A Framework for Optimization of Sensor Activation using Most Permissive Observers

Eric Dallal
University of Michigan
Department of EECS
edallal@umich.edu

Stéphane Lafontune
University of Michigan
Department of EECS
stephane@eecs.umich.edu

Abstract—This paper considers the problem of finding dynamic sensor activation policies that satisfy the property of $K$-diagnosability for discrete event systems modeled by finite state automata. We begin by choosing a suitable information state for the problem and defining a controller. We then define a structure called the most permissive observer, which provides all feasible solutions for the controller (i.e., for sensor activations). By formulating the problem as a state disambiguation problem, we prove a number of monotonicity properties about the information state. Finally, we show that this formulation allows us to efficiently compute the set of all satisfactory control decisions at all points in the system’s execution.

I. INTRODUCTION

This paper studies problems of dynamic sensor optimization in discrete event systems. Our model assumes that a system is modeled as a deterministic finite state automaton and that we have sensors that can detect the occurrence of only a subset of events. These events are classified as “monitorable” and we may choose not to observe them. In engineering problems, this might be useful in order to save energy in the sensors if these are costly to operate. Alternatively, there may be high energy requirements involved in communicating the sensor measurements. Another possibility is that we would like to minimize communication of sensor measurements for the purpose of maintaining security. For any of these or other reasons, we assume that we would like to minimize sensor activations (for a more detailed explanation of the motivation for dynamic sensor activation, see [1]. For a concrete example, see [2]). Additionally, we assume that, among the remaining inherently unobservable events, there is a significant event whose occurrence we would like to be able to determine. In this work, we consider the event in question to be a fault event. The control problem is to determine which sensors to turn on/off at every point in time while maintaining the property of $K$-diagnosability. The $K$-diagnosability property is similar to that of diagnosability as found in [3], but where the diagnosis must be made with a delay of at most $K$ events. Specifically, the system must always be able to distinguish between faulty and non-faulty executions within $K + 1$ events after a fault.

A number of other recent works have considered the problem of optimizing sensor activations while maintaining some notion of diagnosability; see, e.g., [1], [4], [5]. In [4], dynamic programming is used to find an optimal solution to the problem of optimizing sensor activations while maintaining the property of diagnosability simultaneously for automata with multiple fault types. In [5], game theory is used first to define a structure called most permissive observer (which will be redefined in this paper). Results from the theory of games on graphs are then used, resulting in algorithms that solve the optimal sensor activation problem. Their methodology is extended to opacity in [6]. In [1], the problem of decentralized, or multi-agent, diagnosis is solved through a “window-based partition” approach. Algorithms that run in time polynomial in the window partition size are given for both the centralized and decentralized event diagnoses cases.

Several criteria can be used for guiding the solution of the dynamic sensor activation problem. A simple one is that of state disambiguation: sensors must be activated often enough so that certain pairs of discrete states should never be confused at run-time, along any system trajectory. State disambiguation requirements arise naturally at the discrete-event level. For example, the discrete-event-theoretic property of observability [7], which is part of the necessary and sufficient conditions for the existence of discrete-event controllers for partially-observed systems, can be expressed as a state disambiguation property (cf. [8]). We demonstrate in this paper that $K$-diagnosability can be mapped to a problem of state disambiguation over a suitable refinement of the discrete state space. Hence, our results can be adapted, with simple modifications, to solve a standard state disambiguation problem.

Our work is most closely related to the approach of [5], in that the goal is maintaining the $K$-diagnosability property for a system modeled by an automaton, and that we both define the “most permissive observer” (MPO), a discrete structure that captures all controllers that constitute feasible solutions. In [5], the MPO is defined through game theory and the theory of games on graphs. Our MPO is defined as a bipartite graph over suitable information states. Although we both end up with the same structure, our goal is to develop an information state based approach to the construction of the MPO, which allows for two benefits: first, a clearer understanding of informational properties of the MPO; second, a greater adaptability in being applied to other dynamic optimization problems in discrete event systems. As in [5], we tackle the dynamic control problem in two parts: first, we construct the MPO which contains all controllers that satisfy the $K$-diagnosability property; second, we use the MPO as a “feasible space” over
which to optimize. This work addresses the first part. With the MPO, we can then use techniques from [5] or [4] to find a single optimal solution.

Our notion of MPO was first introduced in [9]. In [9], we proved that our information state was “sufficient” in the sense that no generality is lost by examining only controllers based on that information state. Section II of this paper will recall some definitions from [9]. In the main body of this paper, we show how to map the $K$-diagnosability problem to the state disambiguation problem. The most significant contribution of this paper is that this approach allows us to establish the existence of a simple and efficient test on the information state for determining whether or not any particular control decision maintains the $K$-diagnosability property. We also characterize informational properties of the MPO and then present an efficient algorithm for its construction.

For some references on other concepts of observers in logical discrete-event models, see [10]–[13].

The rest of this paper is organized as follows. Section II describes the model under consideration, followed by a number of definitions related to our information state. Section III formally defines the $K$-diagnosability property, explains how to formulate it as a state disambiguation problem, and defines the recursive structure of the MPO. Section IV establishes a number of theorems that show how the “extended specification” (from the literature on state disambiguation, see [14]) can be used to efficiently compute the MPO online. Finally, we have conclusions and references. Note that all proofs have been omitted due to space constraints.

II. Augmented States and the Information State

Consider a deterministic automaton $G = (X,E,f,x_0)$, where $X$ is the set of states, $E$ is the set of events, $f : X \times E \to X$ is the system’s partial transition function, and $x_0$ is the initial state. Define $L(G)$ to be the language of $G$, that is, the set of all strings of events that can occur in $G$. Assume that the set $E$ is partitioned into four disjoint sets $E = E_o \cup E_s \cup E_{uo} \cup E_f$, each corresponding to different categories of events: $E_o$ is the set of events that are always monitored, $E_s$ is the set of events that are observable but may not always be monitored (for example, if a sensor is turned off), $E_f$ is the set of unobservable faulty events (or any set of important unobservable events whose occurrence we would like to diagnose) and $E_{uo}$ is the set of unobservable (non-faulty) events. In this work, we assume that there is only one fault event.

We wish to dynamically diagnose any occurrence of the fault event. Here, we use the notion of $K$-diagnosability (formally defined in section III), which states that we must be able to diagnose the occurrence of any fault (with certainty) within at most $K + 1$ events after the fault event. At each point in the system’s execution, we wish to find the set of all control decisions that allow the system to maintain the $K$-diagnosability property, after each run of control decisions and observed events. The MPO, which we will define in section III and show how to construct in section IV, contains all controllers that constitute feasible solutions to this problem.

The remainder of this section briefly provides the mathematical preliminaries necessary for defining the MPO as it pertains to the problem of $K$-Diagnosability. Parts of this section are recalled or modified from [9], where we first introduced our MPO.

Definition II.1 (Augmented State). The augmented state is a pair $(x_n, n) \in X \times \{-1,0,1,\ldots\}$, where, $n$ represents a “count” of the number of events (of all kinds) that have occurred since a fault event occurred, or $-1$ if no fault event has occurred. The set of such states is denoted by $X^+ = X \times \{-1,0,1,\ldots\}$. The initial augmented state is $x_0^+ = (x_0, -1)$. For any augmented state $x_n^+ \in X^+$, let the state and count components be denoted by $S(x_n^+)$ and $N(x_n^+)$, respectively, so that $x_n^+ = (S(x_n^+), N(x_n^+))$.

Definition II.2 (Augmented Transition Function). We next define the augmented transition function $g : X^+ \times E \to 2^{X^+}$ on augmented states that is induced by the automaton $G = (X,E,f,x_0)$ and the partition of event set $E$. Formally, for any $u = (x_n,u_n)$ and event $e$, we have:

- **Case 1**: If $u_n = -1$ and $e \notin E_f$ then $g(u,e) = \{f(x_n,u,e), -1\}$.
- **Case 2**: If $u_n = -1$ and $e \in E_f$ then $g(u,e) = \{f(x_n,u,e), 0\}$.
- **Case 3**: If $u_n \geq 0$ and $e \notin E_f$ then $g(u,e) = \{f(x_n,u,e), n_u + 1\}$.
- **Case 4**: If $u_n \geq 0$ and $e \in E_f$ then $g(u,e) = \{f(x_n,u,e), 0, f(x_n,u,e), n_u + 1\}$.

It is because of this last case that $g$ is defined as a mapping from augmented states to sets of augmented states rather than simply to augmented states. So that the function can be applied iteratively, we define $g$ applied to a set as the union of $g$ applied to each augmented state in the set. Mathematically, we write: $g(U,e) = \bigcup_{u \in U} g(u,e)$. This definition is extended to strings (rather than merely events) in the usual way. Finally, we define $g(s) = g(x_0^+, s)$ for brevity.

Definition II.3 (Augmented Automaton). We define the augmented automaton as the automaton $G^+ = (X^+, E, g, x_0^+)$ defined over augmented states that is induced from the original automaton $G$. Specifically, we take $X^+$ and $x_0^+$ as defined in Definition II.1 and $g$ as defined in Definition II.2 (the event set $E$ remains the same). Note that $G^+$ might not be deterministic, even if $G$ is, due to case 4 in the definition of the augmented state transition function $g$. Furthermore, without a bound on the count component of augmented states, $G^+$ might not be finite in size (in section IV, we will use a trimmed version of $G^+$ that avoids this problem).

Definition II.4 (Run). A run $\rho$ of length $n$ is defined as a sequence $C_0, e_0, \ldots, C_{n-1}, e_{n-1}$ of control decisions or sensor activations (the $C_i$’s, which are subsets of events to monitor) and observed events (the $e_i$’s). Since the events are observed, they must be among the monitored events. That is, $e_i \in C_i$, for all $i = 0, \ldots, n - 1$. On the other hand, the strict
alternation of control decisions and observed events reflects
the assumption that control decisions are only changed upon
the observance of an event. Denote by \( R_n \) the set of runs of
length \( n \). Finally, let \( \rho(k) = C_0, e_0, \ldots, C_{k-1}, e_{k-1} \) denote
the subsequence of \( \rho \) of length \( k \).

**Definition II.5** (Admissible Controller). An admissible con-
troller is a sequence \( C = (C_0, C_1, \ldots) \) of functions \( C_n : R_n \to 2^E \), where \( n = 0, 1, \ldots \) from runs to control decisions that satisfies the following two conditions:

1) \( C_n(\rho) \supseteq E_o \) for all \( \rho \in R_n \) and all \( n = 0, 1, \ldots \) (\( C_n \)
cannot includes the set of events that are always monitored).
2) \( C_n(\rho) \cap (E_u \cup E_f) = \emptyset \) for all \( \rho \in R_n \) and all \( n = 0, 1, \ldots \) (\( C_n \)
cannot include any event that is unobservable, whether faulty or non-faulty).

Note that \( C_n \) is used to denote both the controller and the
control decision. Hereafter, the context will make it clear
which is which.

**Definition II.6** (Information State). An information state (IS)
is a subset \( S \subseteq X^+ \) of augmented states. We will denote by
\( I = 2^{X^+} \) the set of information states.

**Definition II.7** (Information State Based Controller). An infor-
mation state based controller (or IS-controller) is a function \( C : I \to 2^E \) that satisfies the two conditions of an admissible controller (i.e., \( C(i) \supseteq E_o \) and \( C(i) \cap (E_u \cup E_f) = \emptyset \) for all \( i \in I \)).

Constructing an observer for an automaton and a fixed set
of unobservable events is a relatively simple task. To construct
the observer when the set of observable events is a dynamic
control decision, we must explicitly model the effect of the
controller on the evolution of the information state.

**Definition II.8** (Total Observer). The total observer is
defined as a directed bipartite graph \( (Y \cup Z, A) \). Here, \( Y \)
is the set of information states (i.e., \( Y = I \)), \( Z \) is the set of
information states augmented with monitored event decisions
(i.e., \( Z = I \times 2^E \)), and \( A \) is the set of edges in the graph. A
\( Y \) state is a (information) state in which a control decision
is taken and a \( Z \) state is a pair consisting of a (information)
state and a set of monitored events, in which an observable
event occurs, among those in the current set of monitored
events. Thus, any run results in the alternation between \( Y \)
and \( Z \) states. The set \( A \) contains all transitions from \( Y \)
states to \( Z \) states (all admissible control decisions) and all
transitions from \( Z \) states to \( Y \) states (all observable events).
Specifically, a transition from a \( Y \) state to a \( Z \) state represents
the unobservable reach. As a transition from a \( Z \) state to a
\( Y \) state occurs upon the observance of a monitored event,
it is necessary for each \( Z \) state to “remember” the set of
monitored events from the \( Y \) state that led to it. Let \( I(z) \)
and \( C(z) \) denote \( z \)’s information state and control decision
components, respectively, so that \( z = (I(z), C(z)) \). Formally,
\( (y, z) \in A \), labeled with \( C(y) \) if:

\[
I(z) = \left\{ v \in X^+ : (\exists u \in y)(\exists t \in (E \setminus C(y))^+) \right\}
\]

\[
= \bigcup_{u \in y} \bigcup_{t \in (E \setminus C(y))^+} g(u, t)
\]

\[
C(z) = C(y)
\]

where \( g(\cdot, \cdot) \) is the augmented transition function. In words,
this means that \( I(z) \) is the set of augmented states reachable
from some augmented state of the preceding \( Y \) state through
some string of unmonitored events, and that \( C(z) \) is the set of
monitored events chosen in the preceding \( Y \) state. We write
\( h_{YZ}(y, C(y)) = z \). Formally, \( (z, y) \in A \), labeled with \( e \in C(z) \) if:

\[
y = \{ v \in X^+ : (\exists u \in I(z)) \ [v \in g(u, e)] \}
\]

\[
= \bigcup_{u \in I(z)} g(u, e)
\]

In words, this means that \( y \) is the set of augmented states
reachable from some augmented state of the information state
component of the preceding \( Z \) state through the single event
\( e \). As before, we write \( h_{ZY}(z, e) = y \). The initial state of the
system is the \( Y \) state corresponding to the initial information
state, i.e., \( y_0 = \{ x_0^0 \} \).

**Definition II.9** (\( Y \) and \( Z \) state Controller Induced Information
State Evolution). Given a controller \( C \), we define \( IS_C^Y(y, s) \)
to be the \( Y \) state that results from the occurrence of string \( s \),
when starting in \( Y \) state \( y \). This can be computed as follows:

\[
IS_C^Y(y, s) := y
\]

\[
IS_C^Y(y, e) := \begin{cases} h_{YZ}(h_{YZ}(y, C(y)), e) & \text{if } e \in C(y) \\ y & \text{if } e \notin C(y) \end{cases}
\]

\[
IS_C^Y(y, es) := IS_C^Y(IS_C^Y(y, e), s)
\]

For brevity, we define \( IS_C^Z(y, s) := IS_C^Y(y, 0, s) \). We define
\( IS_C^Z(z, s) \) analogously:

\[
IS_C^Z(z, s) := z
\]

\[
IS_C^Z(z, e) := \begin{cases} h_{YZ}(y', C(y')) & \text{if } e \in C(z) \\ z & \text{if } e \notin C(z) \end{cases}
\]

\[
IS_C^Z(y, es) := IS_C^Z(IS_C^Z(z, e), s)
\]

As before, we define \( IS_C^Z(s) := IS_C^Z(z_0, s) \), where \( z_0 = h_{YZ}(y_0, C(y_0)) \) (which is well defined for a fixed controller).

For a fixed set of monitored events, it is a trivial task to
define the projection of a string. When the set of monitored
events changes dynamically along the string’s execution, in
a way that depends on the particular controller \( C \), it is necessary
to define a controller induced projection.

**Definition II.10** (Controller Induced Projection). Given a
controller \( C \), we define \( P_C(z, s) \) as the string \( t \) that is observed
upon the occurrence of the string \( s \), when starting in \( Z \) state

2713
This can be computed as follows:

\( P_C(z, \varepsilon) := \varepsilon \)

\( P_C(z, e) :=
\begin{cases} 
  e \text{ if } e \in C(z) \\
  \varepsilon \text{ if } e \notin C(z)
\end{cases} 
\)

\( P_C(z, e') :=
\begin{cases} 
  eP_C(z', s) \text{ if } e' \in C(z) \\
  P_C(z, s) \text{ if } e' \notin C(z)
\end{cases} 
\)

where \( z' = h_{YZ}(y', C(y')) \) and \( y' = h_{YZ}(z, e) \)

For the last case, the first argument of \( P_C \) must be updated with the new \( Z \) state if event \( e \) is observed. For brevity, we define \( P_C(s) := P_C(z_0, s) \).

III. \( K \)-DIAgnosability, State Disambiguation, and the Most Permissive Observer

This section consists of three parts. In the first part, we formally define the property of \( K \)-Diaognosability. In the second part, we briefly describe the state disambiguation problem and show that the \( K \) diagnosability property can be formulated as a state disambiguation problem. Finally, we define the Most Permissive Observer (MPO) in the last part.

Definition III.1 (\( K \)-Diaognosability). We recall the standard definition of diagnosability from [3]. Adapted for a fixed \( K \) and the context of a dynamic observer, we say that a system \( G \) is \( K \)-diagnosable given controller \( C \) if there do not exist a pair of strings \( s_Y, s_N \in \mathbb{L}(G) \) such that:

1. \( s_Y \) has an occurrence of a fault event \( f \in E_f \) and \( s_N \) does not.
2. \( s_Y \) has at least \( K + 1 \) events after the fault event \( f \)
3. \( P_C(s_Y) = P_C(s_N) \), that is, the observed string of events is identical given the controller.

We also say that a system \( G \) is \( K \)-diagnosable if there exists a controller \( C \) such that \( G \) is \( K \)-diagnosable given controller \( C \). We call such a controller safe.

Definition III.2 (State Disambiguation Problem). The state disambiguation problem is defined as a triple \((G^{sd}, \Sigma_0, T_{spec})\), where \( G^{sd} = (X^{sd}, \Sigma, f^{sd}, q_0^{sd}) \) is an automaton, \( \Sigma_0 \subseteq \Sigma \) is a set of monitorable events, and \( T_{spec} \subseteq X^{sd} \times X^{sd} \) is a set of pairs that must not be confused. The state disambiguation problem consists of finding a controller \( C \) for \( G^{sd} \), which chooses sensors to activate, such that the state of \( G^{sd} \) is never confused between any pair of states in the specification \( T_{spec} \).

The controller \( C \) is defined as a sequence of functions from runs to control decisions, as in the previous section. That is, \( C = (C_0, C_1, \ldots) \), where \( C_n : R_n \rightarrow 2^{\Sigma_0} \). Using the notation defined in the previous section, we can define the problem formally as that of finding a controller \( C \) such that:

\[ s_1, s_2 \in \mathbb{L}(G) : P_C(s_1) = P_C(s_2) \Rightarrow (f(x_0, s_1), f(x_0, s_2)) \notin T_{spec}. \]

To formulate the \( K \)-diagnosability problem as a state disambiguation problem, we specify each of \( G^{sd}, \Sigma_0, \) and \( T_{spec} \):

- \( G^{sd} = G^+_K \) which is the automaton of Definition II.3, but restricted to augmented states having counts no greater than \( K + 1 \).

- \( \Sigma_0 = E_0 \cup E_s \) is the set of monitorable events (we assume that \( C \) is an admissible controller).

- Finally, we define \( T_{spec} \) as:

\[ T_{spec} = \{ (u, v) \in X^+ \times X^+ \text{ s.t. } N(u) = -1 \text{ and } N(v) = K + 1 \} \]

Comparing definitions III.1 and III.2, we see that this state disambiguation problem can be satisfied if and only if there do not exist two strings \( s_1 \) and \( s_2 \) with \( P_C(s_1) = P_C(s_2) \), \( N(g(x_0^+, s_1)) = -1 \), and \( N(g(x_0^+, s_2)) = K + 1 \). Recall that \( N(g(x_0^+, s)) \) is equal to \(-1\) if there is no fault in \( s \), and the number of events since a fault event otherwise. If we take \( s_1 \) and \( s_2 \) in this problem to correspond to \( s_N \) and \( s_Y \) of definition III.1, we see that the two problems are identical, except for the fact that, in the definition of \( K \)-diagnosability \( s_Y \) must have at least \( K + 1 \) events after a fault, whereas in \( T_{spec} \) the string \( s_2 \) has exactly \( K + 1 \) events after a fault. To see that this makes no difference, suppose that there exist two strings \( s_N \) and \( s_Y \) that violate \( K \)-diagnosability and that \( s_Y \) has \( r \) events after a fault. Then we may simply truncate \( s_Y \) to obtain an \( s'_Y \) with exactly \( K + 1 \) events after a fault. If this shortens the projection \( P_C(s_Y) \), we can shorten \( s_N \) as well so that \( P_C(s'_Y) = P_C(s'_N) \).

Definition III.3 (\( K \)-diagnosable binary function for information states). An information state \( i \in I \) violates \( K \)-diagnosability if there exist two augmented states \( x_1^+, x_2^+ \in i \) where \( x_1^+ = (x_1, -1) \) and \( x_2^+ = (x_2, n) \) for some \( n > K \). In light of the definition of \( T_{spec} \), we define the \( K \)-diagnosability binary function for information states \( D_I : I \rightarrow \{0, 1\} \) as:

\[ D_I(i) = \begin{cases} 
  0, & \exists u, v, \in I : (u, v) \in T_{spec} \\
  1, & \text{else}
\end{cases} \]

In words, \( D_I(i) = 1 \) if and only if \( i \) does not violate the \( K \)-diagnosability property.

Definition III.4 (\( K \)-diagnosability binary functions for \( Y \) and \( Z \) states). Given any safe controller, we now define the notion of safe \( Y \) and \( Z \) states. Specifically, we say that a \( Y \) state is safe if it currently satisfies the \( K \) diagnosability property and there exists some controller that maintains the \( K \)-diagnosability property for all future executions of the system. Since we can choose control decisions but not event occurrences, we define a \( Z \) state to be safe if all of its successor \( Y \) states are safe. We therefore define two \( K \)-diagnosability binary functions, \( D_Y : Y \rightarrow \{0, 1\} \) and \( D_Z : Z \rightarrow \{0, 1\} \) (similar to \( D_I \), but for \( Y \) and \( Z \) states) as follows:

\[ D_Y(y) = \begin{cases} 
  1 & \text{if } D_I(y) = 1 \text{ and } \exists C(y) : D_Z(h_{YZ}(y, C(y))) = 1 \\
  0 & \text{else}
\end{cases} \]

\[ D_Z(z) = \begin{cases} 
  1 & \text{if } D_I(I(z)) = 1 \text{ and } D_Y(h_{YZ}(z, e)) = 1 \quad \forall e \in C(z) \\
  0 & \text{else}
\end{cases} \]
From these definitions, we can say that $G$ is $K$-
diagnosable if and only if $D_1(y_0) = 1$, and $\exists C(y_0):\ D_1(h_{Y Z}(y_0, C(y_0))) = 1$, and $\exists C(y_0) \ni e_0 \in C(y_0):\ D_1(h_{Y Z}(y_0, C(y_0), e_0)) = 1$, etc... Put in terms of the
information state evolution, we can equivalently say that $G$ is $K$-
diagnosable if and only if $D_1(y_0) = 1$, and $\exists C(y_0) :\ D_1(h_{Y Z}(y_0, C(y_0)))$ ... $\subseteq C_1$.

Lemma IV.3 (Equality of projection when everything is
observed is equivalent to equality of projection under all

Definition III.5 (Fault diagnosis binary function). Define the fault diagnosis binary function $D_F : Y \to \{0, 1\}$ as follows:

$$D_F(y) = \begin{cases} 1 & \text{if } N(u) \neq -1 \ \forall u \in y \\ 0 & \text{else} \end{cases} \quad (14)$$

In words, this means that $D_F(y) = 1$ if and only if all possible executions $s \in L(G)$ resulting in information state $y$ had a
fault occurrence. A $Y$ state $y$ satisfying $D_F(y) = 1$ is called diagnosed.

Definition III.6 (Most Permissive Observer). The most per
missive observer is defined as the $K$-diagnosable connected
subgraph of the total observer that includes state $y_0$, together
with an additional state $F$ (called the “fault detected” state).
By the $K$-diagnosable subgraph of the total observer, we
mean the subgraph of the total observer consisting only of $K$-
diagnosable $Y$ and $Z$ states, and the transitions between them.
The recursive structure of $D_Y$ and $D_Z$ therefore captures all
“safe” control decisions $C(y)$, for all $y \in Y$ (and only safe
control decisions), where by safe we mean that these decisions
do not cause an immediate or unavoidable eventual violation of
$K$-diagnosability. The single state $F$ is used to denote any state
where a fault has been diagnosed. That is, a state $y$ satisfying
$D_F(y) = 1$. All such $Y$ states are replaced by the single state
$F$, which is a terminal node in the sense that there are no
transitions out of it (i.e., we make no further control decisions
from $F$).

IV. ALGORITHMIC ASPECTS OF THE MPO

This section consists of three parts. In the first part, we
present the extended specification and describe how to compute it. In the second part, we prove a number of interesting informational properties about the MPO, culminating in two
theorems providing for a simple test in determining the safety of the $Y$ and $Z$ states of the MPO. Finally, we give an
algorithm for constructing the MPO in the last part.

Definition IV.1 (Extended Specification). To begin, we repeat the definition of the extended specification found in [14]. To do so, we must first define the natural projection $P : E^* \to (E_0 \cup E_s)^*$ as:

$P(\varepsilon) := \varepsilon$

$P(e) := \begin{cases} e & \text{if } e \in E_0 \cup E_s \\ \varepsilon & \text{if } e \notin E_0 \cup E_s \end{cases}$

$P(e \cdot s) := P(e) P(s)$

The extended specification is therefore defined as the set of all augmented state pairs that cannot be confused because even if all the sensors in $E_0 \cup E_s$ are turned on for the rest of time, there still exists some sequence of events such that some pair in $T_{\text{spec}}$ will be confused:

$$T_{\text{spec}} = \{ (u, v) \in X^+ \times X^+ : \exists s_1, s_2 \text{ s.t. } P(s_1) = P(s_2) \text{ and } (g(u, s_1), g(v, s_2)) \in T_{\text{spec}} \}$$

There are a number of approaches for computing the extended specification $T_{\text{spec}}$. One approach is to explicitly compute the automaton $G_{\text{spec}}$, reverse its transitions and then do a depth-first search (DFS). Another method is to reverse the augmented state transition function $g$ on-line, while doing a DFS. The first algorithm has the obvious disadvantage of having greater overhead but the substantial advantage of potentially resulting in a smaller $T_{\text{spec}}$, since many of the augmented states encountered in the second method might not actually be reachable. Either way, the algorithm comes down to a DFS over the space of pairs of augmented states with reversed transitions. Specifically, we make use of the fact that, if some event $e$ is unobservable, then confusing $u, v \in X^+$ implies confusing $g(u, e)$ and $v$ (as well as $u$ and $g(v, e)$), whereas if $e$ is observable, then confusing $u, v \in X^+$ implies confusing $g(u, e)$ and $g(v, e)$.

Definition IV.2 (Extended specification binary function for information states). In light of the definition of $T_{\text{spec}}$, we define the extended specification binary function for information states $D_{I} : I \to \{0, 1\}$ as follows:

$$D_{I}(i) = \begin{cases} 0, & \exists u, v \in I : (u, v) \in T_{\text{spec}} \\ 1, & \text{else} \end{cases} \quad (17)$$

In words, $D_{I}(i) = 1$ if and only if $i$ does not violate the
extended specification.

A few theorems are necessary before showing how the extended specification is useful.

Theorem IV.1 (More information and more observation cannot harm safety for $Y$ or $Z$ states). If $Z$ state $z_1$ is safe, and $Z$ state $z_1'$ satisfies $I(z_1') \subseteq I(z_1)$ and $C(z_1') \supseteq C(z_1)$, then $z_1'$ is also safe. Similarly, if $Y$ state $y_1$ is safe, and $Y$ state $y_1'$ satisfies $y_1' \subseteq y_1$ and then $y_1'$ is also safe.

Corollary IV.2 (Observing more cannot harm safety). If control decision $C_1$ is safe in $Y$ state $y_1$, then so is any control decision $C_1' \supseteq C_1$.

Lemma IV.3 (Equality of projection when everything is observed is equivalent to equality of projection under all
controllers). For any two strings \( s_1, s_2 \in E^* \) and starting Z state \( z \), \( P(s_1) = P(s_2) \iff \forall C, P_C(z, s_1) = P_C(z, s_2) \).

**Theorem IV.4** (Safety is equivalent to satisfying the extended specification for Z states). For any Z state \( z \), we have \( D_Z(z) = D^*_f(I(z)) \). That is, a Z state is safe if and only if it satisfies the extended specification.

Note that this immediately implies that the safety of a Z state is solely dependent on its information state component, and not on its associated control decision.

**Theorem IV.5** (Safety is equivalent to satisfying the extended specification for Y states). For any Y state \( y \), we have \( D_Y(y) = D^*_f(y) \). That is, a Y state is safe if and only if it satisfies the extended specification.

These last two theorems have important implications. Without making use of the extended specification, determining the safety of any control decision is no simple matter. The structure of the MPO itself provides the only obvious means for attempting this, which is to search through the space of Y and Z states until all remaining states that can be reached are either known to be unsafe (if \( D_Y(\cdot) = 0 \)), or have already been visited (i.e., the search has run into a loop). Indeed, this was the algorithm we were using before adopting the state disambiguation approach and it can take exponential time in the size of \( G \), every time we want to determine the safety of a control decision. On the basis of previous results, the extended specification approach allows us to circumvent this problem by simply replacing this whole search by a single subroutine called each time we would like to determine the safety of a particular information state. At any Y state \( y \), we can determine whether or not it is safe to take control decision \( C(y) \) through two steps: first, we compute \( z = h_{YZ}(y, C(y)) \) and, second, we verify whether \( I(z) \) satisfies the extended specification. The first step takes is accomplished through a depth first search over \( G_K^+ \) and therefore takes time \( O(K |X| \cdot |E|) \) whereas the second step takes time proportional to the size of the extended specification, which is \( O(K |X|^2) \). Thus, the total running time is \( O(K |X| (|X| + |E|)) \).

The basic outline of an implemented algorithm for constructing the MPO is shown in Algorithm DoDFS. The algorithm simply performs a depth-first search (DFS). The parameter \( G \) represents the finite-state automaton, the parameter \( y \) is a Y state, and the parameter \( E \) contains the set of events \( E_o \) (events \( e \) in the algorithms), \( E_s \) (events \( e \) in the algorithms), \( E_{eo} \) and \( E_f \). The algorithm DoDFS searches through the space of Y states and, for each encountered Y state \( y \), finds the safe control decisions. Finding the safe control decisions is done in lines 2-10. This is accomplished by considering each subset of events \( \{ e \} \subseteq E_{eo} \), and determining whether it is safe to choose to monitor only the events \( e \cup E_{eo} \). This determination is made by a call to \( \text{Del} \) (which simply computes the value of \( D^*_f(I(h_{YZ}(y, e, E_{eo}))) \)). If the control decision is safe, it is added to the list \( sl \) (state list) and \( y \) is marked as safe. Traversing the space of Y states is done on lines 11-19. This is accomplished by considering all safe control decisions of the current Y state, determining the next Y state and, for each such Z state, computing all possible successor Y states and making a recursive call. Since there are a finite number of augmented states with count at most \( K + 1 \), there are a finite number of information states that will be traversed and the algorithm must eventually terminate. The initial call to the algorithm (not shown here) is: DoDFS(\( G, y_0, K, \emptyset, E \)).

**Proposition IV.1** (Running time of DoDFS). The running time of algorithm DoDFS is in \( O(2^{(K+3)\cdot |X|} |E|^2 |K+3| |X|) \).

If a fault event occurs multiple times, it needs to be tracked (i.e., included in the information state) once for each potential occurrence. If there are multiple different faults to diagnose, it is actually preferable to define multiple different MPOs, one for each fault, rather than defining a single one that diagnoses all faults, since the latter method would require keeping a separate count for each type of fault event, which would dramatically increase the size of the state space. If the MPO is found to be in the F state, then the system designer could potentially reset the system and simultaneously reset the MPO.

An example of the MPO is useful at this point:

**Example IV.1** (A simple example). Consider the automaton on the left side of Fig. 1. If we take \( K = 1 \), the specification \( T_{\text{spec}} \) is given by \( T_{\text{spec}} = \{(u, v) \in X^+ \times X^+ : N(u) = -1 \lor N(v) = K + 1 = 2 \} \). If we use the first method for computing \( T^*_{\text{spec}} \), that is, the method in which we construct \( G_K^+ \), then we obtain the following extended specification:

\[
T^*_{\text{spec}} = \{(0, -1), (5, 2), (1, -1), (5, 2), (2, -1), (5, 2), (0, 1), (4, 1)\}
\]

Note that this \( T^*_{\text{spec}} \) is much smaller than the original \( T_{\text{spec}} \). If we had computed \( T^*_{\text{spec}} \) using the on-line algorithm, the result

```
1: procedure DoDFS(G, y, Tespec, sl, E)
2: for all el \( \subseteq E_{eo} \) do \( \triangleright \) Try all subsets of events
3: \( ur \leftarrow \text{GetUR}(y, E_{eo} \cup el) \) \( \triangleright \) Unobservable reach
4: if \( \text{Del}(ur, Tespec) = \text{true} \) then
5: \( \text{Add} (y, E_{eo} \cup el) \) to sl \( \triangleright \) Control decision
6: end if
7: end for
8: if y is not marked “safe” then
9: Mark y as “unsafe”
10: end if
11: for all el \( \subseteq es \) s.t. \( y, E_{eo} \cup el \) do \( \triangleright \) Try all safe control decisions
12: \( ur \leftarrow \text{GetUR}(y, E_{eo} \cup el) \) \( \triangleright \) Unobservable reach
13: for all e \( \in E_{eo} \cup el \) \( \triangleright \) Try all events
14: \( next \leftarrow \text{Next}(ur, e) \) \( \triangleright \) Get next y state
15: if next not marked then
16: DoDFS(G, next, Tespec, sl, E)
17: end if
18: end for
19: end for
20: end procedure
```
would have been at least as large as $T_{\text{spec}}$ (and somewhat larger, in fact). Since $((0, -1), (4, 1)) \in T_{\text{spec}}$, event $b$ must be monitored initially. If $C(y_0) = \{b\}$ only, then $z_0 = h_{Y,Z}(y_0, \{b\}) = (((0, -1), (1, -1), (3, 0)), \{b\})$. Upon the occurrence of event $b$, we obtain $y_1 = h_{Y,Z}(z_0, b) = \{(2, -1), (4, 1)\}$. Since $((2, -1), (5, 2)) \in T_{\text{spec}}$, it is now necessary to monitor event $a$. Thus, $z_1 = h_{Y,Z}(y_1, \{a\}) = \{(2, -1), (4, 1)\}$ and if the system is truly in state $(2, -1)$, then no observable event will ever occur and we will remain perpetually uncertain as to whether or not a fault event occurred. This is not a problem, because we do know that, if a fault occurred, only one event has occurred since. On the other hand, if we do observe event $a$, then we obtain $y_2 = h_{Y,Z}(z_0, a) = (5, 2)$, and we can diagnose the fault. Now, return to the very first control decision. If we choose to monitor both events $a$ and $b$, then we obtain $z_0 = h_{Y,Z}(y_0, \{a, b\}) = (((0, -1), (3, 0)), \{a, b\})$. If event $b$ occurs, then we obtain $y_1 = h_{Y,Z}(z_0, b) = \{(4, 1)\}$ and we can immediately diagnose the fault. On the other hand, if event $a$ occurs, then we obtain $y_1 = h_{Y,Z}(z_0, a) = \{(1, -1)\}$, and all further control decisions are irrelevant since we know that no fault has occurred in the past or can occur in the future. Interestingly, this means that, by choosing to observe more initially, we can actually achieve 0-diagnosability. The MPO obtained is shown on the right side of Fig. 1. In the diagram of the MPO, rectangular states correspond to $Y$ states and oval states correspond to $Z$ states. This is the same notation used in [5].

V. CONCLUSION

We considered the problem of finding optimal dynamic sensor activation policies that enforce the $K$-diagnosability property. We defined the most permissive observer (MPO), whose structure contains all the solutions to the problem. We then proceeded to formulate the $K$-diagnosability property in terms of the state disambiguation problem, and proved the central result of this paper, namely that there exists a simple test for determining safe control decisions at each point in the system’s execution. We then translated this result into an efficient algorithm for constructing the MPO, which can be done on-line. That is, although the entire MPO is exponential in size, we showed that the safety of individual control decisions can be efficiently determined at each information state. In future work, we will investigate improving the efficiency of our algorithms, and work on the problem of finding a single optimal controller, using the MPO as a basis.

ACKNOWLEDGMENT

The work of the authors is supported in part by NSF grant CNS-0930081 and by a fellowship from Fonds FQRNT, Government of Québec, Canada.

REFERENCES