Active Diagnosis with Observable Quiescence

Stanislav Böhm*, Stefan Haar†‡, Serge Haddad†‡, Piotr Hofman† and Stefan Schwoon†‡
* IT4 Innovations, National Supercomputing Center, Ostrava, Czech Republic
† LSV, ENS Cachan & CNRS, 61 avenue Président Wilson, 94230 Cachan, France
‡ Inria

***Abstract*— Active diagnosis of a discrete-event system consists in controlling the system such that faults can be detected. Here we extend the framework of active diagnosis by introducing modalities for actions and states and a new capability for the controller, namely observing that the system is quiescent. We design a game-based construction for both the decision and the synthesis problems that is computationally optimal. Furthermore we prove that the size and the delay provided by the active diagnoser (when it exists) are almost optimal.**

I. Introduction

Diagnosis and control are important tasks in managing discrete-event systems (DES). In this paper, we contribute to the study of *active diagnosis*, which combines the two aspects: In a system whose events are partially observable, a controller observing an ongoing execution is charged with diagnosing whether a certain event, usually called *fault* and not directly observable, has happened or not. To this end, the controller may intervene and restrict the behaviour of the system in precisely defined ways. The active-diagnosis problem is to determine whether it is possible to control the system in such a way that diagnosis is always possible and, if so, synthesize a corresponding controller.

The active-diagnosis problem for DES was first studied in [1]. More recently, [2] proposed a new construction for active diagnosers based on automata and game theory that is provably optimal w.r.t. the size of the computed controller and the computational complexity for building it. Moreover, a number of variations have been studied. They range from the selection of minimal sets of observable labels that make the system diagnosable [3], to online aspects that either turn on and off sensors [3], [4] or modify an action plan [5] in order to reduce the amount of ambiguity, and to the case of probabilistic systems [6]. See also [7] for an extensive survey of DES problems.

In this work, we extend the frameworks of [1], [2]. Most importantly, we consider the case where the controller is able to observe that the system is quiescent and to exploit such observations for active diagnosis. While we believe that such an extension is natural and worth studying, it has a suprisingly large effect on the formal framework for active diagnosis. Notably, one of the tasks of the controller is, given the stream of observations σ, to decide whether σ indicates that a fault has happened. In [1], [2], this decision depends on σ alone, whereas in our framework it also depends on the control exercised during the execution that produced σ.

S. Böhm was supported by the project New Creative Teams in Priorities of Scientific Research (CZ.1.07/2.3.00/30.0055), supported by Operational Programme Education for Competitiveness and co-financed by the European Social Fund and the state budget of the Czech Republic. The work was also supported by the European Regional Development Fund in the IT4Innovations Center of Excellence project (CZ.1.05/1.1.00/02.0070).
P. Hofman was supported by Labex Digicosme, Univ. Paris-Saclay, project VERICONISS.

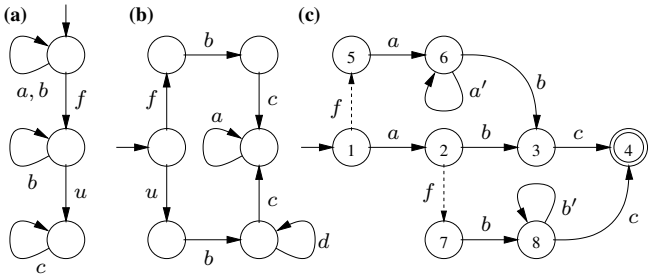

Fig. 1. Examples of LTS. In (c), lazy events are indicated by dashed arrows and idle states by a double boundary.

We will present details of the model together with motivating examples in Section II. Section III presents our framework for active diagnosis, while in Section IV we show how to solve the active-diagnosis problem with the aforementioned additions, and Section V establishes worst-case lower bounds for certain aspects. For lack of space, some proofs have been omitted from this paper. They are published in [8], together with a few supplementary results.

II. The Model

Before giving a formal definition of our framework, we discuss some motivating examples for active diagnosis and our extensions. As in [1], [2], we consider systems whose events can be *observable* or *unobservable*; a controller observing an ongoing execution of the system will only see its observable events. Consider Figure 1 (a). As in the examples that follow, all events except f and u are observable, where f represents the fault. In a purely passive diagnosis setting, that system is considered *undiagnosable*: an observer seeing only a stream of bs cannot decide whether f has happened or not. However, suppose that event b is *controllable*, meaning that the observer can enable or disable it. Then, after seeing a number of bs, the observer may temporarily disable b, and the next observable action of the system must be a or c, revealing whether the fault has happened or not.

The first of our extensions w.r.t. [1], [2] concerns the ability of the diagnoser to deal with temporary quiescence

of the system. The concept of quiescence is well-established and used in *conformance testing*, see e.g. Tretmans [9]–[11], where it is used to observe that the system under test will not produce any output unless provided with additional input.

We shall construct controllers with the capacity to observe that the system is quiescent and react by re-enabling some events. Quiescence can be exploited in active diagnosis, as the example in Figure 1 (b) shows, where only c is controllable. That system is considered non-actively-diagnosable in the frameworks of [1], [2], which are language-based: the upper half of the system can only do (a prefix of) fbc, which is undistinguishable from a sequence possible in the lower half; yet, a controller cannot prevent the system from entering the upper half with uncontrollable actions fb. Suppose now another approach in which a controller blocks c after seeing b and then has the capacity to observe that either d happens (i.e. the system is in q') or that the system is quiescent (i.e. the current state is q). The controller can then issue the corresponding diagnosis verdict and, e.g., re-enable c to let the system continue.

Quiescence is semantically different from extending the system with some event δ symbolising "passage of time". While, e.g., Figure 1 (b) can be made actively diagnosable in the sense of [1], [2] by adding a δ-loop to q, our method has at least two advantages: First, it does not require the designer of a system to add artefacts that are merely required by deficiencies of some analysis method. Secondly, such a transformation is not always possible. In fact, the long version of this paper [8] shows an example that is actively diagnosable in our framework but not in [1], [2], and no addition of δ-loops can render it so, even when δ is considered controllable. Treating quiescence directly thus genuinely extends the power of active diagnosers.

Moreover, quiescence allows us to meaningfully distinguish two types of events that we call *eager* and *lazy*. An eager event enabled in some state eventually happens unless another event pre-empts it; typically, eager events are those happening during the normal course of the system. By contrast, a *lazy* event represents an entirely non-deterministic action, e.g. a fault or input from the environment. Thus, quiescence can be observed in a state even when one or more lazy actions are enabled.

Figure 1 (c) represents a three-stage production process with actions a, b, c, all controllable. The fault f is lazy, indicated by dashed arcs. If all goes well, the system proceeds from state 1 to state 4 with abc. However, steps a and b can be erroneous, which must be tested for before proceeding with b resp. c. E.g., after seeing a, the system could be in states 2 or 6, and the controller will temporarily disable b to see whether it observes a' or quiescence. However, if the system is in state 2, this temporary disabling does not necessarily provoke the lazy event f. In our framework, the system can reach state 4 without fault while enabling a controlling observer to verify that the execution was fault-free. By contrast, [1], [2] consider all events to be eager; for them, the system is actively diagnosable but at the price of eliminating all non-faulty executions.

Finally, notice that state 4 in Figure 1 (c) has a double boundary. We use this to indicate so-called *idle* states, and we forbid controllers to observe quiescence forever unless the system is in such an idle state. In the example, this allows the designer of the system to formulate the requirement that an active diagnoser either detect a fault or allow the production to run to completion. This concept generalizes the liveness requirement made in [2], where all runs must be infinite. We remark that a similar concept called *marked* states is discussed for controllability problems in [7]. However, in our case quiescence may be a temporary state of the system from which it may re-awaken, e.g. following a lazy event triggered by the environment.

III. Definitions

This section serves to provide formal notations related to discrete-event systems and active diagnosis. As mentioned in Section II, we shall study discrete-event systems whose events, here called *actions*, have additional properties, such as being observable, controllable, or eager, and we will successively introduce the notions related to these concepts. Given a set X, we use the standard notations X^* (resp. X^+, X^ω) to denote the finite (resp. non empty finite, infinite) sequences over X; the empty sequence is denoted ε, and the length of a finite sequence σ by $|\sigma|$. Given a sequence $\sigma = a_1 \cdots a_n \in X^+$, $last(\sigma)$ denotes a_n. Given a sequence $\sigma = a_1 \cdots \in X^* \cup X^\omega$ and $1 \leq i \leq j \leq |\sigma|$, $\sigma[i]$ denotes a_i and $\sigma[i,j]$ denotes $a_i \cdots a_j$.

Labeled transition systems

We consider an extension of labeled transition systems:

Definition 1: A *labeled transition system* (LTS) is a tuple $\mathcal{A} = \langle Q, q_0, \Sigma, T \rangle$ where Q is a set of states with initial state $q_0 \in Q$, Σ is a finite set of actions, and $T \subseteq Q \times \Sigma \times Q$ is the set of transitions. $\mathcal{A}$ is *deterministic* if for every pair $q \in Q, a \in \Sigma$ there is at most one q' such that $\langle q, a, q' \rangle \in T$, and *complete* if there is at least one.

If $\langle q, a, q' \rangle \in T$, we write $q \xrightarrow{a} q'$ and say that a is *enabled* in q; for a deterministic automaton we also write $T(q, a) = q'$. The set of enabled actions in state q is denoted $en(q)$. An infinite *run* over the word $\sigma = a_1 a_2 \ldots \in \Sigma^\omega$ is an alternating sequence of states and actions $(q_i a_{i+1})_{i \geq 0}$ such that $q_i \xrightarrow{a_{i+1}} q_{i+1}$ for all $i \geq 0$, and we write $q_0 \overset{\sigma}{\Longrightarrow}$ if such a run exists. A finite run over $w \in \Sigma^*$ is defined analogously, and we write $q \overset{w}{\Longrightarrow} q'$ if such a run ends at state q'. A state q is *reachable* if there exists a run $q_0 \overset{w}{\Longrightarrow} q$ for some w.

Definition 2 (languages): Let $\mathcal{A} = \langle Q, q_0, \Sigma, T \rangle$ be an LTS. The finite and infinite language of $\mathcal{A}$ are defined by $\mathcal{L}^*(\mathcal{A}) = \{ w \in \Sigma^* \mid \exists q : q_0 \overset{w}{\Longrightarrow} q \}$ and $\mathcal{L}^\omega(\mathcal{A}) = \{ \sigma \in \Sigma^\omega \mid q_0 \overset{\sigma}{\Longrightarrow} \}$.

Partially observable controllable systems

We now define partially observable controllable systems (POCS) on which we shall perform active diagnosis. Syntactically, a POCS $\mathcal{S}$ is an LTS $\mathcal{A}$ enlarged with three binary

partitions: an action may be (1) *observable* (in Σ_o) or *unobservable* (in Σ_{uo}), (2) *controllable* (in Σ_c) or *uncontrollable* (in Σ_{uc}), and (3) *eager* (in Σ_e) or *lazy* (in Σ_ℓ). We require that unobservable actions are uncontrollable. Below, we shall define the semantics of a POCS under a controller $cont$ as a new LTS $\mathcal{S}_{cont}$. Intuitively, during an execution of system $\mathcal{A}$, a controller may forbid a subset of the controllable actions based on the observable actions seen so far, thereby restricting the behaviour of $\mathcal{A}$. This implies that $\mathcal{A}$ must be *convergent*, i.e. there is no infinite run with a suffix of unobservable actions: $\mathcal{L}^\omega(\mathcal{A}) \cap \Sigma^* \Sigma_{uo}^\omega = \emptyset$. Occasionally, the control may lead to a situation in which the system reaches a state where all non-blocked actions are lazy. In this situation, either one such action occurs or the controller sees that "nothing is happening", represented by a special observation symbol δ. We say that the system is in a *quiescent* state (not to be confused with a deadlock state). The controller may then once again change the set of allowed actions. The set $Idle$ indicates the states in which the system may legitimately remain quiescent forever; in particular, a correct controller should not block all eager actions indefinitely when $\mathcal{A}$ is in a state $q \notin Idle$. The next two definitions formalize POCS and their semantics.

Definition 3: A *partially observable controllable system* (POCS) is a tuple $\mathcal{S} = \langle \mathcal{A}, \Sigma_o, \Sigma_c, \Sigma_e, f, Idle \rangle$, where $\mathcal{A} = \langle Q, q_0, \Sigma, T \rangle$ is a convergent LTS with $\Sigma_o, \Sigma_e \subseteq \Sigma$ such that $Idle \subseteq Q$, $\Sigma_c \subseteq \Sigma_o$, and $f \in \Sigma \setminus \Sigma_o$ is a distinguished action called *fault*.

As previously discussed, we write $\Sigma_{uo}, \Sigma_{uc}, \Sigma_\ell$ for the complements of $\Sigma_o, \Sigma_c, \Sigma_e$. Let us denote $\Xi := \Sigma \cup \{\delta\}$ and $\Xi_o := \Sigma_o \cup \{\delta\}$ the set of (observable) actions extended with δ, and let $\sigma \in \Xi^*$. The projection $\mathcal{P}(\sigma)$ erases all letters not from Ξ_o, more precisely $\mathcal{P}(\varepsilon) = \varepsilon$, and $\mathcal{P}(\sigma a)$ equals $\mathcal{P}(\sigma)a$ if $a \in \Xi_o$ and $\mathcal{P}(\sigma)$ otherwise. For $\sigma \in \Xi^\omega$, its projection is the limit of the projections of its finite prefixes. When using the projection to another subset X, we write $\mathcal{P}_X$.

A *controller* for $\mathcal{S}$ is a mapping $cont : \Xi_o^* \to 2^{\Sigma_c}$. The behaviour of an LTS $\mathcal{S}$ under control of $cont$ is defined by another LTS $\mathcal{S}_{cont}$ whose states are pairs where the first component is the sequence of observations associated with the current execution and the second component is the current state of S. The transitions are those allowed by the controller w.r.t. the observed sequence with an update of the observed sequence in case of an observable event.

Definition 4 (Controlled system): Let $\mathcal{S}$ be a POCS and $cont$ a controller. Then $\mathcal{S}_{cont} := \langle Q_{cont}, q_{0cont}, \Xi, T_{cont} \rangle$ is defined as the smallest LTS satisfying:

(i) $q_{0cont} := \langle \varepsilon, q_0 \rangle \in Q_{cont}$;
(ii) if $\langle \sigma, q \rangle \in Q_{cont}$, $a \in cont(\sigma) \cup \Sigma_{uc}$, and $q \xrightarrow{a} q'$, then $\langle \sigma\mathcal{P}(a), q' \rangle \in Q_{cont}$ and $\langle \langle \sigma, q \rangle, a, \langle \sigma\mathcal{P}(a), q' \rangle \rangle \in T_{cont}$;
(iii) if $\langle \sigma, q \rangle \in Q_{cont}$ and $cont(\sigma) \cap en(q) \subseteq \Sigma_\ell$ then $\langle \sigma\delta, q \rangle \in Q_{cont}$ and $\langle \langle \sigma, q \rangle, \delta, \langle \sigma\delta, q \rangle \rangle \in T_{cont}$.

An *observable sequence* is an item of Ξ_o^*. An *observed sequence* is an item of $\Lambda(cont) := \{ \sigma \mid \langle \sigma, q \rangle \in Q_{cont} \}$.

Ambiguity

A finite or infinite word σ over Σ (resp. Ξ) is *faulty* if it contains an occurrence of f; otherwise it is called *correct*. Given an observed sequence σ, the aim of diagnosis is to determine whether a fault has surely occurred. The ambiguous sequences are exactly the observed sequences where diagnosis is not yet possible.

Definition 5 (ambiguous and surely faulty sequence): Let $\mathcal{S}_{cont}$ be a controlled system, $\sigma_1, \sigma_2 \in \mathcal{L}^\omega(\mathcal{S}_{cont})$ be two sequences and $\sigma \in \Xi_o^\omega$ such that: (i) $\mathcal{P}(\sigma_1) = \mathcal{P}(\sigma_2) = \sigma$, (ii) σ_1 is correct, and (iii) σ_2 is faulty. Then σ is called *ambiguous* in $\mathcal{S}_{cont}$, and the pair $\langle \sigma_1, \sigma_2 \rangle$ is a *witness* for the ambiguity of σ. Ambiguous finite sequences are defined analogously. A sequence $\sigma' \in \Xi_o^*$ is *surely faulty* in $\mathcal{S}_{cont}$ for all $\sigma \in \mathcal{L}^*(\mathcal{S}_{cont})$ such that $\mathcal{P}(\sigma) = \sigma'$, σ is faulty.

Active diagnosability

In the diagnosis framework, the goal of the controller is to make the system diagnosable, and to perform diagnosis. Thus, an *active diagnoser* is a controller equipped with a diagnosis function. The active dignoser must (1) eliminate ambiguity, (2) detect fault and (3) does not leave the system stuck forever in a non-idle quiescent state.

Definition 6 (Active Diagnoser): Let $\mathcal{S}$ be a POCS and $h = \langle cont, diag \rangle$, where $cont$ is a controller and $diag$ a mapping from $\Lambda(cont)$ to $\{\bot, \top\}$. We call h *active diagnoser* for $\mathcal{S}$ iff:

1) $\mathcal{S}_{cont}$ does not contain any infinite ambiguous sequence;
2) $diag(\sigma) = \top$ if and only if σ is surely faulty in $\mathcal{S}_{cont}$;
3) For all infinite runs $(s_i a_{i+1})_{i \geq 0}$ of $\mathcal{S}_{cont}$, if there exists i_0 with $a_i = \delta$ for all $i \geq i_0$ then $s_{i_0} = \langle \sigma', q \rangle$ for some σ' and $q \in Idle$.

For $k \geq 1$, h is called a *k-active diagnoser* if for all $\sigma = \sigma' f \sigma'' \in \mathcal{L}^*(\mathcal{S}_{cont})$ with $|\mathcal{P}(\sigma'')| \geq k$, $diag(\mathcal{P}(\sigma)) = \top$, i.e. every fault is diagnosed after at most k observations. The minimal k s.t. h is a k-active diagnoser is called the *delay* of h. We call $\mathcal{S}$ *(k-)actively diagnosable* if a (k-)active diagnoser exists, and the minimal such k the *index* of $\mathcal{S}$.

An active diagnoser does not necessarily have a finite delay [2]. However, we will see that if $\mathcal{S}$ is actively diagnosable, there does exist a k-active diagnoser for some k.

We are now in a position to formally state the relevant problems for active diagnosis. Let $\mathcal{S}$ be a POCS with finitely many states. We are interested in:

- the *active diagnosis decision problem*, i.e. decide whether $\mathcal{S}$ is actively diagnosable;
- the *synthesis problem*, i.e. build an active diagnoser (if $\mathcal{S}$ is actively diagnosable);
- the *minimal-delay synthesis problem*, i.e. decide whether $\mathcal{S}$ is actively diagnosable and in the positive case build an active diagnoser with minimal delay.

We introduce the notion of *pilot* as a finite representation of an active diagnoser.

Definition 7 (pilot): Let $\mathcal{S}$ be a POCS. Then $\mathcal{C} = \langle \mathcal{B}_\mathcal{C}, cont_\mathcal{C}, diag_\mathcal{C} \rangle$ is called *pilot* for $\mathcal{S}$ if $\mathcal{B}_\mathcal{C} = \langle Q^c, q_0^c, \Xi_o, T^c \rangle$ is a deterministic complete LTS,

$\langle cont_{\mathcal{C}}, diag_{\mathcal{C}}\rangle : Q^c \to 2^{\Sigma_c} \times \{\bot, \top\}$, are labellings. Let $h_{\mathcal{C}} = \langle cont, diag\rangle$ associated with $\mathcal{C}$ be defined by $cont(\sigma) = cont_{\mathcal{C}}(q)$ and $diag(\sigma) = diag_{\mathcal{C}}(q)$ for all $\sigma \in \Lambda(cont)$, where q is the unique state such that $q_0^c \overset{\sigma}{\Longrightarrow} q$. Then $\mathcal{C}$ is a (k-)active diagnoser for $\mathcal{S}$ if $h_{\mathcal{C}}$ is one.

IV. Diagnoser construction

We simultaneously solve the decision and synthesis problems. We shall try to construct a pilot-based active diagnoser for a POCS $\mathcal{S}$. The construction succeeds iff $\mathcal{S}$ is actively diagnosable. According to Definition 6, the main challenges in building an active diagnoser are to ensure that (i) the controlled system does not get stuck forever in a non-idle quiescent state, (ii) the controller excludes the ambiguous sequences, and (iii) diagnosis information is provided.

The approach in [2] consisted of two stages. First one builds a Büchi automaton that accepts the infinite unambiguous observed sequences of $\mathcal{A}$. Then using this automaton, one builds a Büchi game where the *Control* player chooses the allowed controllable actions and then the *Environment* player selects the next observable action. The correctness of this approach partly relies on the fact that given two controls $cont$ and $cont'$ and an observed sequence σ of both $\mathcal{S}_{cont}$ and $\mathcal{S}_{cont'}$, σ is ambiguous in $\mathcal{S}_{cont}$ iff it is ambiguous in $\mathcal{S}_{cont'}$. This is no longer the case here. For instance, suppose that in Figure 1 (b) both c, d are controllable. Then $b\delta$ is ambiguous if the controller blocks both c and d, and unambiguous if the controller blocks only c.

Here, our solution consists in directly building a generalized Büchi game and taking into account the control that has already been performed to specify the relevant information that must be memorized to define the winning states.

Definition 8 (game): A game (between two players called Control and Environment) is a tuple $\mathcal{G} = \langle V_C, V_E, E, v_0, \mathcal{P}_F\rangle$, where V_C are the vertices owned by Control, V_E are the vertices owned by Environment; $V_{\mathcal{G}} = V_C \uplus V_E$ denotes all vertices, and $v_0 \in V_C$ is an *initial vertex*. $E \subseteq V_{\mathcal{G}} \times V_{\mathcal{G}}$ is a set of directed edges such that for all $v \in V_C$ there exists $(v, w) \in E$, and $\mathcal{P}_F \subseteq 2^{V_{\mathcal{G}}}$ is a *winning condition*.

A *play* ρ is a sequence of $V_{\mathcal{G}}^{\omega}$ such that $\rho[0] = v_0$ and $\langle \rho[i], \rho[i+1]\rangle \in E$ for all $i \geq 0$; we call $\rho[0, k]$, for some $k \geq 0$, a *partial play* if $\rho[k] \in V_C$, and define $state(\rho[0, k]) := \rho[k]$. We write $Play^*(\mathcal{G})$ for the set of partial plays of $\mathcal{G}$. A play ρ is called *winning* (for Control) if, for all $V \in \mathcal{P}_F$, $\rho[i] \in V$ for infinitely many i.

Definition 9 (strategy): Let $\mathcal{G} = \langle V_C, V_E, E, v_0, \mathcal{P}_F\rangle$ be a game. A *strategy* (for Control) is a function $\theta\colon Play^*(\mathcal{G}) \to V_{\mathcal{G}}$ such that $\langle state(\xi), \theta(\xi)\rangle \in E$ for all $\xi \in Play^*(\mathcal{G})$. A play ρ *adheres* to θ if $\rho[i] \in V_C$ implies $\rho[i+1] = \theta(\rho[0, i])$ for all $i \geq 0$. A strategy is called *winning* if every play ρ that adheres to θ is winning.

Let $M = \langle Q_M, q_M, V_{\mathcal{G}}, T_M\rangle$ be a complete deterministic LTS, where $FM(\xi)$ denotes the state $q \in Q_M$ such that $q_M \overset{\xi}{\Longrightarrow} q$, and let $\alpha\colon Q_M \times V_C \to V_{\mathcal{G}}$ such that for all $q \in Q_M$ and $v \in V_C$ we have $\langle q, \alpha(q, v)\rangle \in E$. The strategy $\theta_{M,\alpha}$ defined by $\theta(\xi) = \alpha(\mathrm{FM}(\xi), state(\xi))$ is called *finite-memory strategy* if Q_M is finite; it is called *one-bit strategy* if $Q_M = \{0, 1\}$.

In the games that we have defined, a play can only be stuck in a state of Environment and considered as losing for this player. Thus we do not consider finite maximal plays for defining the winning strategies of Control.

The controller we are looking for will memorize a tuple of subsets of states $\langle U, V, W, X\rangle$ with the following meaning. Whatever the subset, it represents possible states that have been reached *after the last observable action or that corresponds to a quiescent state w.r.t. the current control*. U represents the possible states reached by a *correct* run, and $V \uplus W$ represent the possible states reached by a *faulty* run. Among the latter, W represents the states for which the controller tries to solve the ambiguity with U, while V is some "waiting room"; the ambiguity between U and V will be resolved later. X represents a *subset* of the possible non-idle states reached by a run for which no action has been performed between the two last observations. The controller tries to discard these states either by observing that they could not occur or by allowing an urgent action. The other states with the same features will be handled later. We denote $S = \{\langle U, V, W, X\rangle \mid U, V, W \subseteq Q \wedge X \subseteq (U \cup V \cup W) \setminus Idle \wedge U \cup V \cup W \neq \emptyset \wedge V \cap W = \emptyset\}$, $Solved_1 = \{\langle U, V, W, X\rangle \in S \mid U = \emptyset \vee W = \emptyset\}$, $Solved_2 = \{\langle U, V, W, X\rangle \in S \mid X = \emptyset\}$, and $Reach(\langle U, V, W, X\rangle) = U \cup V \cup W$. The next definition describes how the controller updates its tuple once an observed action occurs (including the quiescent signal δ). The range of this function also includes the tuple $\langle \emptyset, \emptyset, \emptyset, \emptyset\rangle$ in order to capture the impossible observations.

Definition 10 (Knowledge update): Let $\mathcal{S}$ be a POCS. Then Δ, the *knowledge transition partial function* from $S \times 2^{\Sigma_c} \times (\Sigma_o \cup \{\delta\})$ to $S \cup \{\langle \emptyset, \emptyset, \emptyset, \emptyset\rangle\}$, is defined for $s = \langle U, V, W, X\rangle$, $\Sigma' \in 2^{\Sigma_c}$, $a \in \Sigma' \cup \{\delta\} \cup \Sigma_{uc} \setminus \Sigma_{uo}$ by $\Delta(s, \Sigma', a) := \langle U', V', W', X'\rangle$ as follows.

- When $a \neq \delta$, let $V_a = \{\, q' \mid q \in V, q \overset{\sigma a}{\Longrightarrow} q', \sigma \in \Sigma_{uo}^* \,\} \cup \{\, q' \mid q \in U, q \overset{\sigma a}{\Longrightarrow} q', \sigma \in \Sigma_{uo}^* f \Sigma_{uo}^* \,\}$. Then:
  - $U' = \{\, q' \mid q \in U, q \overset{\sigma a}{\Longrightarrow} q', \sigma \in (\Sigma_{uo} \setminus \{f\})^* \,\}$;
  - If $W = \emptyset$ then $W' = V_a$
    else $W' = \{\, q' \mid q \in W, q \overset{\sigma a}{\Longrightarrow} q', \sigma \in \Sigma_{uo}^* \,\}$;
  - If $W = \emptyset$ then $V' = \emptyset$ else $V' = V_a \setminus W'$;
  - $X' = \emptyset$.
- When $a = \delta$, let $Quiet_{\Sigma'} = \{\, q \in Q \mid en(q) \cap \Sigma_e \subseteq \Sigma_c \setminus \Sigma' \,\}$ and $V_\delta = \{\, q' \in Quiet_{\Sigma'} \mid q \in V, q \overset{\sigma}{\Longrightarrow} q', \sigma \in \Sigma_{uo}^* \,\} \cup \{\, q' \in Quiet_{\Sigma'} \mid q \in U, q \overset{\sigma}{\Longrightarrow} q', \sigma \in \Sigma_{uo}^* f \Sigma_{uo}^* \,\}$. Then:
  - $U' = \{\, q' \in Quiet_{\Sigma'} \mid q \in U, q \overset{\sigma}{\Longrightarrow} q', \sigma \in (\Sigma_{uo} \setminus \{f\})^* \,\}$;
  - If $W = \emptyset$ then $W' = V_\delta$
    else $W' = \{\, q' \in Quiet_{\Sigma'} \mid q \in W, q \overset{\sigma}{\Longrightarrow} q', \sigma \in \Sigma_{uo}^* \,\}$;
  - If $W = \emptyset$ then $V' = \emptyset$ else $V' = V_\delta \setminus W'$;
  - If $X = \emptyset$ then $X' = (Quiet_{\Sigma'} \cap Reach(s)) \setminus Idle$
    else $X' = Quiet_{\Sigma'} \cap X$.

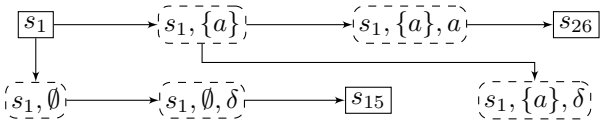

Fig. 2. Excerpt of the Büchi game for the POCS of Figure 1 (c). Environment states are shown with dashed boundary. We use $s_1 := \langle \{1\}, \emptyset, \emptyset, \emptyset \rangle$, $s_{15} := \langle \{1\}, \emptyset, \{5\}, \{1\} \rangle$, and $s_{26} := \langle \{2\}, \emptyset, \{6\}, \emptyset \rangle$.

Before formally defining the game, we discuss its intuition: Control and Environment play alternatingly; the Control chooses the allowed controllable actions based on its knowledge of the possible states, and the Environment chooses an action among those permitted by the Control. The controller states V_C are the tuples $\langle U, V, W, X \rangle$. Once the controller chooses a set of actions, the play moves to a state in $V_C \times 2^{\Sigma_c} \subseteq V_E$, where Environment selects an allowed observable action in Ξ_o and reaches a game state in $V_C \times 2^{\Sigma_c} \times \Xi_o \subseteq V_E$. This state has (1) either a single successor, a controller state whose tuple is given by the above update function when the tuple is *not empty*, (2) or none at all, when the action leads to an empty tuple, meaning that the behaviour was not possible. The generalized Büchi condition is given by $\{Solved_1, Solved_2\}$.

Definition 11 (controller-synthesis game): Let $\mathcal{S}$ be a POCS. We denote $\mathcal{G}(\mathcal{S})$ the game $\langle V_C, V_E, E, v_0, \mathcal{P}_F \rangle$, where $V_C = S$, $v_0 = \langle \{q_0\}, \emptyset, \emptyset, \emptyset \rangle$, $V_E = (V_C \times 2^{\Sigma_c}) \cup (V_C \times 2^{\Sigma_c} \times \Xi_o)$, $E = E_1 \cup E_2 \cup E_3$, where

- $E_1 = \{ \langle s, \langle s, \Sigma' \rangle \rangle \mid s \in S,\ \Sigma' \in 2^{\Sigma_c} \}$;
- $E_2 = \{ \langle \langle s, \Sigma' \rangle, \langle s, \Sigma', a \rangle \rangle \mid s \in S,$ $\Sigma' \in 2^{\Sigma_c}, a \in \Sigma' \cup \{\delta\} \cup \Sigma_{uc} \setminus \Sigma_{uo} \}$;
- $E_3 = \{ \langle \langle s, \Sigma', a \rangle, s' \rangle \mid\ s' = \Delta(s, \Sigma', a) \}$.

and $\mathcal{P}_F = \{Solved_1, Solved_2\}$.

Example 1: Figure 2 depicts an excerpt of the game for for POCS of Figure 1 (c). From the initial state, we have represented two control decisions: either Control allows $\{a\}$ or disallows all controllable actions. If Control chooses $\{a\}$, and Environment chooses action δ, the next state has no successor, since under this control, a is the single observable action, so Environment loses immediately. If Control chooses $\emptyset$ and Environment chooses action δ, the triple of the reached state is $\langle \{1\}, \emptyset, \{5\} \rangle$ as f is lazy and may have occurred or not. The booleans of all next Control states that are reached from the initial state are false, since the initial state belongs to F, and the possible subsets of states after the move (like $\{1, 5\}$ or $\{2, 6\}$) are not included in $Idle$.

We can now address the decision and synthesis problems. To this aim, we shall mainly exploit the following facts: (1) Generalized Büchi games can be solved in polynomial time (see, e.g., [12]), (2) a one-bit winning strategy (when the number of winning conditions is two) can always be chosen for Control if it wins and (3) there is a tight correspondence between winning strategies and active diagnosers.

Let $\xi \in Play^*(\mathcal{G}(\mathcal{S}))$ be a partial play. We define $word(\xi)$ as the observable actions played along ξ, i.e. $word(\varepsilon) = \varepsilon$, $word(\xi v) = word(\xi)$ if $v \notin V_C \times 2^{\Sigma_c} \times \Xi_o$, and $word(\xi \langle v, \Sigma', a \rangle) = word(\xi) a$. In a similar way, $states(\xi)$ are the states of V_C touched along ξ, formally $states(\xi) = \mathcal{P}_{V_C}(\xi)$. We naturally extend these notions to plays ρ.

Definition 12 (from control to strategy and play): Let $cont$ be a controller for $\mathcal{S}$. The strategy θ_{cont} of the game $\mathcal{G}(\mathcal{S})$ is defined as follows. Let $\xi \in Play^*(\mathcal{G}(\mathcal{S}))$ be a partial play ending in a state of the controller. Then $\theta_{cont}(\xi) = cont(word(\xi))$. Let σ be an observed sequence of $\mathcal{S}_{cont}$. Then the play $\xi_{cont}(\sigma)$ is inductively defined by:

- If $\sigma = \varepsilon$ then $\xi_{cont}(\sigma) = \langle \{q_0\}, \emptyset, \emptyset, \emptyset \rangle$.
- If $\sigma = \sigma' a$, then $\xi_{cont}(\sigma) = \xi_{cont}(\sigma') \langle l, \Sigma' \rangle \langle l, \Sigma', a \rangle s'$, where $l := last(\xi_{cont}(\sigma'))$, $\Sigma' := cont(\sigma')$, and s' is the single successor of $\langle l, \Sigma', a \rangle$ in $\mathcal{G}(\mathcal{S})$.

The following observation is straightforward. Let ξ be an infinite play that adheres to strategy θ_{cont}. Then $word(\xi)$ is an observed sequence of $\mathcal{S}_{cont}$.

Proposition 1: If $cont$ is a controller for $\mathcal{S}$ and σ an observed sequence of $\mathcal{S}_{cont}$, the following are equivalent:

1) the play $\xi_{cont}(\sigma)$ is winning for the controller;
2) σ is unambiguous and for all $\sigma' = q_0 a_1 \cdots a_n (q_n \delta)^\omega$ such that $\mathcal{P}(a_1 \cdots) = \sigma$, $q_n \in Idle$.

We can now state the main result of this section:

Theorem 1: Let $\mathcal{S}$ be a POCS with n states and m controllable actions. The active-diagnosis decision and synthesis problems for $\mathcal{S}$ can be solved in $2^{\mathcal{O}(n+m)}$ time. If $\mathcal{S}$ is actively diagnosable, then one can synthesize a pilot $\mathcal{C}$ with at most $2 \cdot 11^n$ states, where $\mathcal{C}$ is an active diagnoser for $\mathcal{S}$.

By a straightforward adaptation of the proof of Theorem 1 in [13], we can prove EXPTIME-hardness of the decision problem. So we get the following corollary.

Corollary 1: The active diagnosis decision problem is EXPTIME-complete.

Observe that in any play of a winning one-bit strategy, one can visit at most twice (one per bit value) the same state while U and W remain non empty since otherwise, a losing play could be built ending with a loop. So once a fault has occurred, there are two possible situations: (1) either W remains non empty and after at most $2|V_C|+1$ observations, U becomes empty, or (2) while U remains non empty, W becomes empty after at most $2|V_C|+1$ observations and filled again by at least the state in V that corresponds to the faulty run and then after at most $2|V_C|$ additional observations U becomes empty. Summarizing, the delay achieved by our active diagnoser is at most $4|V_C| + 1 \leq 4 \cdot 11^n + 1 = 2^{\mathcal{O}(n)}$. The active diagnoser that we synthesize does not necessarily have minimal delay. However Theorem 5 of the next section shows that $2^{\mathcal{O}(n)}$ states are not enough for obtaining such an active diagnoser, and Theorem 4 shows that for some systems, the minimal achievable delay is indeed exponential. For completeness, we mention the following result whose proof can be found in [8].

Theorem 2: Let $\mathcal{S}$ be an actively diagnosable POCS with n states. One can construct a pilot $\mathcal{C}$ with $2^{\mathcal{O}(n^2)}$ states such that $\mathcal{C}$ is an active diagnoser for $\mathcal{S}$ with minimal delay.

V. Lower bounds

In this section, we establish that the active diagnoser built in the proof of Theorem 1 is almost optimal w.r.t. the number

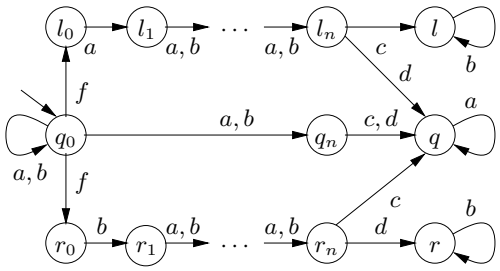

Fig. 3. A POCS with $\Sigma_o = \{a, b, c, d\}$, $\Sigma_c = \{c, d\}, \Sigma_\ell = Idle = \emptyset$.

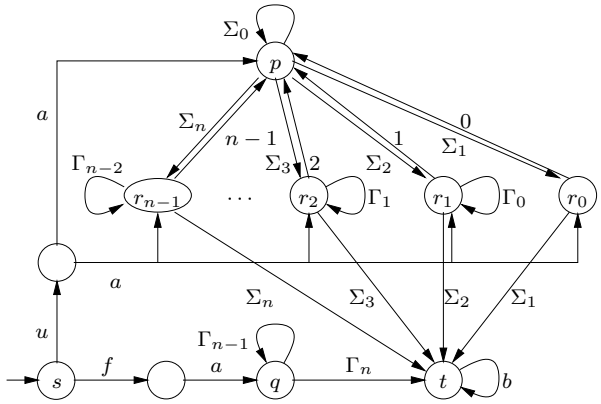

Fig. 4. A system whose delay is $2^n + 2$. The observable actions are $\Sigma_o = \{a, b, 0, \ldots, n\}$, the controllable actions $\Sigma_c = \{0, \ldots, n\}$, and we denote $\Sigma_i := \{i, \ldots, n\}$ and $\Gamma_i := \{0, \ldots, i\}, \Sigma_\ell = Idle = \emptyset$.

of states and the delay before fault detection provided by any diagnoser (both in $2^{\mathcal{O}(n)}$ for our construction). We remark that the lower bounds in this section match those shown in [2] despite the extended capabilities of our diagnosers. The examples demonstrating the lower bounds are inspired by [2] but had to be carefully adapted to deal with the capacity of controllers to detect quiescence.

Theorem 3 (lower bound for active-diagnoser): There is a family $(\mathcal{S}_n)_{n\geq 1}$ of actively diagnosable POCS such that the size of $\mathcal{S}_n$ is $\mathcal{O}(n)$ and any pilot that is an active diagnoser for $\mathcal{S}_n$ has at least 2^n states.

$\mathcal{S}_n$ is shown in Figure 3. Its key idea is that the controller must remember a sequence of the n last observed actions that are a or b. Theorem 3 means that an active diagnoser with $2^{\Omega(n)}$ states may be required for a POCS of size n.

Theorem 4: There exists a family $(\mathcal{S}_n)_{n\geq 1}$ of actively diagnosable POCS such that $\mathcal{S}_n$ has $\mathcal{O}(n)$ states and $\mathcal{S}_n$ has an index $\geq\ 2^n + 2$.

In Figure 4, states $r_0 \ldots r_{n-1}$ implement a binary counter initialized to $2^n - 1$ after reading the first a, which has to be decremented one by one to zero before a diagnosis. Theorem 4 means that a delay of $2^{\Omega(n)}$ may be the best possible delay for a POCS of size n.

Theorem 5 (minimal-delay diagnoser): There exists a family $(\mathcal{S}_n)_{n\geq 1}$ of $f(n)$-actively diagnosable POCS (for some function f) with $\mathcal{O}(n)$ states such that for all n, any pilot that is an active diagnoser for S_n has at least $n!$ states.

Consider Figure 5, where in order to guarantee a delay $n + 3$ one has to remember the sequence of the last n

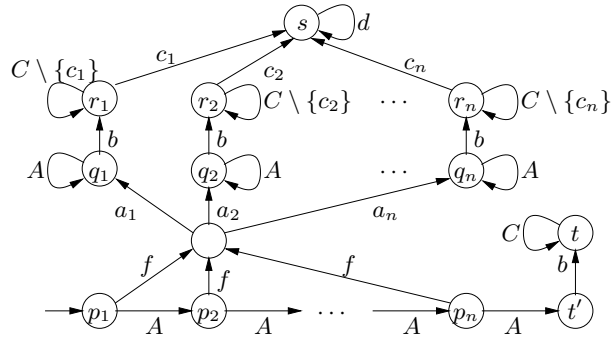

Fig. 5. A POCS $\mathcal{S}_n$ whose minimal-delay active diagnoser requires at least $n!$ states. In the figure, $A := \{a_1, \ldots, a_n\}$, $C := \{c_1, \ldots, c_n\}$, $\Sigma_o = A \cup C \cup \{b, d\}$, $\Sigma_c = C, \Sigma_\ell = Idle = \emptyset$.

observation (in case when they are pairwise different) i.e. a permutation of actions $\{a_1 \ldots a_n\}$. Thus an active diagnoser with $2^{\Omega(n \log(n))}$ states may be required for a POCS of size n to achieve minimal delay.

Note that the families of examples in Theorems 4 and 5 use alphabets of variable size. However, the results also hold for fixed-size alphabets; the proofs are contained in [8].

VI. Conclusion

We have extended the scope of active diagnosis in two directions: (1) enlarging the behaviour of the system with modalities for actions and (2) allowing the controller to observe quiescence of the system. We have adressed this problem by modeling it by a Büchi game and obtained almost optimal decision and synthesis procedures. We plan to study other extensions like action priorities and multiple faults. We also want to analyze the safe active diagnosis introduced in a probabilistic framework [6], where the active diagnoser is required to preserve correct behaviours as most as possible.

References

[1] M. Sampath, S. Lafortune, and D. Teneketzis, "Active diagnosis of discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 43, no. 7, pp. 908–929, July 1998.

[2] S. Haar, S. Haddad, T. Melliti, and S. Schwoon, "Optimal constructions for active diagnosis," in *Proc. FSTTCS*, ser. Leibniz International Proceedings in Informatics, vol. 24, 2013, pp. 527–539.

[3] F. Cassez and S. Tripakis, "Fault diagnosis with static and dynamic observers," *Fundamenta Informaticae*, vol. 88, pp. 497–540, 2008.

[4] D. Thorsley and D. Teneketzis, "Active acquisition of information for diagnosis and supervisory control of discrete-event systems," *Journal of Discrete Event Dynamic Systems*, vol. 17, pp. 531–583, 2007.

[5] E. Chanthery and Y. Pencolé, "Monitoring and active diagnosis for discrete-event systems," in *Proc. SafeProcess*, 2009, pp. 1545–1550.

[6] N. Bertrand, E. Fabre, S. Haar, S. Haddad, and L. Hélouët, "Active diagnosis for probabilistic systems," in *Proc. FOSSACS*, vol. LNCS 8412, 2014, pp. 29–42.

[7] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems - Second Edition*. Springer, 2008.

[8] S. Böhm, S. Haar, S. Haddad, P. Hofman, and S. Schwoon, "Active diagnosis with observable quiescence," LSV, ENS Cachan, France, Research Report LSV-15-01, April 2015.

[9] J. Tretmans, "Conformance testing with labelled transition systems: Implementation relations and test generation," *Computer Networks and ISDN Systems*, vol. 29, no. 1, pp. 49–79, 1996.

[10] ——, "Test generation with inputs, outputs and repetitive quiescence," *Software - Concepts and Tools*, vol. 17, no. 3, pp. 103–120, 1996.

[11] ——, “Model based testing with labelled transition systems,” in *Formal Methods and Testing*, vol. LNCS 4949, 2008, pp. 1–38.
[12] E. Grädel, W. Thomas, and T. Wilke, Eds., *Automata, Logics, and Infinite Games*. Springer, 2002, vol. LNCS 2500.
[13] S. Haar, S. Haddad, T. Melliti, and S. Schwoon, “Optimal constructions for active diagnosis,” LSV, ENS Cachan, France, Research Report LSV-13-12, July 2013.