-transitions are not in A_i. If the set of final locations is L for each A_i, then $\mathcal{L}^*(A_i) = \boldsymbol{\pi}_i(\mathit{NonFaulty}^{tr}(A))$. To accept $\boldsymbol{\pi}_i^{-1}\big(\boldsymbol{\pi}_i(\mathit{NonFaulty}^{tr}(A))$ we add transitions $(\ell, \text{TRUE}, \lambda, \varnothing, \ell)$ for each location ℓ of E_i and for each $\lambda \in \Sigma \setminus \Sigma_i$. Let A_i^* be the automaton on the alphabet Σ constructed this way. By definition of A_i^*, $\mathcal{L}^*(A_i^*) = \boldsymbol{\pi}_i^{-1}\big(\boldsymbol{\pi}_i(\mathit{NonFaulty}^{tr}(A))\big)$.

Define $\mathcal{B} = A^f(\Delta) \times A_1^* \times A_2^* \times \cdots \times A_n^*$ with the set of final locations $F_{\mathcal{B}} = \{\mathit{Bad}\} \times L \times \cdots \times L$. We let $R_{\mathcal{B}} = \varnothing$. Using equation 2 we obtain:

Lemma 3. *A is $(\Delta, \mathcal{E})$-codiagnosable iff $\mathcal{L}^*(\mathcal{B}) = \varnothing$.*

The size of the input for problem 4 is $|A| + \log \Delta + n \cdot |\Sigma|$. The size of $A^f(\Delta)$ is (linear in) the size of A and $\log \Delta$, i.e., $O(|A| + \log \Delta)$. The size of A_i^* is also bounded by the size of A. It follows that $|A^f(\Delta)| + \sum_{i=1}^n |A_i^*|$ is bounded by $(n+1)|A|$ and is polynomial in the size of the input of problem 4. We thus have a polynomial reduction from Problem 4 to the intersection emptiness problem for TA. We can now establish the following result:

Theorem 6. *Problem 4 is PSPACE-complete for Timed Automata. It is already PSPACE-hard for Deterministic Finite Automata.*

4.2 $\mathcal{E}$-Codiagnosability (Problem 5)

In this section we show how to solve the $\mathcal{E}$-codiagnosability problem. The algorithm is a generalisation of the procedure for deciding diagnosability of discrete event and timed systems (see [18] for a recent presentation).

For standard fault diagnosis (one diagnoser and $\mathcal{E} = \{\Sigma\}$), A is not diagnosable if there is an infinite faulty run in A the projection of which is the same as the projection of a non-faulty one [18].

The procedure for checking diagnosability of FA and TA slightly differ due to specific features of timed systems. We recall here the algorithms to check diagnosability of FA and TA [18,6] and extend them to codiagnosability.

Codiagnosability for Finite Automata. To check whether a FA A is diagnosable, we build a synchronized product $A^f \times A_1$, s.t. A^f behaves exactly like A but records in its state whether a fault has occurred, and A_1 behaves like A without the faulty runs (transitions labelled f are cut off). This corresponds to $A^f(\Delta)$ defined in section 4.1 without the clock Δ.

A *faulty run* in the product $A^f \times A_1$ is a run for which A^f reaches a faulty state of the form $(q, 1)$. To decide whether A is diagnosable we build an extended version of

$A^f \times A_1$ which is a Büchi automaton $\mathcal{B}$ [18]: $\mathcal{B}$ has a boolean variable z which records whether A^f participated in the last transition fired by $A^f \times A_1$. A state of $\mathcal{B}$ is a pair (s, z) where s is a state of $A^f \times A_1$. $\mathcal{B}$ is given by the tuple $((Q \times \{0,1\} \times Q) \times \{0,1\}, ((q_0, 0), q_0, 0), \Sigma_\tau, \longrightarrow_{\mathcal{B}}, \varnothing, R_{\mathcal{B}})$ with:

- $(s, z) \xrightarrow{\lambda}_{\mathcal{B}} (s', z')$ if (i) there exists a transition $t : s \xrightarrow{\lambda} s'$ in $A^f \times A_1$, and (ii) $z' = 1$ if λ is a move of A^f and $z' = 0$ otherwise;
- $R_{\mathcal{B}} = \{(((q, 1), q'), 1) \mid ((q, 1), q') \in A^f \times A_1\}$.

The important part of the previous construction relies on the fact that, for A to be non Σ-diagnosable, A^f should have an infinite faulty run (and take infinitely many transitions) and A_1 a corresponding non-faulty run (note that this one can be finite) giving the same observation. With the previous construction, we have [18]: A is diagnosable iff $\mathcal{L}^\omega(\mathcal{B}) = \varnothing$.

The construction for codiagnosability is an extension of the previous one adding $A_2, \cdots, A_n$ to the product. Let $\mathcal{B}^{co} = A^f \times A_1 \times \cdots \times A_n$ with A_i defined in section 4.1. In $\mathcal{B}^{co}$ we again use the variable z to indicate whether A^f participated in the last move. Define the set of repeated states of $\mathcal{B}^{co}$ by: $R_{\mathcal{B}^{co}} = \{(((q, 1), \overline{q}), 1) \mid ((q, 1), \overline{q}) \in A^f \times A_1 \times \cdots \times A_n\}$. By construction, a state in $R_{\mathcal{B}^{co}}$ is: (1) faulty as it contains a component $(q, 1)$ for the state of A^f and (2) A^f participated in the last move as $z = 1$. It follows that:

Lemma 4. *A is $\mathcal{E}$-codiagnosable iff $\mathcal{L}^\omega(\mathcal{B}^{co}) = \varnothing$.*

Theorem 7. *Problem 5 is PSPACE-complete for Deterministic Finite Automata.*

Codiagnosability for Timed Automata. Checking diagnosability for timed automata requires an extra step in the construction of the equivalent of automaton $\mathcal{B}$ defined above: indeed, for TA, a run having infinitely many discrete steps could well be *zeno*, i.e., the duration of such a run can be finite. This extra step in the construction was first presented in [6]. It can be carried out by adding a special timed automaton *Div* with two locations $\{0, 1\}$ and synchronizing it with $A^f \times A_1$. If we use $F = \varnothing$ and $R = \{1\}$ for *Div*, any accepted run is *time divergent* and thus cannot be zeno. Let $\mathcal{D} = A^f \times \mathit{Div} \times A_1$ and let $F_{\mathcal{D}} = \varnothing$ and $R_{\mathcal{D}}$ be the set of states where A^f is in a faulty location and *Div* is in location 1. For standard fault diagnosis, the following holds [6,18]: A is diagnosable iff $\mathcal{L}^\omega(\mathcal{D}) = \varnothing$.

The construction to check codiagnosability is obtained by adding $A_2, \cdots, A_n$ in the product. Let $\mathcal{D}^{co} = A^f \times \mathit{Div} \times A_1 \times \cdots \times A_n$.

Lemma 5. *A is $\mathcal{E}$-codiagnosable iff $\mathcal{L}^\omega(\mathcal{D}^{co}) = \varnothing$.*

Theorem 8. *Problem 5 is PSPACE-complete for Timed Automata.*

4.3 Optimal Delay (Problem 6)

Using the results for checking $\mathcal{E}$-codiagnosability and $(\Delta, \mathcal{E})$-codiagnosability, we obtain algorithms for computing the optimal delay.

Lemma 4 reduces codiagnosability of FA to Büchi emptiness on a product automaton. The number of states of the automaton $\mathcal{B}^{co}$ is bounded by $4 \cdot |A|^n$, and the number of faulty states by $2 \cdot |A|^n$. This implies that:

Proposition 2. *Let A be a FA. If A is $\mathcal{E}$-codiagnosable, then A is $(2 \cdot |A|^n, \mathcal{E})$-codiagnosable.*

From Proposition 2, we can conclude that:

Theorem 9. *Problem 6 can be solved in PSPACE for FA.*

For timed automata, a similar reasoning can be done on the region graph of $\mathcal{D}^{co}$. If a TA A is $\mathcal{E}$-codiagnosable, there cannot be any cycle with faulty locations in $RG(\mathcal{D}^{co})$. Otherwise there would be a non-zeno infinite word in $\mathcal{L}(\mathcal{D}^{co})$ and thus an infinite time-diverging faulty run in A, with corresponding non-faulty runs in each A_i, giving the same observation. Let K be the size of $RG(\mathcal{D}^{co})$. If A is $\mathcal{E}$-codiagnosable, then a faulty state in $RG(\mathcal{D}^{co})$ can be followed by at most K states. Otherwise a cycle in the region graph would occur and thus $\mathcal{L}^{\omega}(\mathcal{D}^{co})$ would not be empty. This also implies that all the states (s, r) in $RG(\mathcal{D}^{co})$ that can follow a faulty state must have a *bounded* region. As the amount of time that can elapse in one region is at most 1 time unit[4], the maximum duration of a faulty run in $\mathcal{D}^{co}$ is bounded by K. This implies that:

Proposition 3. *Let A be a TA. If A is $\mathcal{E}$-codiagnosable, then A is $(K, \mathcal{E})$-codiagnosable with $K = |RG(\mathcal{D}^{co})|$.*

The size of the region graph of $\mathcal{D}^{co}$ is bounded by $|L|^{n+1} \cdot ((n+1)|X|+1)! \cdot 2^{(n+1)|X|+1} \cdot M^{(n+1)|X|+1}$. Thus the encoding of constant K has size $O(n \cdot |A|)$.

Theorem 10. *Problem 6 can be solved in PSPACE for Timed Automata.*

5 Synthesis of Codiagnosers

The reader is referred to the extended version of this paper [14] for a detailed presentation of this section. The synthesis of codiagnosers for FA and TA can be carried out by extending the known construction for diagnosers [3].

The construction of a diagnoser for timed automata [6] consists in computing *on-the-fly* the current possible states of the timed automaton A^f after reading a timed word w. This procedure is effective but gives a diagnoser which is a Turing machine. The machine computes a state estimate of A after each observable event, and if it contains only faulty states, it announces a fault.

Obviously the same construction can be carried out for codiagnosis: we define the Turing machines $M_i, 1 \leq i \leq n$ that estimate the state of A. When one M_i's estimate on an input Σ_i-trace w contains only faulty states, we set $D_i(w) = 1$ and 0 otherwise. This tuple of Turing machines is a $(\Delta, \mathcal{E})$-codiagnoser.

Computing the estimates with Turing machines might be too expensive to be implemented at runtime. More efficient and compact codiagnosers might be needed with reasonable computation times. In the next section, we address the problem of codiagnosis for TA under *bounded resources*.

[4] The constants in the automata are integers.

6 Codiagnosis with Deterministic Timed Automata

The fault diagnosis problem using timed automata has been introduced and solved by P. Bouyer *et al.* in [7]. The problem is to determine, given a TA A, whether there exists a *diagnoser* D for A, that can be represented by a deterministic timed automaton.

6.1 Fault Diagnosis with Deterministic Timed Automata

When synthesizing (deterministic) timed automata, an important issue is the amount of *resources* the timed automaton can use: this can be formally defined [19] by the (number of) clocks, Z, that the automaton can use, the maximal constant $\max$, and a *granularity* $\frac{1}{m}$. As an example, a TA of resource $\mu = (\{c, d\}, 2, \frac{1}{3})$ can use two clocks, c and d, and the clocks constraints using the rationals $-2 \leq k/m \leq 2$ where $k \in \mathbb{Z}$ and $m = 3$. A *resource* μ is thus a triple $\mu = (Z, \max, \frac{1}{m})$ where Z is finite set of clocks, $\max \in \mathbb{N}$ and $\frac{1}{m} \in \mathbb{Q}_{>0}$ is the *granularity*. DTA$_\mu$ is the class of DTA of resource μ.

Remark 2. Notice that the number of locations of the DTA in DTA$_\mu$ is not bounded and hence this family has an infinite (yet countable) number of elements.

If a TA A is Δ-diagnosable with a diagnoser that can be represented by a DTA D with resource μ, we say that A is (Δ, D)-diagnosable. P. Bouyer *et al.* in [7] considered the problem of deciding whether there exists a DTA diagnoser with resource μ:

Problem 7 (Δ-DTA-Diagnosability [7])
INPUTS: *A TA $A = (L, l_0, X, \Sigma_{\tau,f}, E, Inv)$, $\Delta \in \mathbb{N}$, a resource $\mu = (Z, \max, \frac{1}{m})$.*
PROBLEM: *Is there any $D \in$ DTA$_\mu$ s.t. A is (Δ, D)-diagnosable ?*

Theorem 11 (P. Bouyer *et al.*, [7]). *Problem 7 is 2EXPTIME-complete.*

The solution to the previous problem is based on the construction of a *two-player safety game*, $G_{A,\Delta,\mu}$. In this game a set of states, *Bad*, must be avoided for A to be Δ-diagnosable. The most permissive winning strategy gives the *set* of all DTA$_\mu$ diagnosers (the most permissive diagnosers) which can diagnose A (or $\varnothing$ is there is none). We refer to the extended version [14], section 6.1 for a detailed presentation of this construction.

6.2 Algorithm for Codiagnosability

In this section we include the alphabet Σ of a DTA in the resource μ and write $\mu = (\Sigma, Z, \max, \frac{1}{m})$.

Problem 8 (Δ-DTA-Codiagnosability)
INPUTS: *A TA $A = (L, l_0, X, \Sigma_{\tau,f}, E, Inv)$, $\Delta \in \mathbb{N}$, and a family of resources $\mu_i = (\Sigma_i, Z_i, \max_i, \frac{1}{m_i}), 1 \leq i \leq n$ with $\Sigma_i \subseteq \Sigma$.*
PROBLEM: *Is there any codiagnoser $\overline{D} = (D_1, D_2, \cdots, D_n)$ with $D_i \in$ DTA$_{\mu_i}$ s.t. A is $(\Delta, \overline{D})$-codiagnosable ?*

	Δ-Codiagnosability	Codiagnosability	Optimal Delay	Synthesis (Bounded Resources)
FA	**PSPACE-C.** PTIME [5,4]	**PSPACE-C.** PTIME [5,4]	**PSPACE** PTIME [5,4]	**EXPTIME** EXPTIME [3]
TA	**PSPACE-C.** PSPACE-C. [6]	**PSPACE-C.** PSPACE-C. [6]	**PSPACE** PSPACE [18]	**2EXPTIME-C.** 2EXPTIME-C. [7]

Table 1. Summary of the Results

To solve Problem 8, we extend the algorithm given in [7] for DTA-diagnosability. Let G^i be the game G_{A,Δ,μ_i} and *Bad*$_i$ the set of bad states. Given a strategy f_i, we let $f_i(G^i)$ be the outcome[5] of G^i when f_i is played by Player 0. Given $w \in TW^*(\Sigma)$ and a DTA A on Σ, we let *last*(w, A) be the location reached when w is read by A.

Lemma 6. *A is $(\Delta, \overline{D})$-codiagnosable iff there is a tuple of strategies $\overline{f}$ s.t.*

$$(1)\ \forall 1 \leq i \leq n, \overline{f}[i] \textit{ is state-based on the game } G^i, \textit{ and}$$

$$(2)\ \forall w \in Tr(A) \quad \begin{cases} \textit{If } S_i = \textit{last}(\boldsymbol{\pi}_{\Sigma_i}(w), f_i(G^i)), 1 \leq i \leq n, \\ \textit{then } \exists 1 \leq j \leq n, \textit{ s.t. } S_j \notin \textit{Bad}_j. \end{cases}$$

Item (2) of Lemma 6 states that there is no word in A for which all the Player 0 in the games G^i are in bad states. The strategies for each Player 0 are not necessarily winning in each G^i, but there is always one Player 0 who has not lost the game G^i. From the previous Lemma, we can obtain the following result:

Theorem 12. *Problem 8 is 2EXPTIME-complete.*

7 Conclusion & Future Work

Table 1 gives an overview of the results described in this paper (bold face) for the co-diagnosis problems in comparison with the results for the diagnosis problems (second line, normal face). Our ongoing work is to extend the results on *diagnosis using dynamic observers* [20,17] to the codiagnosis framework.

Ackowledgements. The author would like to thank the anonymous reviewers for their helpful comments.

References

1. Ramadge, P., Wonham, W.: Supervisory control of a class of discrete event processes. SIAM Journal of Control and Optimization **25**(1) (1987) 1202–1218
2. Ramadge, P., Wonham, W.: The control of discrete event systems. Proc. of the IEEE **77**(1) (1989) 81–98

[5] $f_i(G^i)$ is a timed transition system.

3. Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, D.: Diagnosability of discrete event systems. IEEE Transactions on Automatic Control **40**(9) (September 1995)
4. Jiang, S., Huang, Z., Chandra, V., Kumar, R.: A polynomial algorithm for testing diagnosability of discrete event systems. IEEE Transactions on Automatic Control **46**(8) (August 2001)
5. Yoo, T.S., Lafortune, S.: Polynomial-time verification of diagnosability of partially-observed discrete-event systems. IEEE Transactions on Automatic Control **47**(9) (September 2002) 1491–1495
6. Tripakis, S.: Fault diagnosis for timed automata. In Damm, W., Olderog, E.R., eds.: Proceedings of the International Conference on Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT'02). Volume 2469 of LNCS., Springer Verlag (2002) 205–224
7. Bouyer, P., Chevalier, F., D'Souza, D.: Fault diagnosis using timed automata. In Sassone, V., ed.: Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05). Volume 3441 of LNCS., Edinburgh, U.K., Springer Verlag (April 2005) 219–233
8. Debouk, R., Lafortune, S., Teneketzis, D.: Coordinated decentralized protocols for failure diagnosis of discrete event systems. Discrete Event Dynamic Systems **10**(1-2) (2000) 33–86
9. Wang, Y., Yoo, T.S., Lafortune, S.: Diagnosis of discrete event systems using decentralized architectures. Discrete Event Dynamic Systems **17**(2) (2007) 233–263
10. Qiu, W., Kumar, R.: Decentralized failure diagnosis of discrete event systems. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans **36**(2) (2006) 384–395
11. Basilio, J., Lafortune, S.: Robust codiagnosability of discrete event systems. In Society, I.C., ed.: Proceedings of the American Control Conference (ACC'09). (2009) 2202–2209
12. Holzmann, G.J.: Software model checking with spin. Advances in Computers **65** (2005) 78–109
13. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In Bernardo, M., Corradini, F., eds.: Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004. Volume 3185 of LNCS., Springer Verlag (September 2004) 200–236
14. Cassez, F.: The complexity of codiagnosability for discrete event and timed systems. Research report, National ICT Australia (April 2010) 24 pages, document available from arXiv `http://arxiv.org/abs/1004.2550`.
15. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science **126** (1994) 183–235
16. Kozen, D.: Lower bounds for natural proof systems. In: FOCS, IEEE (1977) 254–266
17. Cassez, F., Tripakis, S.: Fault diagnosis with static or dynamic diagnosers. Fundamenta Informaticae **88**(4) (November 2008) 497–540
18. Cassez, F.: A Note on Fault Diagnosis Algorithms. In: 48th IEEE Conference on Decision and Control and 28th Chinese Control Conference, Shanghai, P.R. China, IEEE Computer Society (December 2009)
19. Bouyer, P., D'Souza, D., Madhusudan, P., Petit, A.: Timed control with partial observability. In Hunt, Jr, W.A., Somenzi, F., eds.: Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03). Volume 2725 of LNCS., Boulder, Colorado, USA, Springer (July 2003) 180–192
20. Cassez, F., Tripakis, S., Altisen, K.: Sensor minimization problems with static or dynamic observers for fault diagnosis. In: 7th Int. Conf. on Application of Concurrency to System Design (ACSD'07), IEEE Computer Society (2007) 90–99
21. Aceto, L., Laroussinie, F.: Is your model checker on time? on the complexity of model checking for timed modal logics. J. Log. Algebr. Program. **52-53** (2002) 7–51

A Proofs for Section 2

A.1 Proof of Proposition 1

Proof. Let $A_1, A_2, \cdots, A_n$ be n deterministic automata with accepting states $F_1, F_2, \cdots, F_n$ on the alphabet Σ. Let λ be a fresh letter not in Σ. Define automaton A'_i by: from any state q in F_i, add a transition $(q, \lambda, \bot)$ where $\bot$ is new state. Let $F'_1 = \{\bot\}$ and F'_i be all the states of A'_i. It is clear that $\mathcal{L}^*(A'_1) = \mathcal{L}^*(A_1).\lambda$.

We can prove that $\cap_{i=1}^n \mathcal{L}^*(A_i) \neq \varnothing \iff \cap_{i=1}^n \mathcal{L}^*(A'_i) \neq \varnothing$. Indeed, assume $w \in \cap_{i=1}^n \mathcal{L}^*(A_i) \neq \varnothing$. Then $A_1 \times A_2 \times \cdots \times A_n$ reaches the state $(q_1, q_2, \cdots, q_n)$ after reading w and $\forall 1 \leq i \leq n, q_i \in F_i$. Thus in $A'_1 \times A'_2 \times \cdots \times A'_n$ the same state can be reached and then λ can be fired in the product leading to $(\bot, \bot, \cdots, \bot)$. Conversely, if a word w is accepted by the product $A'_1 \times \cdots \times A'_n$, w must end with λ. Let $w = u.\lambda \in \cap_{i=1}^n \mathcal{L}^*(A'_i) \neq \varnothing$. After reading u the state of the product must be $(q_1, q_2, \cdots, q_n)$ with $\forall 1 \leq i \leq n, q_i \in F_i$, and the transitions fired when reading u are also in $A_1 \times A_2 \times \cdots \times A_n$ which implies $u \in \cap_{i=1}^n \mathcal{L}^*(A_i)$. □

A.2 Proof of Theorem 4

Proof. PSPACE-hardness follows from the fact that checking $\cap_{i=1}^n \mathcal{L}^*(A_i) \neq \varnothing$ on finite automata is already PSPACE-hard [16] or alternatively because reachability for timed automata is PSPACE-hard [15].

PSPACE-easiness can be established as Theorem 31 (section 4.1) of [21]: the regions of the product of TA A_i can be encoded in polynomial space in the size of the clock constraints of the product automaton. An algorithm to check emptiness is obtained by: 1) guessing a sequence of pairs (location,region) in the product automaton and 2) checking whether it is accepted. This can be done in NPSPACE and by Savitch's Theorem in PSPACE. □

A.3 Proof of Theorem 5

Proof. PSPACE-hardness follows from the reduction of Problem 2 to Problem 3 or again because checking Büchi emptiness for timed automata is PSPACE-hard [15].

Consider the product automaton A^+ the construction of which is described at the end of section 2.5. PSPACE-easiness is established by: 1) guessing a state of $RG(A^+)$ of the form $((\bar{\ell}, n), r)$ and 2) checking it is reachable from the initial state (PSPACE) and reachable from itself (PSPACE). As n is encoded in binary the result follows. □

B Proofs for Section 3

B.1 Proof of Lemma 1

Proof.

– If part. Assume equation (1) holds and A is $(\Delta, \mathcal{E})$-codiagnosable. Then there is a codiagnoser $\overline{D} = (D_1, \cdots, D_n)$ satisfying Definition 3. For each ϱ_i we must have $D_i(\boldsymbol{\pi}_i(tr(\varrho_i))) = 0$ because each ϱ_i is non faulty. But we must also have for at least one index i, $D_i(\boldsymbol{\pi}_i(tr(\varrho_i))) = D_i(\boldsymbol{\pi}_i(tr(\varrho))) = 1$ because ϱ is Δ-faulty, which is impossible.

– Only If part. Assume A is not $(\Delta, \mathcal{E})$-codiagnosable and $\forall \varrho \in \mathit{Faulty}_{\geq \Delta}(A)$, equation (1) does not hold. In this case, there is an index $1 \leq i \leq n$ s.t. :

$$\forall \varrho' \in \mathit{NonFaulty}(A), \quad \boldsymbol{\pi}_i(\mathit{tr}(\varrho)) \neq \boldsymbol{\pi}_i(\mathit{tr}(\varrho')).$$

Define $D_i(w) = 1$ when $w \in \boldsymbol{\pi}_i(\mathit{Faulty}^{\mathit{tr}}_{\geq \Delta}(A)) \setminus \boldsymbol{\pi}_i(\mathit{NonFaulty}^{\mathit{tr}}(A))$ and 0 otherwise. Then $\overline{D} = (D_1, \cdots, D_n)$ is a Δ-codiagnoser for A. Indeed, let $\varrho \in \mathit{NonFaulty}(A)$. Then $\boldsymbol{\pi}_i(\mathit{tr}(\varrho)) \in \boldsymbol{\pi}_i(\mathit{NonFaulty}^{\mathit{tr}}(A))$ and thus $D_i(\boldsymbol{\pi}_i(\mathit{tr}(\varrho))) = 0$. Let $\varrho \in \mathit{Faulty}_{\geq \Delta}(A)$ and assume $D_i(\boldsymbol{\pi}_i(\mathit{tr}(\varrho))) = 0$ for each $1 \leq i \leq n$. By definition of D_i we must have $\boldsymbol{\pi}_i(\mathit{tr}(\varrho)) \in \boldsymbol{\pi}_i(\mathit{NonFaulty}^{\mathit{tr}}(A))$. In this case, there is some run $\varrho_i \in \mathit{NonFaulty}(A)$ s.t. $\boldsymbol{\pi}_i(\mathit{tr}(\varrho)) = \boldsymbol{\pi}_i(\mathit{tr}(\varrho_i))$ and thus equation (1) holds which contradicts the initial assumption. □

B.2 Proof of Lemma 2

Proof. Assume equation 2 does not hold and let $w \in \mathit{Faulty}^{\mathit{tr}}_{\geq \Delta}(A)$, and for each $1 \leq i \leq n$, $w \in \boldsymbol{\pi}_i^{-1}\big(\boldsymbol{\pi}_i(\mathit{NonFaulty}^{\mathit{tr}}(A))\big)$. This implies that:

– $\exists \varrho \in \mathit{Faulty}_{\geq \Delta}(A)$ s.t. $\mathit{tr}(\varrho) = w$;
– for each i, $w \in \boldsymbol{\pi}_i^{-1}\big(\boldsymbol{\pi}_i(\mathit{NonFaulty}^{\mathit{tr}}(A))\big)$ and $\boldsymbol{\pi}_i(w) \in \boldsymbol{\pi}_i(\mathit{NonFaulty}^{\mathit{tr}}(A))$. Thus, there is a run $\varrho_i \in \mathit{NonFaulty}(A)$, s.t. $\boldsymbol{\pi}_i(w) = \boldsymbol{\pi}_i(\mathit{tr}(\varrho)) = \boldsymbol{\pi}_i(\mathit{tr}(\varrho_i))$ and as equation (1) of Lemma 1 is satisfied, A is not $(\Delta, \mathcal{E})$-codiagnosable.

For the converse, assume A is not $(\Delta, \mathcal{E})$-codiagnosable. By Lemma 1, equation (1) is satisfied and:

– there is a run ϱ with $\mathit{tr}(\varrho) \in \mathit{Faulty}^{\mathit{tr}}_{\geq \Delta}(A)$;
– for each i, there is some $\varrho_i \in \mathit{NonFaulty}(A)$ s.t. $\boldsymbol{\pi}_i(\mathit{tr}(\varrho)) = \boldsymbol{\pi}_i(\mathit{tr}(\varrho_i))$. Hence $\mathit{tr}(\varrho) \in \boldsymbol{\pi}_i^{-1}(\boldsymbol{\pi}_i(\mathit{NonFaulty}^{\mathit{tr}}(A)))$ for each i,

and this implies that equation 2 does not hold. □

C Proofs for Section 4

C.1 Proof of Lemma 3

Proof. The sets of clocks of the A_i's and $A^f(\Delta)$ are disjoint: for each $1 \leq i < j \leq n$, $X_i \cap X_j = \varnothing$ and $X_i \cap X = \varnothing$. It follows from Fact 1 that $\mathcal{L}^*(\mathcal{B}) = \mathcal{L}^*(A^f(\Delta)) \cap \big(\bigcap_{i=1}^n \mathcal{L}^*(A_i^*)\big)$. By Lemma 2 and the construction of $A^f(\Delta)$ and the A_i's, the result follows. □

C.2 Proof of Theorem 6

Proof. PSPACE-easiness follows from the polynomial reduction above and Lemma 3. PSPACE-hardness is obtained by reducing the variant of the *intersection emptiness problem* for DTA to the $(\Delta, \mathcal{E})$-codiagnosability problem. This problem is PSPACE-hard (Proposition 1).

Let $A_i, 1 \leq i \leq n$, be n deterministic finite automata over the alphabet Σ. Assume A_1 has one accepting state and for $A_2, \cdots, A_n$ all states are accepting.

We construct B as shown on Figure 1: $a_2, \cdots, a_n$ are fresh letters not in Σ; the target state of a_i is the initial state of A_i. The initial state of B is ι. Let $\Sigma_i = \Sigma \setminus \{a_i\}$ for each $2 \leq i \leq n$. From the final state of A_1 there is a transition labeled f to a new state e.

We can prove that B is $(1, \mathcal{E})$-codiagnosable if and only if $\cap_{i=1}^n \mathcal{L}^*(A_i) = \varnothing$ with $\mathcal{E} = (\Sigma_i)_{1 \leq i \leq n}$. Assume $w \in \cap_{i=1}^n \mathcal{L}^*(A_i) \neq \varnothing$. Take the run of trace $\tau.w.f.\tau$ in B. This run is 1-faulty. For each $2 \leq i \leq n$, there is a run of trace $a_i.w$ which is non faulty. Moreover, $\boldsymbol{\pi}_i(a_i.w) = w$ and thus B is not $(1, \mathcal{E})$-codiagnosable.

Now, assume B is not $(1, \mathcal{E})$-codiagnosable. There is a 1-faulty run, and this must be a run of trace $\tau.w.f.\tau$ with $w \in \mathcal{L}^*(A_1)$, and for each $2 \leq i \leq n$, there is a non-faulty run ϱ_i the trace of which is u_i, with $\boldsymbol{\pi}_i(u_i) = w$. It must be the case that $u_i = a_i.w_i$ as otherwise $\boldsymbol{\pi}_i(u_i)$ would start with $a_k, k \neq i$ and thus it would be impossible to have $\boldsymbol{\pi}_i(u_i) = w$. As $u_i = a_i.w_i$, $\boldsymbol{\pi}_i(u_i) = w_i = w$, and $w \in \mathcal{L}^*(A_i)$, it follows that $w \in \cap_{i=1}^n \mathcal{L}^*(A_i)$ and thus $\cap_{i=1}^n \mathcal{L}^*(A_i)$ is not empty.
Finally $\cap_{i=1}^n \mathcal{L}^*(A_i) \neq \varnothing$ if and only if B is not $(1, \mathcal{E})$-codiagnosable.

The size of B is in $O(\sum_{i=1}^n |A_i| + n)$ which is equal to $O(\sum_{i=1}^n |A_i|)$ as $|A_i| \geq 1$. The size of the input for Problem 4 is thus $O(\sum_{i=1}^n |A_i|) + n \cdot (|\Sigma| + n)$ which is quadratic and thus polynomial in $\sum_{i=1}^n |A_i|$. The intersection emptiness problem for DTA is polynomially reducible to the $(\Delta, \mathcal{E})$-codiagnosability Problem and Problem 4 is PSPACE-hard for DTA. □

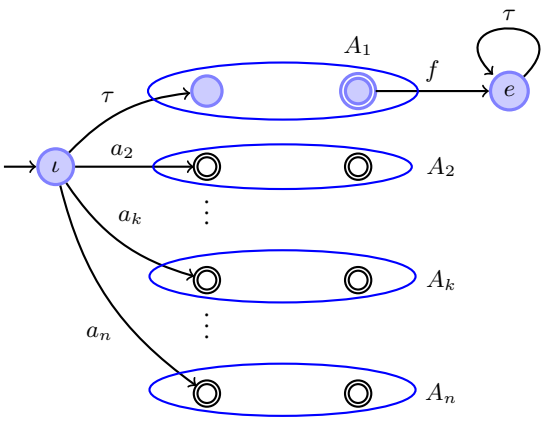

Fig. 1. Reduction for Theorem 6: Automaton B

C.3 Proof of Theorem 7

Proof. PSPACE-easiness follows form the fact that checking whether $\mathcal{L}^\omega(\mathcal{B}^{co}) = \varnothing$ can be done in PSPACE (Theorem 3). PSPACE-hardness follows from a reduction of

Problem 1 to Problem 5 using the same encoding as the one given in the proof of Theorem 6: the automaton B of Fig. 1 is not $(\Delta, \mathcal{E})$-codiagnosable for any $\Delta \in \mathbb{N}$. □

C.4 Proof of Theorem 8

Proof. The size of $\mathcal{D}^{co}$ is in $O((n+1) \cdot |A|)$ and thus polynomial in the size of the input of Problem 5 ($|A| + n \cdot |\Sigma|$). PSPACE-easiness follows because the intersection emptiness problem for Büchi automata can be solved in PSPACE. PSPACE-hardness holds because it is already PSPACE-hard for FA. □

C.5 Proof of Proposition 2

Proof. If $\mathcal{L}(\mathcal{B}^{co}) = \varnothing$ there cannot be a faulty run of length more than $2 \cdot |A|^n$ otherwise at least one faulty state s will be encountered twice on this run, and in this case we could construct an infinite faulty run which contradicts the fact that $\mathcal{L}(\mathcal{B}^{co}) = \varnothing$. □

C.6 Proof of Theorem 9

Proof. Checking whether A is $\mathcal{E}$-codiagnosable can be done in PSPACE. If the result is "yes", we can do a binary search for the optimal delay: start with $\Delta = 2 \cdot |A|^n$, and check whether A is $(\Delta, \mathcal{E})$-codiagnosable. If "yes", divide Δ by 2 and so on. The encoding of $2 \cdot |A|^n$ has size $O(n \cdot \log |A|)$ and thus is polynomial in the size of the inputs of Problem 6. □

C.7 Proof of Theorem 10

Proof. Checking whether a TA A is $\mathcal{E}$-codiagnosable can be done in PSPACE. If the result is "yes", we can do a binary search for the maximum delay: start with $\Delta = K = |RG(\mathcal{B}^{co})|$, and check whether A is $(\Delta, \mathcal{E})$-codiagnosable. If "yes", divide Δ by 2 and so on. The encoding of K has size $O(n \cdot |A|)$ and thus is polynomial in the size of the input of Problem 6. □

D Proofs for Section 6

D.1 Proof of Lemma 6

Proof.

If part. Assume there is a tuple of state-based strategies $\overline{f} = (f_1, f_2, \cdots, f_n)$ on each game G^i, s.t. (2) is satisfied. From (1), each choice of Player 0 in G^i determines one transition from each square state (see the definition of G^i and square states in section 6.1). Thus the graph of G^i can be folded into a set of transitions $q \xrightarrow{g,a,Y} q'$ if the choice of Player 0 is g, a, Y in square state (q, g, a). This gives a DTA $G^{i,c}$. We can then build a diagnoser D_i defined by the DTA as follows: (i) for each state $q = \{(\ell_1, r_1), \cdots, (\ell_k, r_k)\}$ in $G^{i,c}$, if all the ℓ_j are Δ-faulty, q is accepting; (ii)

given $w \in \mathit{Tr}(A)$, if $\boldsymbol{\pi}_{\Sigma_i}(w) \in \mathcal{L}(G^{i,c})$, let $D_i(\boldsymbol{\pi}_{\Sigma_i}(w)) = 1$ and otherwise 0. $\overline{D}$ is a Δ-codiagnoser for A. Indeed, let $w \in \mathit{NonFaulty}^{tr}(A)$. In each game $G^{i,c}$, we cannot reach a Δ-faulty state because of (2). Hence $\sum_{i=1}^{n} \overline{D}[i] = 0$. Now assume $w \in \mathit{Faulty}^{tr}_{\geq\Delta}(A)$: In each $G^{i,c}$ we must reach a state q_i containing a Δ-faulty state. By (2), there is some j s.t. $q_j \notin \mathit{Bad}_j$ and this implies that q_j is made only of Δ-faulty states and q_j is accepting, thus $\overline{D}[j](\boldsymbol{\pi}_{\Sigma_j}(w)) = 1$.

Only If part. For this part we first show that a tuple of strategies $\overline{f}$ exists and then address the state-based problem. Let $\overline{D} = (D_1, D_2, \cdots, D_n)$ be the tuple of DTA that diagnoses A. For each game G^i, define the strategy f_i by: let $\varrho = (g_1, \lambda_1)(g_1, \lambda_1, Y_1) \cdots (g_{k-1}, \lambda_{k-1})(g_{k-1}, \lambda_{k-1}, Y_{k-1})(g_k, \lambda_k)$ be a run in G^i; $f_i(\varrho) = (g, a, Y)$ if in D_i the symbolic sequence $(g_1, \lambda_1) \cdots (g_k, \lambda_k)$ reaches a location ℓ and there is a transition $(\ell, (g, a, Y), \ell')$ in D_i. By assumption, as $\overline{D}$ is a Δ-codiagnoser, for each $w \in \mathit{Faulty}^{tr}_{\geq\Delta}(A)$, there is at least one D_j which reaches an accepting state after reading $\boldsymbol{\pi}_{\Sigma_j}(w)$.

As a consequence, in the corresponding game, G^j, the state reached is made only of Δ-faulty states. Indeed, if a non-faulty state is reachable, then the word w is also the projection of a non faulty run. Hence D_j should announce 0 which is a contradiction.

If $w \in \mathit{NonFaulty}^{tr}(A)$, all the states reached in each G^i are non faulty.

Now assume we have the strategies $f_i, 1 \leq i \leq n$. We can construct state-based strategies on each game G^i. Given f_1, (not necessarily winning) on G^1, let T_1 be the set of bad states reachable in $f_1(G^1)$. Define the language $\mathcal{L}_1$ to be the set of words $w \in \mathit{Tr}(A)$ s.t. a state in T_1 is reachable in $f_1(G^1)$ when reading $\boldsymbol{\pi}_{\Sigma_1}(w)$. These are the words on which f_1 is not winning in G^1.

Let $\mathit{Reach}(f_1(G^1))$ be the set of states reachable in G^1. There is a strategy (f_1) to avoid $B_1 = \mathit{Reach}(G^1) \setminus \mathit{Reach}(f_1(G^1))$. Hence there is a state-based strategy f_1' that avoids B_1.

Let $1 \leq i < n$. Consider the game $f_{i+1}(G^{i+1})$ restricted to the (projections of the) words $w \in \mathcal{L}_i$. The idea is that on $\mathcal{L}_i$, a strategy $f_j, j \leq i$ is winning in G^j. In this restricted game, we define the set T_{i+1} of bad states that are still reachable. Let $\mathcal{L}_{i+1}$ be the set of words $w \in \mathit{Tr}(A)$ s.t. a state in T_{i+1} is reachable in the restricted timed transition system $f_{i+1}(G^{i+1})$.

Notice that we can construct a state-based strategy f_i' which avoids the same states as f_i does. For each restricted game $f_i'(G^i)$ we define the diagnoser D_i as before. If for some i, $\mathcal{L}_i = \varnothing$, we can define the diagnosers $D_k, k \geq i$ to always announce 0 for each word.

$\overline{f'}$ is a $(\Delta, \mathcal{E})$-codiagnoser for A and all the $\overline{f'}[i]$ are state-based on G^i. □

D.2 Proof of Theorem 12

Proof. 2EXPTIME-hardness follows from Theorem 11, from [7]. 2EXPTIME easiness is obtained using the following algorithm:

1. compute the games $G^i, 1 \leq i \leq n$;
2. select a state-based strategy on each game G^i;

3. check condition (2) of Lemma 6.

The sizes of the games G^i are doubly exponential in A, Δ and the resources μ_i (recall that Σ_i is included in μ_i). There is a doubly exponential number of state-based strategies for each game G^i. Once selected we have a DTA $G^{i,c}$.

Checking condition (2) of Lemma 6 can be done on the product $A(\Delta) \times G^{1,c} \times \cdots \times G^{n,c}$. It amounts to deciding whether a location in $L_3 \times \mathit{Bad}_1 \times \cdots \mathit{Bad}_n$ is reachable. Reachability can be checked in PSPACE for product of TA (Theorem 2). As the size of the input is doubly exponential in the size of A, this results in a 2EXPSPACE algorithm.

Nevertheless, there is no exponential blow up in the number of clocks of the product. Actually the size of $\mathit{RG}(A(\Delta) \times G^{1,c} \times \cdots \times G^{n,c})$ is $|L| \cdot 2^{2^{|A|+|\mu_1|}} \cdot \cdots \cdot 2^{2^{|A|+|\mu_n|}} \cdot (n \cdot |X|)! \cdot 2^{n \cdot |X|} \cdot K^{n \cdot |X|}$ with K the maximal constant in A, Δ, and the resources μ_i. It is doubly exponential in the size of A, Δ and the resources μ_i. Reachability can be checked in linear time on this graph and thus in doubly exponential time in the size of A, Δ and the resources. Step 3 above is done at most a doubly exponential number of times and the result follows. □