A Note on Fault Diagnosis Algorithms

Franck Cassez

A Note on Fault Diagnosis Algorithms

Franck Cassez, *Member, IEEE*

***Abstract*— In this paper we review algorithms for checking diagnosability of discrete-event systems and timed automata. We point out that the diagnosability problems in both cases reduce to the emptiness problem for (timed) Büchi automata. Moreover, it is known that, checking whether a discrete-event system is diagnosable, can also be reduced to checking bounded diagnosability. We establish a similar result for timed automata. We also provide a synthesis of the complexity results for the different fault diagnosis problems.**

Note: This paper is an extended version of the paper published in the proceedings of CDC'09.

I. Introduction

Discrete-event systems [1], [2] (DES) can be modelled by finite automata over an alphabet of *observable* events Σ. To address decision problems under *partial observation* of DES, it is sufficient to add a special event τ which represents all the *unobservable* actions.

The *Fault diagnosis problem* is a typical example of a problem under partial observation. We assume 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 and *al.* [3] and the reader is referred to this paper for a clear and exhaustive introduction to the subject[1]. The aim of fault diagnosis is to detect *faulty* sequences of the DES by observing only the events in Σ. A *faulty* sequence is a sequence of the DES containing an occurrence of event f. We assume that 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 (a word in Σ^*) was produced by a faulty sequence (in $(\Sigma \cup \{\tau, f\})^*$) 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 occurred, although it is not observable. If a diagnoser can detect a fault at most Δ steps[2] after it 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 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 [5], [6]. Nevertheless the size of the diagnoser can be exponential as it involves a determinization step. The extension of this DES framework to timed automata [7] (TA) has been proposed by S. Tripakis [8], and he proved that the problem of checking diagnosability of a timed automaton is PSPACE-complete. In the timed case, the diagnoser may be a Turing machine. In a subsequent work by P. Bouyer and *al.* [9], the problem of checking whether a timed automaton is diagnosable by a diagnoser which is a *deterministic* timed automaton was studied (we will not refer to this work in this paper.)

The algorithms proposed in the DES framework [5], [6] and in the timed automata framework [8] rely on different assumptions and use different techniques: for example [5], [6] assumes that the DES is *live* and contains no unobservable loops; the algorithm to check the diagnosability problem then consists in checking whether a cycle exists in a suitable product automaton; the algorithm of [8] for timed automata consists in checking whether a infinite word can be accepted by a (product) Büchi automaton: the main reason for the use of a Büchi acceptance condition in this case is to ensure time divergence.

Our Contribution. In this paper, we try to put into perspective the results of [5], [6], [8], [10] by giving a uniform presentation of the algorithms for fault diagnosis both in the DES and timed automata settings. We also establish a (not difficult but still) missing result for timed automata: diagnosability can be reduced to bounded diagnosability. Another contribution of this paper is to examine in details the complexity of the problems and this is summarized in Table I.

The results in this paper that are not new and have already been published are followed by the reference(s) in the core of the text of after the Theorem keyword.

One such result is Theorem 3 which already appeared in [10]. It generalizes the previous results of [5], [6] and shows that fault diagnosis reduces to Büchi emptiness for DES. This has some interesting consequences regarding the algorithmic aspects of the problem as well as the tools that can be used to verify diagnosability. These considerations (Section IV) might be of interest for the DES community.

Organisation of the Paper. Section II recalls the definitions of timed automata. Section III introduces the fault diagnosis problems we are interested in. Sections IV and V describes the algorithms to solve the diagnosability problems respect-

Franck Cassez is with National ICT Australia & CNRS, Locked Bag 6016, The University of New South Wales, Sydney NSW 1466, Australia. franck.cassez@cnrs.irccyn.fr,Franck.Cassez@nicta.com.au

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

[1]The "companion paper" [4] focuses on a less stringent notion of diagnosability called I-diagnosability.

[2]Steps are measured by the number of events, observable or not, which occur in the DES.

ively for DES and TA. Section VI summarizes the results.

II. 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 $\mathbb{R}_{\geq 0}$ is the non-negative real numbers.

A. 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 $\mathbb{R}^X_{\geq 0}$ be the set of clock valuations over X. We let $\mathbf{0}_X$ be 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 v. Given $R \subseteq X$ and a valuation v, $v[R]$ is the valuation defined by $v[R](x) = v(x)$ if $x \notin R$ and $v[R](x) = 0$ otherwise.

B. 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. $(\mathbb{R}_{\geq 0}.\Sigma)^\omega$). We let $\mathit{Dur}(w)$ 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 $\mathit{Unt}(w)$ be the *untimed* version of w obtained by erasing all the durations in w, e.g., $\mathit{Unt}(0.4\ a\ 1.0\ b\ 2.7\ c) = abc$. 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.

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

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

C. Timed Automata

Timed automata (TA) 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 [7].

Definition 1 (Timed Automaton): 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*; for $(\ell, g, a, r, \ell') \in E$, 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. ■

An example of TA is given in Fig. 1. 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:

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

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

- $v_i + \delta \models \mathit{Inv}(\ell_i)$ 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) $v_i + \delta_i \models g_i$ and (ii) $v_{i+1} = (v_i + \delta_i)[r_i]$.

The set of finite (resp. infinite) runs from a state s is denoted $\mathit{Runs}^*(s, A)$ (resp. $\mathit{Runs}^\omega(s, A)$) and we define $\mathit{Runs}^*(A) = \mathit{Runs}^*((l_0, \mathbf{0}), A)$ and $\mathit{Runs}^\omega(A) = \mathit{Runs}^\omega((l_0, \mathbf{0}), A)$. As before $\mathit{Runs}(A) = \mathit{Runs}^*(A) \cup \mathit{Runs}^\omega(A)$. If ϱ is finite and ends in s_n, we let $\mathit{last}(\varrho) = s_n$. 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*, $\mathit{tr}(\varrho)$, of a run ϱ is the timed word $\boldsymbol{\pi}_{/\Sigma}(\delta_0 a_0 \delta_1 a_1 \cdots a_n \delta_n \cdots)$. We let $\mathit{Dur}(\varrho) = \mathit{Dur}(\mathit{tr}(\varrho))$. For $V \subseteq \mathit{Runs}(A)$, we let $\mathit{Tr}(V) = \{\mathit{tr}(\varrho) \mid \varrho \in V\}$, 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 finite automaton (FA) is a particular TA with $X = \varnothing$. Consequently guards and invariants are vacuously true and time elapsing transitions do not exist. We write $A = (L, l_0, \Sigma_\tau, E, F, R)$ for a FA. A run is thus a sequence of the form:

$$\varrho \quad = \quad \ell_0 \xrightarrow{a_0} \ell_1 \cdots\cdots \xrightarrow{a_{n-1}} \ell_n \cdots$$

where for each $i \geq 0$, $(\ell_i, a_i, \ell_{i+1}) \in E$. Definitions of traces and languages are straightforward. 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, $\mathit{Dur}(\varrho) = n$ and otherwise $\mathit{Dur}(\varrho) = \infty$.

D. Region Graph of a TA

The *region graph* $\mathit{RG}(A)$ of a TA A is a finite quotient of the infinite graph of A which is time-abstract bisimilar to A [7]. It is a FA on the alphabet $E' = E \cup \{\tau\}$. The states of $\mathit{RG}(A)$ 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 [7] is:

Theorem 1 ([7]): $\mathcal{L}(RG(A)) = Unt(\mathcal{L}(A))$.

In other words:

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

The (maximum) size of the region graph is exponential in the number of clocks and in the maximum constant of the automaton A (see [7]): $|RG(A)| = |L| \cdot |X|! \cdot 2^{|X|} \cdot K^{|X|}$ where K is the largest constant used in A.

E. Product of TA

Definition 2 (Product of TA): Let $A_i = (L_i, l_0^i, X_i, \Sigma_\tau^i, E_i, Inv_i)$, $i \in \{1,2\}$, be TA s.t. $X_1 \cap X_2 = \varnothing$. The *product* of A_1 and A_2 is the TA $A_1 \times A_2 = (L, l_0, X, \Sigma_\tau, E, Inv)$ 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 $E \subseteq L \times \mathcal{C}(X) \times \Sigma_\tau \times 2^X \times L$ 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 $Inv(\ell_1,\ell_2) = Inv(\ell_1) \wedge Inv(\ell_2)$. ■

The definition of product also applies to finite automata.

III. Fault Diagnosis Problems

The material in this section is based on [6], [8], [10]. 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 [6]. 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 as a TA $A = (L, l_0, X, \Sigma_{\tau,f}, E, Inv)$. Fig. 1 gives an example of such a system ($\alpha \in \mathbb{N}$ is a 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 of A

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

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

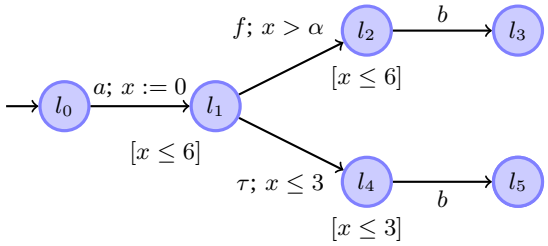

Figure 1. The Timed Automaton $\mathcal{A}(\alpha)$

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

$$Faulty^{tr}_{\geq\Delta}(A) = Tr(Faulty_{\geq\Delta}(A))$$

and

$$NonFaulty^{tr}(A) = Tr(NonFaulty(A))$$

which are the traces[3] of Δ-faulty and non-faulty runs of A.

The purpose of fault diagnosis is to detect a fault as soon as possible. Faults 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 $\pi_{/\Sigma}(w)$. 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 3 (Δ-Diagnoser): Let A be a TA over the alphabet $\Sigma_{\tau,f}$ and $\Delta \in \mathbb{N}$. A Δ-*diagnoser* for A is a mapping $D : TW^*(\Sigma) \to \{0,1\}$ such that:

- for each $\varrho \in NonFaulty(A)$, $D(tr(\varrho)) = 0$,
- for each $\varrho \in Faulty_{\geq\Delta}(A)$, $D(tr(\varrho)) = 1$. ■

A is Δ-diagnosable if there exists a Δ-diagnoser for A. A is diagnosable if there is some $\Delta \in \mathbb{N}$ s.t. A est Δ-diagnosable.

Remark 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 1: The TA $\mathcal{A}(3)$ in Fig. 1 taken from [8] 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 2 time units. If $\alpha = 2$, in $\mathcal{A}(2)$ there are two runs:

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

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

that satisfy $tr(\rho_1(\delta)) = tr(\rho_2(\delta))$, 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.

The classical fault diagnosis problems are the following:

Problem 1 (Bounded or Δ-Diagnosability):
INPUTS: A TA $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, Inv)$ and $\Delta \in \mathbb{N}$.
PROBLEM: Is A Δ-diagnosable?

Problem 2 (Diagnosability):
INPUTS: A TA $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, Inv)$.
PROBLEM: Is A diagnosable?

Problem 3 (Maximum delay):
INPUTS: A TA $A = (L, \ell_0, X, \Sigma_{\tau,f}, E, Inv)$.
PROBLEM: If A is diagnosable, what is the minimum Δ s.t. A is Δ-diagnosable ?

We do not address here the problem of synthesizing a diagnoser and the reader is referred to [6], [5], [8], [9] for a detailed presentation.

A necessary and sufficient condition for diagnosability was already established in [3], but was stated on a candidate diagnoser. We give here a simple language based condition, valid in both the discrete and timed cases. According to Definition 3, A is diagnosable, iff, there is some $\Delta \in \mathbb{N}$ s.t. A is Δ-diagnosable. Thus:

$$A \textbf{ is not} \text{ diagnosable} \Longleftrightarrow \forall \Delta \in \mathbb{N}, A \text{ is not } \Delta\text{-diagnosable.}$$

Moreover a trace based definition of Δ-diagnosability can be stated as: A is Δ-diagnosable iff

$$Faulty^{tr}_{\geq \Delta}(A) \cap NonFaulty^{tr}(A) = \varnothing. \tag{1}$$

This gives a necessary and sufficient condition for non-diagnosability and thus diagnosability:

$$A \textbf{ is not} \text{ diagnosable} \Longleftrightarrow \begin{cases} \forall \Delta \in \mathbb{N}, \\ \quad \exists \rho \in NonFaulty(A) \\ \quad \exists \rho' \in Faulty_{\geq \Delta}(A) \text{ s.t.} \\ \quad\quad tr(\rho) = tr(\rho'), \end{cases} \tag{2}$$

or in other words, there is no pair of runs (ρ_1, ρ_2) with $\rho_1 \in Faulty_{\geq \Delta}(A)$, $\rho_2 \in NonFaulty(A)$ the traces of which are equal.

IV. Algorithms for Discrete Event Systems

In this section we briefly review the main results about diagnosability of discrete-event systems. We consider here that the DES is given by a FA $A = (Q, q_0, \Sigma_{\tau,f}, \rightarrow)$.

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 (of some lemmas in [10]) 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.

A. Problem 1

To check Problem 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.

Let $A_1 = (Q \times \{-1, 0, \cdots, \Delta+1\}, (q_0, -1), \Sigma_\tau, \rightarrow_1)$ 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 2: A is Δ-diagnosable iff $\mathcal{L}^*(\mathcal{B}) = \varnothing$.

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 $(\Delta+1) \cdot |A|^2$ which is exponential in the size of the inputs of the problem A and Δ because Δ is given in binary. Thus Problem 1 can be solved in EXPTIME. As storing Δ requires only polynomial space Problem 1 is in PSPACE. Actually checking Problem 1 can be done in PTIME (see the end of this section).

B. Problem 2

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. We let $\rightarrow_{1,2}$ be the transition relation of $A_1 \times A_2$. A faulty run of $A_1 \times A_2$ is a run for which A_1 reaches a faulty state of the form $(q, 1)$. To decide whether A is diagnosable we build an extended version of $A_1 \times A_2$ which is a Büchi automaton $\mathcal{B}$ as follows: $\mathcal{B}$ has a boolean variable z which records whether A_1 participated in the last transition fired by $A_1 \times A_2$. Assume we have a predicate[4] $A_1 Move(t)$ which is true when A_1 participates in a transition t of the product $A_1 \times A_2$. A state of $\mathcal{B}$ is a pair (s, z) where s is a state of $A_1 \times A_2$. $\mathcal{B}$ 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:

- $(s, z) \xrightarrow{\sigma}_\mathcal{B} (s', z')$ if (i) there exists a transition $t : s \xrightarrow{\sigma}_{1,2} s'$ in $A_1 \times A_2$, and (ii) $z' = 1$ if $A_1 Move(t)$ and $z' = 0$ otherwise;
- $R_\mathcal{B} = \{(((q, 1), q'), 1) \mid ((q, 1), q') \in A_1 \times A_2\}$.

$\mathcal{B}$ accepts the language $\mathcal{L}(\mathcal{B}) = \mathcal{L}^\omega(\mathcal{B}) \subseteq \Sigma^\omega$. Moreover this language satisfies a nice property:

Theorem 3 ([10]): A is diagnosable iff $\mathcal{L}^\omega(\mathcal{B}) = \varnothing$.

This theorem has for consequence that the diagnosability problem can be checked in quadratic time: the automaton $\mathcal{B}$

[4]This is easy to define when building $A_1 \times A_2$.

has size $4 \cdot |A|^2$ i.e., $O(|A|^2)$ and checking emptiness for Büchi automaton can be done in linear time. Thus diagnosability can be checked in PTIME. Polynomial algorithms for checking diagnosability (Problem 2) were already reported in [5], [6]. 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 [5], [6], 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 is a minor difference complexity-wise, but in practice this can be useful to reduce the size of the product.

Moreover, reducing Problem 2 to emptiness checking of Büchi automata is interesting in many respects:

- the proof (see [10]) of Theorem 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 [11]). Numerous algorithms, like *on-the-fly* algorithms [12] have been designed to improve memory/time consumption (see [13] 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[5] like DESUMA/UMDES [14] (notice that the comparison with DESUMA/UMDES concerns only the diagnosability algorithms; DESUMA/UMDES can perform a lot more than checking diagnosability).

From Theorem 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 2 reduces to a particular instance of Problem 1 which was already stated in [6]:

Theorem 4 ([6]): A is diagnosable if and only if A is $(2 \cdot |Q|^2)$-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 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 2 as a Büchi emptiness problem can be done in $O(|A|^2)$. Thus the extra-cost of using a reachability algorithm is still reasonable.

The Büchi-emptiness algorithm used to solve Problem 2 can also be used to solve Problem 1 for a given Δ and automaton A with set of states Q: if $\Delta \geq 2 \cdot |Q|^2$, then we check wether A is diagnosable and this gives the answer to Problem 1; otherwise, if $\Delta < 2 \cdot |Q|^2$, we check wether A is Δ-diagnosable but in polynomial time. Hence Problem 1 can be solved in polynomial time $O(|A|^4)$.

Finally, solving Problem 3 can be done by a binary search solving iteratively Δ-diagnosability problems starting with $\Delta = 2 \cdot |A|^2$. Thus Problem 3 can be solved in $O(|A|^4)$. Using a different approach, Problem 3 was reported to be solvable in $O(|Q|^3)$ in [15].

In the sequel we recall the algorithm for checking diagnosability for TA and establish a counterpart of Theorem 4 for TA.

V. ALGORITHMS FOR TIMED AUTOMATA

We first recall how to check Δ-diagnosability for TA which first appeared in [8].

A. Problem 1

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

- $((\ell, n), g, \lambda, r, (\ell', n)) \in E_1$ 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 $(\ell, g, f, r, \ell') \in E$;
- $\mathit{Inv}_1((\ell, n)) = \mathit{Inv}(\ell)$;
- 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 the clocks x in X are replaced by their counterpart x_2; r_2 is r with the same renaming;
- $\mathit{Inv}_2(\ell) = \mathit{Inv}(\ell)$.

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. If this set is not empty, 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 5 ([8]): A is Δ-diagnosable iff $\mathcal{L}^*(\mathcal{D}) = \varnothing$.

Checking reachability of a location for TA is PSPACE-complete [7]. 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. Hence:

Corollary 1: Problem 1 can be solved in PSPACE for TA.

B. Problem 2

As for the untimed case, we build an automaton $\mathcal{D}$, which is 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 [8]:

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

- some runs of A_2 might prevent time from elapsing from a given point in time. In this case, equation (1) cannot be satisfied but this is for an artificial 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[6] 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 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 Fig. 2. If we use $F = \varnothing$ and $R = \{1\}$

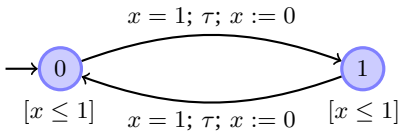

Figure 2. Timed Automaton *Div*(x)

for *Div*(x), any accepted run is time divergent. Let $\mathcal{D} = (A_1 \times A_2) \times$ *Div*(x) with $F_{\mathcal{D}} = \varnothing$ and $R_{\mathcal{D}}$ is the set of states where A_1 is in a faulty state and *Div*(x) is location 1. The following theorem is the TA counterpart of Theorem 3:

Theorem 6 ([8]): A is diagnosable iff $\mathcal{L}^{\omega}(\mathcal{D}) = \varnothing$.

Deciding whether $\mathcal{L}^{\omega}(A) \neq \varnothing$ for TA is PSPACE-complete [7]. Thus deciding diagnosability is in PSPACE.

The reachability problem for TA can be reduced to a diagnosability problem [8]. 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: $(End, \text{TRUE}, \lambda, \varnothing, End)$ for $\lambda \in \{\tau, f\}$. Then: A' is not diagnosable iff *End* is reachable in A. It follows that:

Theorem 7 ([8]): Problem 2 is PSPACE-complete for TA.

We can draw another conclusion from the previous theorem: if a TA A is diagnosable, there cannot be any cycle with faulty states in the region graph of $A_1 \times A_2 \times$ *Div*(x). Indeed, otherwise, by Theorem 1, there would be a non-Zeno word in $A_1 \times A_2 \times$ *Div*(x) itself[7]. Let $\alpha(A)$ denote the size of the region graph *RG*$(A_1 \times A_2 \times$ *Div*$(x))$. If A is diagnosable, then (P_1): a faulty state in *RG*$(A_1 \times A_2 \times$ *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 $A_1 \times A_2 \times$ *Div*(x). Hence (P_2): all the regions following a faulty state in *RG*$(A_1 \times A_2 \times$ *Div*$(x))$ are bounded. As the amount of time which can elapse within a region is less than 1 time unit[8], this implies that the duration of the longest faulty run in $A_1 \times A_2 \times$ *Div*(x) is less than $\alpha(A)$. Actually as every other region is a *singular region*[9], it must be less than $(\alpha(A)/2) + 1$. Thus we obtain the following result:

Theorem 8: A is diagnosable if and only if A is $(\alpha(A)/2) + 1$-diagnosable.

As diagnosability can be reduced to Δ-diagnosability for TA:

Corollary 2: Problem 1 is PSPACE-complete for TA.

Problem 3 can be solved by a binary search and is also in PSPACE for TA. Although Problem 1 and Problem 2 are PSPACE-complete for timed automata, the price to pay to solve Problem 2 as a reachability problem is much higher than solving it as a Büchi emptiness problem: indeed the size of the region graph of $A_1(\alpha(A)) \times A_2$ is the square of the size of the region graph of $A_1 \times A_2 \times$ *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.

VI. Conclusion

The main conclusions we can draw from the previous presentation are two-fold.

From a theoretical viewpoint, it shows that the fault diagnosis algorithms for DES and for TA are essentially the same: in both cases, diagnosability can be reduced to Büchi emptiness; and also to bounded diagnosability. The interesting point is that the complexity of the algorithms are the same for DES and TA except that for timed automata, the complexity measure is space (Table I).

TABLE I
Summary of the Results

	Δ-Diagnosability	Diagnosability	
	Reach Algorithm	*Büchi Algorithm*	*Reach Algorithm*
DES	PTIME $O(\|A\|^4)$	PTIME $O(\|A\|^2)$	PTIME $O(\|A\|^4)$
TA	PSPACE-C.	PSPACE-C. $O(\|A\|^2)$	PSPACE-C. $O(\|A\|^4)$

From a practical viewpoint, it clearly shows that the model-checking algorithms and tools developed in the model-checking/verification community can be used to solve

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

[7] Note that this is true because we add the automaton *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.

[8] We assume the constants are integers.

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

the diagnosability problems; these tools usually have a very expressive specification language (e.g., Promela/Spin [16], UPPAAL [17] or KRONOS [18]) and very efficient data structures/implementations (e.g., [13] or [19]).

We can also use the results in Table I to guide our choice of algorithms for checking diagnosability. Let *Reach* denote the reachability algorithm for checking Δ-diagnosability and *Buchi* denote the Büchi emptiness algorithm for checking diagnosability:

- time-wise, solving the diagnosability problem for a finite automaton using *Reach* is a bit more expensive than using *Buchi*, but the difference is not drastic;
- for a timed automaton A it is totally different: space-wise the amount of space required by *Reach* is the square of the amount of space required by *Buchi*. Time-wise this means a worst case blow up from $2^{|A|^2}$ to $2^{|A|^4}$. It is thus clear that one should use the Büchi emptiness algorithm in this case. Checking Büchi emptiness for TA is efficiently implemented in a version of KRONOS (Profounder) [20] and in UPPAAL-TiGA [21], the game version of UPPAAL [22].

The previous results show that model-checking tools (both for finite and timed automata) are suitable to solve the diagnosis problems, and provide expressive specification languages and efficient algorithms and tools.

REFERENCES

[1] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal of Control and Optimization*, vol. 25, no. 1, pp. 1202–1218, 1987.

[2] ——, "The control of discrete event systems," *Proc. of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[3] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 40, no. 9, Sept. 1995.

[4] ——, "Failure diagnosis using discrete-event models," *IEEE Transactions on Control Systems technology*, vol. 4, no. 2, Mar. 1996.

[5] S. Jiang, Z. Huang, V. Chandra, and R. Kumar, "A polynomial algorithm for testing diagnosability of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 46, no. 8, Aug. 2001.

[6] T.-S. Yoo and S. Lafortune, "Polynomial-time verification of diagnosability of partially-observed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 47, no. 9, pp. 1491–1495, Sept. 2002.

[7] R. Alur and D. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1994.

[8] S. Tripakis, "Fault diagnosis for timed automata," in *Proceedings of the International Conference on Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT'02)*, ser. LNCS, W. Damm and E.-R. Olderog, Eds., vol. 2469. Springer Verlag, 2002, pp. 205–224.

[9] P. Bouyer, F. Chevalier, and D. D'Souza, "Fault diagnosis using timed automata," in *Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, ser. LNCS, V. Sassone, Ed., vol. 3441. Edinburgh, U.K.: Springer Verlag, Apr. 2005, pp. 219–233. [Online]. Available: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/fossacs05-BCD.pdf

[10] F. Cassez and S. Tripakis, "Fault diagnosis with static or dynamic diagnosers," *Fundamenta Informaticae*, vol. 88, no. 4, pp. 497–540, Nov. 2008.

[11] G. J. Holzmann, "Software model checking with spin," *Advances in Computers*, vol. 65, pp. 78–109, 2005.

[12] J.-M. Couvreur, A. Duret-Lutz, and D. Poitrenaud, "On-the-fly emptiness checks for generalized büchi automata," in *SPIN*, ser. LNCS, P. Godefroid, Ed., vol. 3639. Springer Verlag, 2005, pp. 169–184.

[13] S. Schwoon and J. Esparza, "A note on on-the-fly verification algorithms," in *TACAS*, ser. LNCS, N. Halbwachs and L. D. Zuck, Eds., vol. 3440. Springer Verlag, 2005, pp. 174–190.

[14] L. Ricker, S. Lafortune, and S. Genc, "Desuma: A tool integrating giddes and umdes," in *Proc. of the* 8^{th} *Workshop on Discrete Event Systems (WODES'08)*. Ann Arbor, MI, USA: IEEE Computer Society, July 2006.

[15] T.-S. Yoo and H. Garcia, "Computation of fault detection delay in discrete-event systems," in *Proceedings of the 14th International Workshop on Principles of Diagnosis, DX'03*, Washington, D.C., USA, June 2003, pp. 207–212.

[16] G. J. Holzmann, "Software model checking with spin," *Advances in Computers*, vol. 65, pp. 78–109, 2005.

[17] G. Behrmann, A. David, and K. G. Larsen, "A tutorial on UPPAAL," in *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*, ser. LNCS, M. Bernardo and F. Corradini, Eds., vol. 3185. Springer Verlag, September 2004, pp. 200–236.

[18] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine, "Kronos: A model-checking tool for real-time systems," in *CAV*, ser. LNCS, A. J. Hu and M. Y. Vardi, Eds., vol. 1427. Springer Verlag, 1998, pp. 546–550.

[19] G. Behrmann, J. Bengtsson, A. David, K. G. Larsen, P. Pettersson, and W. Yi., "UPPAAL implementation secrets," in *Proc. of 7*th *International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*, 2002.

[20] S. Tripakis, "Checking timed buchi automata emptiness on simulation graphs," *ACM Transactions on Computational Logic*, 2009, forthcoming.

[21] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, and D. Lime, "Uppaal-tiga: Time for playing games!" in *CAV*, ser. LNCS, W. Damm and H. Hermanns, Eds., vol. 4590. Springer Verlag, 2007, pp. 121–125.

[22] A. David, K. G. Larsen, and D. Lime, "UPPAAL-TiGA 2009 – Towards Realizable Strategies," 2009, private Communication.