A Learning-based Active Fault-tolerant Control Framework of Discrete-event Systems

Jin Dai, Ali Karimoddini and Hai Lin

Abstract—A fault-tolerant controller is a controller that drives the plant to satisfy control specifications under both nominal operation and after the occurrence of faults. This paper introduces a framework that ensures the fault-tolerance of a discrete-event system (DES) plant, which includes: (i) a nominal supervisor with respect to the nominal plant to ensure the control specification that is prior to the occurrence of a fault, and this supervisor is continued to be applied to enforce a safety specification even after a fault until its detection; (ii) a fault diagnoser such that a fault can always be detected within a bounded delay of its occurrence; (iii) a post-fault supervisor to eventually satisfy, possibly with an elegant degradation, an online generated post-fault control specification. Moreover, to deal with unmodeled plant dynamics and environment uncertainties, we adopt an $L^*$-learning based approach to the design of both the diagnoser and the supervisors. An illustrative example is provided to detail the control framework introduced in this paper.

I. INTRODUCTION

Modern ubiquitous engineering systems have become more and more complex in architecture, and hence become more vulnerable to faults and failures that can cause unpredictable and undesired consequences. Therefore, how to detect, diagnose faults and how to take appropriate actions after the detection of these faults to maintain the systems’ performance become important issues. Motivated by its wide application range in modeling and analysis of complex systems [1][2], in this paper, we consider fault diagnosis and fault-tolerant control problems for discrete-event systems (DESs). The main objective of fault diagnosis and detection is to study methodologies for identifying and exactly characterizing possible faults arisen in the evolution of a system. Extensive studies have been devoted to fault diagnosis and detection of DESs. In [3], the authors investigated the diagnosability of DESs where faults are modeled as faulty events. Since then, several methodologies have been developed to solve the diagnosis problem of DES, see e.g. [4][5][6][7] and the references therein.

Following up a fault has been detected and diagnosed, a natural step is to reconfigure the control law so that to tolerate the fault. Fault-tolerance allows the system to continue its operation after the occurrence of a fault while guarantees post-fault performance objectives (possibly with elegant degradation) for the faulty system. Despite blooming developments in fault diagnosis and detection, relatively less effort has been made to the study of fault-tolerant control problem. The approach used in [8] involved supervisor switching upon occurrence of a fault. In [9], the authors designed supervisors that can tolerate sensor failures. The authors of [10] studied the sensor and actuator failure tolerance in multi-agent systems modeled as concurrent DESs. Fault-tolerance control using Petri nets was investigated in [11] to enforce the liveness with failures using system reconfigurations. Wen et al. [12][13] proposed a framework for the fault-tolerant control of DES where the fault-tolerance and recovery were based on the language equivalence and convergence by means of control. This approach, together with the results in [14], are deemed as “passive” since they aimed to design a unified supervisor to satisfy control specifications both in the nominal operation and after the occurrence of a fault. On the other hand, Paoli et al. [7][15] investigated an active fault-tolerance of controlled DES, and their approach incorporated a safe fault diagnosis with a post-fault supervisor reconfiguration. The fault-tolerant control to ensure a DES plant’s safety was studied in [16], and the authors proposed state-feedback based control scheme under both complete and partial observations. It is worth pointing out that both passive and active approaches aforementioned require a full prior knowledge of the DES plant model prior to as well as following the occurrence of faults; however, the assumption on the knowledge of a post-fault model might be unrealistic in some real world applications.

In this paper, we investigate the active fault-tolerance of DESs from a different angle. Starting from a generic model of the DES plant possessing both nominal and faulty behaviors, we want to design an architecture in which the supervisor actively reacts to the detection of a fault. Toward this end, we separate the nominal supervision from the post-fault control and propose a framework that involve a safe diagnoser and supervisor switching strategy so that: (i) a nominal supervisor working in the nominal mode to ensure the control specification that is prior to the occurrence of a fault, and this supervisor is continued to be applied even after a fault until its detection; (ii) a diagnoser can always detect before the system evolves any traces that violate the safety specification; (iii) a post-fault supervisor is used to eventually satisfy an online generated post-fault control specification. We derive necessary and sufficient conditions for the existence of appropriate post-fault supervisors that attain fault-tolerance. Furthermore, our proposed framework assumes only prior knowledge of the non-faulty part of the uncontrolled plant, and to deal with uncertainty arisen in the
faulty behaviors we adopt an \( L^* \)-learning based approach to the design of both the safe diagnoser and the supervisors, in case even no prior knowledge of the plant is given.

This paper is organized as follows. Section II introduces basic notations and preliminaries in supervisory control and fault diagnosis, based on which the active fault-tolerant control problem is formulated. Section III provides an algorithm based on \( L^* \) learning algorithm proposed by Angluin [17] to design the diagnoser; while Section IV investigates the learning-based supervisor synthesis and reconfiguration problems, and a second learning-based synthesis procedure is proposed. Section V gives a simple example to illustrate the proposed diagnosis-control framework. In Section VI, a conclusion is presented.

II. PRELIMINARIES AND PROBLEM FORMULATION

Following the supervisory control theory of discrete event systems [1][18], the uncontrolled system (a.k.a., a plant) is modeled by a deterministic finite automaton (DFA) \( G = (Q, \Sigma, q_0, \delta, Q_m) \), where \( Q \) and \( \Sigma \) are the finite state and event sets, respectively; \( \delta : Q \times \Sigma \rightarrow Q \) is the (partial) transition function, \( q_0 \) is the initial state, and \( Q_m \subseteq Q \) is the set of marked states. Let \( \Sigma^* \) denote all the finite-length traces over \( \Sigma \) plus the zero-length trace \( \epsilon \), \( \delta \) can then be extended to \( \delta : Q \times \Sigma^* \rightarrow Q \) in a natural way. A language is a subset of \( \Sigma^* \). The language generated by \( G \) is given by \( L(G) = \{s \in \Sigma^* | \exists q_0 \in Q_0, \delta(q_0, s) \} \) where \( \delta(q_0, s) \) means that the transition \( \delta(q_0, s) \) is defined. The marked language of \( G \) is defined as \( L_m(G) = \{s \in L(G) | \delta(q_0, s) \in Q_m \} \).

Given two DFAs \( G_1 = (Q_1, \Sigma, q_{01}, \delta_1, Q_{m1}) \) and \( G_2 = (Q_2, \Sigma, q_{02}, \delta_2, Q_{m2}) \), \( G_1 \) is said to be a subautomaton of \( G_2 \), denoted as \( G_1 \subseteq G_2 \), if there exists an injective map \( h : Q_1 \rightarrow Q_2 \) such that \( \forall s \in L(G_1), h(\delta_1(q_{01}, s)) = \delta_2(q_{02}, s) \).

For traces \( s, t \in \Sigma^* \), we use \( s \leq t \) to denote that \( s \) is a prefix of \( t \) and \( s < t \) to denote that \( s \) is a proper prefix of \( t \). For a language \( K \subseteq \Sigma^* \), \( pr(K) = \{s \in \Sigma^* | \exists t \in K : s \leq t \} \) denotes the prefix-closure of \( K \), and \( K \) is said to be prefix-closed if \( pr(K) = K \). We use the notion \( K \setminus t = \{s \in \Sigma^* | ts \in K \} \) to denote the set of traces that occur in language \( K \) after the occurrence of the trace \( t \).

For control purposes, the event set \( \Sigma \) of \( G \) is partitioned into set of controllable events \( \Sigma_c \) and uncontrollable events \( \Sigma_{uc} \). A language \( K \) is said to be controllable (with respect to \( G \) and \( \Sigma_{uc} \)) if \( pr(K) \subseteq L(G) \subseteq pr(K) \). Furthermore, we assume that \( \Sigma \) is also partitioned as \( \Sigma = \Sigma_o \cup \Sigma_{uo} \), where \( \Sigma_o \) represents the set of observable event and \( \Sigma_{uo} \) represents the set of unobservable events. We associate with \( \Sigma_o \) the natural projection \( P_o : \Sigma^* \rightarrow \Sigma_o \). A language \( K \) is said to be observable (with respect to \( P_o \) and \( G \)) if \( \forall s, t \in pr(K), s \in \Sigma : P_o(s) = P_o(t), s \sigma \in pr(K), t \sigma \in L(G) \Rightarrow t \sigma \in pr(K) \). Language \( K \) is said to be normal if \( P_o^{-1}(pr(K)) \cap L(G) \subseteq pr(K) \). A supervisor \( S \) is another finite automaton that runs in parallel with \( G \), and the closed-loop system is obtained through the parallel composition of \( S \) and \( G \), denoted as \( S || G \) [2]. A supervisor \( S \) is said to be:

- nonmarking if \( L_m(S || G) = L(S || G) \cap L_m(G) \);
- \( \Sigma_{uc} \)-enabling if it does not disable any uncontrollable event; and
- \( P_o \)-compatible if the controls following the indistinguishable traces are identical. It has been shown that given a nonempty and prefix-closed language \( K \subseteq L(G) \), there exists a \( \Sigma_{uc} \)-enabling, \( P_o \)-compatible and non-marking supervisor \( S \) such that \( L(S || G) = K \) if and only if \( K \) is controllable and observable [19]. Under the case where \( K \) is not controllable, if \( \Sigma = \Sigma_o \), there exists a supervisor \( S \) such that \( L(S || G) \) achieves the supremal controllable sublanguage [2] \( sup(C(K)) \); otherwise, there exists \( S \) such that \( L(S || G) \) achieves the supremal controllable and normal sublanguage \( sup(CN(K)) \) of \( K \).

To pursue such an active fault-tolerance, potential faults of the system are considered at this point. Let \( G^N = (Q^N, \Sigma, q_0^N, \delta^N, Q_m^N) \subseteq G \) denote the non-faulty part of \( G \). We assume that there does not exist in \( G \) any cycle of unobservable events. Instead of the nominal model \( G^N \). The non-fault control specification is given as a non-empty prefix-closed sublanguage \( K^N \subseteq L(G^N) \). We assume without loss of generality that \( K^N \) is controllable with respect to \( G^N \) and \( \Sigma_{uc} \), and a nominal supervisor \( S^N \) is expected to be constructed such that \( G_{cl} = S^N || G^N \) satisfies \( K^N \). Since \( G \) embeds the (potential) faulty behaviors of the system, the actual controlled behavior of the system is characterized by \( G_{cl}^N = S^N || G^N \). Let \( f_i \subseteq \Sigma_{uc} \cap \Sigma_{uo} \) denote the set of fault events. We write \( \Psi(f_i) = \{s_{f_i} \in L(G)\} \).

For the sake of simplicity, in the following of this paper, we consider a single fault event \( f \); clearly, \( L(G_{cl}^N) \cap \Psi(f) = \emptyset \).

In the following of this paper, by “faults” we mean a trace in \( L(G_{cl}^N) \) that contains \( f \) or violates \( K^N \). The system under fault-tolerant control should not execute “unsafe” behaviors regardless of faults. Safe behaviors include all of non-faulty behaviors and some of the tolerable post-fault behaviors and are captured by prefix-closed sublanguage \( K^S \subseteq L(G) \), and we assume that \( K^N \subseteq K^S \). Therefore it is clear that \( K^N \subseteq K^S \cap L(G_{cl}^N) \).

Fig. 1: Active fault-tolerant control framework.

In this paper, we propose a framework shown in Fig. 1 to enforce active fault-tolerance. The framework incorporates a fault diagnoser with the nominal and post-fault supervisors, and aims to solve the following three problems.

1) Safe nominal supervision Given the controllable nominal specification \( K^N \) and the safety constraint such that \( K^N \subseteq K^S \cap L(G_{cl}^N) \), obtain a nominal supervisor \( S^N \) such that: \( L(G_{cl}^N) = K^N \) and \( K^N \subseteq L(G_{cl}^N) \subseteq K^S \).
2) **Fault diagnosis** A diagnoser $G_d$ is designed to monitor behaviors of $G^N_{cl}$ through the observation projection $P_o$; once a fault occurs, the diagnoser should accomplish diagnosis before the system generates any traces violating $K^S$.

3) **Control-reconfiguration** Once a fault is detected and diagnosed by $G_d$ after a trace $s \in L(G^N_{cl})$ is executed, we stop the operation of the nominal supervisor $S^N$, generate online a post-fault specification $K^F = K^F(s)$ and reconfigure the supervisor $S$ such that $L(S||G) \setminus s = K^F \setminus s$.

The relationship between languages of $G^N$, $G_{cl}$, $G^N_{cl}$ and $S||G$ are depicted in Fig. 2.

![Diagram](image)

**Fig. 2:** Specifications for active fault-tolerance of supervised DES.

### III. LEARNING-BASED DESIGN OF FAULT DIAGNOSER

We investigate the fault diagnosis problem of the plant $G^N_{cl}$ in this section provided that a nominal supervisor $S^N$ is synthesized such that $L(S^N||G^N) \subseteq \sup C(K^N)$ (cf. Section IV for details). The diagnosis problem is to detect, isolate, and identify the faults from the observed behaviors of $P_o(L(G^N_{cl}))$. Through this paper, however, we assume that the closed-loop plant $G^N_{cl}$ is not known and only observable events of the system can be seen. The approach is to develop a diagnoser that can actively learn the faulty situations in the system. For this purpose, we adopt the $L^*$ algorithm [17] to effectively and actively learn the DES dynamics of the diagnoser, which can be viewed as an observer that is described by regular language $P_o(L(G^N_{cl}))$ with certain “labels”. Constructing the diagnoser $G_d$ relies on asking minimum queries to the oracle in $L^*$ who answers correctly two types of questions:

- **Membership queries:** in which the algorithm asks whether a string $s \in \Sigma^*$ belongs to $P_o(L(G^N_{cl}))$.
- **Conjecture queries:** in which the algorithm asks whether $L(G_d) = P_o(L(G^N_{cl}))$. If not, the oracle returns the string $s$ as a counterexample.

With these queries, each time, the algorithm acquires information about a finite collection of strings over $\Sigma_o$, and classifies them as either members or non-members of $P_o(L(G^N_{cl}))$. Then, this information will be used to create a series of observation tables to incrementally record and maintain the information whether strings in $\Sigma^*$ belong to $P_o(L(G^N_{cl}))$. The observation table $(S, E, T)$ is a 3-tuple where $S \subseteq \Sigma^*$ is an non-empty finite set of prefix-closed set of traces, $E \subseteq \Sigma^*$ is a non-empty set of suffix-closed set of traces, and $T^*$ is the membership oracle to be defined. The i-th observation table $T_i$ can be depicted as a 2-dimensional array whose rows are labeled by strings $s \in S \cup S \Sigma_o$, and whose columns are labeled by symbols $\sigma \in E$. The membership oracle, $T$, is a function from $\{s, s \in S \cup S \Sigma_o, \sigma \in E\}$ to the label set $\Sigma_L = \{0, Y, N, U\}$. $T(s\sigma) = 0$ if $s \sigma \notin P_o(L(G^N_{cl})); T(s\sigma) = Y$ if $s \sigma \in P_o(L(G^N_{cl}))$ is a faulty trace, i.e., for all $t \in S \Sigma^*$ such that $P_o(t) = P_o(s\sigma)$, $t \in L(G) - K^N$; $T(s\sigma) = N$ if $s \sigma \in P_o(L(G^N_{cl}))$ is not faulty; and finally $T(s\sigma) = U$ if $s \sigma \in P_o(L(G^N_{cl}))$ is undecided whether it is faulty or not, i.e., there exist $t, t' \in S \Sigma^*$ such that $P_o(t) = P_o(t') = P_o(s\sigma)$, $t \in L(G) - K^N$ but $t' \in K^N$. The row function $row : (S \cup S \Sigma_o)E \rightarrow \Sigma_L^E$ denotes the table entries in rows.

**Algorithm 1** $L^*$ algorithm for construction of a diagnoser.

**Input:** $K^N$, $\Sigma$ and $\Sigma_o \subseteq \Sigma$, membership queries $T$ for $t \in \Sigma^*$ and a counterexample teacher.

**Output:** $G_d$ such that $L(G_d) = P_o(L(G^N_{cl}))$ and with appropriate state labels.

1: Set $S = \epsilon$ and $E = \epsilon$.
2: Use the membership oracle to form the initial observation table $T_i(S, E, T)$ where $i = 1$.
3: while $T_i(S, E, T)$ is not completed do
4: if $T_i$ is not consistent then
5: find $s_1, s_2 \in S, \sigma \in \Sigma_o$ and $e \in E$ such that $row(s_1) = row(s_2)$ but $T_i(s_1\sigma e) \neq T_i(s_2\sigma e)$; 
6: Add $\sigma e$ to $E$;
7: Update $T_i$ for $(S \cup S \Sigma)E$ using membership queries.
8: end if
9: if $T_i$ is not closed then
10: find $s_1 \in S, \sigma \in \Sigma_o$ such that $row(s_1\sigma)$ is different from $row(s)$ for all $s \in S$;
11: Add $s_1\sigma e$ to $S$;
12: Update $T_i$ for $(S \cup S \Sigma_o)E$ using membership queries.
13: end if
14: end while
15: Once $T_i$ is completed, let $G_{di} = G_d(T_i)$ as the conjectured diagnoser.
16: if the counterexample oracle declares that the conjecture to be false and a counterexample $cex \in \Sigma^*$ is generated then
17: Add $pr(cex)$ into $S$;
18: update $T_i$ to $T_{i+1}$ by using the counterexample $cex$;
19: end if
20: Set $i = i + 1$ and return to while.
21: if $\exists n \in \mathbb{N}$ such that $G_{dn+1} = G_{dn}$, then
22: Return $G_{dn}$.
23: end if

An observation table is complete if it is both closed and consistent. The closedness requires that for all $t \in S \Sigma$, there exists an $s \in S$ such that $row(s) = row(t)$. If a table is
not closed, it means that there exists a trace \( s \sigma \in (\Sigma S - S) \) such that \( \text{row}(s \sigma) \) is different from \( \text{row}(s) \) for all \( s \in S \). Therefore, to make the table closed, \( s \sigma \) and its associated row should be added to the rows in \( S \). The consistency however requires that if for any two strings \( s_1, s_2 \in S \) with \( \text{row} (s_1) = \text{row} (s_2) \), then for all \( s \in \Sigma_o \), \( \text{row} (s_1 s) = \text{row} (s_2 s) \). If the table is not consistent, then there exist two strings \( s_1, s_2 \in S \) with \( \text{row} (s_1) = \text{row} (s_2) \), and \( s \in \Sigma_o \), and \( e \in E \), such that \( T(s_1 s e) \neq T(s_2 s e) \). In this case, to make the table consistent, it is sufficient to add \( s \sigma \) to \( E \).

After completing the observation table \( T_i \), we can construct a DFA, \( G_d (T_i) = \text{CoAc} (X_d, \Sigma_o, \delta_d, x_{d,0}, X_{d,m}, \) where \( X_d = \{ \text{row}(s) : s \in S \} \), \( x_{d,0} = \text{row}(\varepsilon) \), \( X_{d,m} = \{ \text{row}(s) : (s \in S) \wedge T(s) \neq \emptyset \} \), \( \delta (\text{row}(s), \sigma) = \text{row}(s \sigma) \), and \( \text{CoAc} \) is an operator that removes the states from which there does not exist a path to marked states. Associated with each state of the resulting DFA, a label can be defined \( L(s) = T(\varepsilon s) \). Then we use the obtained automaton, \( G_d (T_i) \), as the diagnoser for the plant \( L \). This diagnoser keeps running until a counter-example, \( cex \in (P_o (L(G_{cl}^N)) - L(G_d(T_i))) \cup (L(G_d(T_i)) - P_o (L(G_{cl}^N))) \), is detected by the membership oracle. In this case, \( cex \) and all its prefixes will be added into \( S \). Then, the table should be updated for the newly added elements. The whole algorithm is detailed in Algorithm 1. The \( L^* \) algorithm is guaranteed to construct a minimal DFA with a computational time bounded by a polynomial in \( |X_d| \) and the number of events contained in the longest counterexample provided by the oracle when answering equivalence queries.

Recall the diagnosability of \( K^N \) with respect to \( L(G_{cl}^N) \) and \( P_o \) [3][6]:

**Definition 1**: The prefix-closed nominal specification language \( K^N \subseteq L(G_{cl}^N) \) and \( P_o \) if

\[
(\exists n \in \mathbb{N})(\forall s \in L(G_{cl}^N) - K^N)(\forall t \in L(G_{cl}^N) \setminus s, |t| \geq n \text{ or } st \text{ deadlocks}) \Rightarrow (\forall u \in P_o^{-1}P_o(st) \cap L(G_{cl}^N), u \in L(G_{cl}^N) - K^N)
\]

Definition 1 states that \( K^N \) is diagnosable if it is possible to detect with a finite delay occurrences of faults of any type using the record of observed events in \( P_o (L(G_{cl}^N)) \). In this section, we consider the case that \( K^N \) as well as all its sublanguages are diagnosable with respect to \( L(G_{cl}^N) \) and \( P_o \). The following theorem characterizes the conditions of diagnosability via the diagnoser.

**Theorem 1**: [3]The prefix-closed language \( K^N \) is diagnosable with respect to \( L(G_{cl}^N) \) and \( P_o \) if and only if its diagnoser \( G_d \) satisfies the following condition: there are no fault-indeterminate cycles\(^1\) in \( G_d \) that consists of undecidable states.

The following simple example illustrates Algorithm 1.

**Example 1**: Consider the closed-loop plant \( G_{cl}^N \).

\[
\begin{array}{cccc}
    & e & f & a,d & b \\
    c & & & & \\
\end{array}
\]

\(^1\)For indeterminate cycles, readers are referred to [3][7] for details.

### IV. Learning-based Supervisor Synthesis and Reconfiguration

The problem of automatic supervisor synthesis and fault-tolerance is considered in this section. Given \( K^N \) controllable with respect to \( G^N \) and \( \Sigma_{ac} \), one can trivially synthesize a supervisor \( S^N \) such that \( L(S^N || G^N) = K^N \). However, under supervision of \( S^N \), whether or not \( L(S^N || G) \subseteq K^S \) is undecidable when prior knowledge of \( G \) is unaccessible. In this section, we first propose necessary and sufficient conditions for the existence of a safe nominal supervisor; then, we propose a modified \( L^* \) algorithm to synthesize a nominal supervisor with safety constraint \( K^S \); finally, the control reconfiguration problem is studied.

#### A. Existence of the Safe Nominal Supervisor

We consider the design of nominal supervisor under safety constraint \( K^S \), which requires that \( K^N \subseteq L(G_{cl}^N) \subseteq K^S \) in addition to \( L(G_{cl}^N) = K^S \). Define

\[
K^I = \{ s \in K^S - K^N | P_o^{-1}P_o(s) \cap L(G^N) \cap K^S \neq \emptyset \}
\]

as the set of faulty traces in \( K^S \) that are indistinguishable from non-faulty traces of \( K^S \), for which the control shall not reconfigure.

The following theorem gives necessary and sufficient conditions for the existence of a safe nominal supervisor.

**Theorem 2**: Given \( G^N \) and \( G \) of non-faulty and overall behaviors of the plant, respectively, for a prefix-closed nominal specification \( K^N \subseteq L(G^N) \) and a prefix-closed safety specification \( K^S \subseteq L(G) \), a nominal supervisor \( S^N \) such that

1. \( L(G_{cl}^N) = K^N \)
2. \( K^I \cup K^N \subseteq L(G_{cl}^N) \subseteq K^S \)

exists if and only if

1. \( \inf O_L(G)(K^I \cup K^N) \subseteq \sup C_L(G)(K^S) \)
2. \( \inf C_O_L(G)(K^I \cup K^N) \cap L(G^N) = K^N \)
where \( \inf O_{L(G)}(\cdot) \), \( \inf_{L(G)} CO(\cdot) \) and \( \sup_{L(G)} O_{L(G)}(\cdot) \) denote operators for infimal observable superlanguage, infimal controllable and observable superlanguage and supremal controllable sublanguage, all with respect to \( L(G) \), respectively. 

\textbf{Proof} \Rightarrow \) First, Objective 2) 

\[ K^1 \cap K^N \subseteq L(G^N_{cl}) \subseteq K^S \]

is guaranteed when Condition 1) holds [20]. Next, since \( \inf CO_{L(G)}(K^1 \cup K^N) \) is controllable and observable with respect to \( L(G) \), there exists a supervisor \( S^N \) such that \( L(G^N_{cl}) = \inf CO_{L(G)}(K^1 \cup K^N) \), and under Condition 2) 

\[ L(G_{cl}) = L(G^N_{cl}) \cap L(G^N) = \inf CO_{L(G)}(K^1 \cap K^N) \cap L(G^N) = K^N \]

which fulfills Objective 1). \( \Leftarrow \) If a supervisor \( S^N \) satisfying Objectives 1) and 2) exists, then 

\[ K^1 \cap K^N \subseteq L(G^N_{cl}) \]

and \( L(G^N_{cl}) \) is a controllable and observable sublanguage of \( K^S \) containing \( K^N \), therefore 

\[ \inf O_{L(G)}(K^1 \cup K^N) \subseteq L(G^N_{cl}) \subseteq \sup C_{L(G)}(K^S) \]

which coincides with Condition 1). Furthermore, according to Objective 1), one can write that 

\[ K^N \subseteq (K^1 \cup K^N) \cap L(G^N) \subseteq \inf CO_{L(G)}(K^1 \cup K^N) \cap L(G^N) \subseteq L(G^N_{cl}) \cap L(G^N) = K^N \]

which indicates clearly that \( \inf CO_{L(G)}(K^1 \cup K^N) \cap L(G^N) = K^N \). 

Based on Theorem 2, we propose the following “modular” supervisory control approach to ensure the safety constraints:

- Design a supervisor \( S^N_1 \) such that \( L(S^N_1 || G^N) = K^N \);
- Design a second supervisor \( S^N \) with respect to \( G \) such that \( L(S^N_1 || G) = \Omega(K^S) \), where \( \Omega(K^S) \) is a controllable and observable sublanguage of \( \sup C_{L(G)}(K^S) \);
- Use \( S^N = S^N_1 || S^N_2 \) as the nominal supervisor.

Provided the accomplishment of \( S^N_2 \), we can write that 

\[ L(G^N_{cl}) = L(S^N_1 || S^N_2 || G) \subseteq L(S^N_2 || G) = \Omega(K^S) \subseteq K^S \]

and 

\[ L(G_{cl}) = L(S^N_1 || S^N_2 || G^N) = L(S^N_1 || S^N_2 || G) \cap L(S^N_1 || G^N) \]

\[ \subseteq \Omega(K^S) \cap K^N \subseteq K^N \]

which implies the correctness of the modular approach (since all sublanguages of \( K^N \) are assumed to be diagnosable).

\section{B. Learning-based Nominal Supervisor Synthesis}

1) Learning for Controllability: We proceed to the core challenge of obtaining \( S^N_2 \) such that \( L(S^N_2 || G) \subseteq K^S \) when \( G \) is not given \emph{a priori}. Even if with full knowledge of the partially observed DES \( G \), the synthesis of a supervisor to achieve \( \sup CN_{L(G)}(K^S) \) has been shown to be NP-hard [21]. Due to the computational complexity consideration, we aim to develop a new supervisor synthesis method. The proposed algorithm includes two steps: first, we synthesize a supervisor \( S^C \) such that \( L(S^C || G) = \sup C_{L(G)}(K^S) \), assuming that \( \Sigma_{\omega \omega} = \emptyset \) by using a modified \( L^* \) algorithm \( L^*_C \); next, we modify \( S^C \) to be \( S^S \) such that \( L(S^S || G) = \Omega(K^S) \), which is a prefix-closed, controllable and observable sublanguage of \( \sup C_{L(G)}(K^S) \) that contains \( \sup CN_{L(G)}(K^S) \).

The learning-based synthesis algorithm lays its foundation on illegal behaviors. A trace \( s \in L(G) \) is said to be illegal if \( s \notin K^S \). Different from conventional \( L^* \), \( L^*_C \) asks the membership oracle with conditional membership queries upon basis of the illegal behaviors. A trace \( st \in \Sigma^* \) is said to be uncontrollably illegal if \( s \in K^S \), \( st \notin K^S \) with \( t \in \Sigma_{uc} \). Let \( C \) denote the collection of uncontrollably illegal behaviors. Define 

\[ D_u(C) = \{ s \in L(G) : \exists t \in \Sigma_{uc} \text{ such that } st \in C \} \]

to represent the collection of the traces formed by discarding the uncontrollable suffixes of illegal behaviors in \( C \), and let \( C_i \) denote the set of uncontrollably illegal behaviors after the
i-th iteration, then if a new uncontrollably illegal behavior $s_i$ (a new counterexample) is presented by the oracle, we obtain $C_{i+1} = \{s_i\} \cup C_i$.

The $L^*_C$ algorithm is then constructed on the basis of the operator $D_w(C)$. For $i \in \mathbb{N}$, if $t \in \Sigma^*$,

$$T^*_C(t) = \begin{cases} 0, & \text{if } t \notin K^S \\ 1, & \text{otherwise} \end{cases}$$

For $i \geq 1$

$$T^*_C(t) = \begin{cases} 0, & \text{if } T^*_{C_{i-1}}(t) = 0 \text{ or } t \in D_w(C_i) \\ 1, & \text{otherwise} \end{cases}$$

After the $i$-th iteration, the conjectured supervisor is constructed as $S^C_i = M(S, E, T)$ by following the procedure in Section III and is then presented to the oracle, if the oracle generates a counterexample $t \in (K^S - L(S^C_i)) \cup (L(S^C_i) - K^S)$, then we add $pr(t)$ to $S_i$ of the $i$-th observation table and construct a new observation table $T_{i+1}$. Details of algorithm $L^*_C$ are summarized in Algorithm 2.

**Algorithm 2** $L^*_C$

**Input:** $K^S$, $\Sigma$ and $\Sigma_o \subseteq \Sigma$, membership queries $T^c$ for $t \in \Sigma^*$ and a counterexample teacher.

**Output:** $S^C : L_m(S^C_i) = L(S^C) = \sup C_{L(G)}(K^S)$.

1. Set $S = \epsilon$ and $E = \epsilon$.
2. Form the initial observation table $T_i(S, E, T^c)$ where $i = 1$
3. while $T_i(S, E, T^c)$ is not completed do
4. Complete $T_i$ (similar to Algorithm 1, lines 3-14)
5. end while
6. Let $S^C_i = M(T_i)$ as the conjectured DFA that recognizes $\sup C_{L(G)}(K^S)$
7. if the counterexample oracle declares that the conjecture to be false and a counterexample (illegal behavior) $t \in \Sigma^*$ is generated then
8. Add $pr(t)$ into $S$;
9. update $T_i$ to $T_{i+1}$ by using the counterexample $t$ and $T^c_{i+1}$;
10. extend $T_i$ to $(S \cup \Sigma E)U$ using membership queries $T^*_{i+1}$ to the oracle;
11. end if
12. Set $i = i + 1$ and return to while until the oracle declares that the conjecture is true.
13. return $S^C = M_i$.

The correctness and convergence of $L^*_C$ are formally summarized as the following theorem.

**Theorem 3:** [22] Assume that the safety specification $K^S \subseteq L(G)$ is non-empty and prefix-closed, then $L^*_C$, with $T^c(t)$ converges to a supervisor $S^C$, such that $L(S^C) = \sup C_{L(G)}(K^S)$. Furthermore, this iteration procedure of synthesizing $S^C$ will be done in a finite number of counterexample tests.

2) **Construction of the Nominal Supervisor:** Next, we consider the construction of $S^N_2$. From $L^*_C$, we can conclude that $S^C = (Q_S, \Sigma_o, q_0, S)$ with $L(S^C) = \sup C_{L(G)}(K^S)$. For each $q \in Q_S$, we define

$$Act(q) = \{\sigma \in \Sigma | \delta_S(q, \sigma)!\}$$

as the set of active events at state $q$. For any subset $Q \subseteq Q_S$, the set of $\sigma$-reachable states from $Q$ with an observable event $\sigma \in \Sigma_o \cap (\cup_{q' \in Q} Act(q'))$ is defined as

$$R(Q, \sigma) = \{q \in Q_S | (\exists q' \in Q)q = \delta_S(q', \sigma)\}$$

and the unobservable reach is defined as

$$UR(Q) = \{q \in Q_S | (\exists q' \in Q)(\exists s \in \Sigma_o)q = \delta_S(q', s)\}$$

Define a DFA $S^N$ over the observable event set $\Sigma_o$ as follows: $S^N = \text{trim}(Q_N, \Sigma_o, \delta_N, Q_0, N)$, where

- $Q_N = 2^{Q_S}$.
- $Q_0 = \text{UR}(\{q_0, s\}) \subseteq Q_N$.
- For $\sigma \in \Sigma_o$ and $Q \subseteq Q_N$, $\delta_N(Q, \sigma) = \text{UR}(R(Q, \sigma))$.

Let $L(G_d) = \Omega(K^N)$ for the aforementioned supervisor $S^N_2$, the following theorem indicates that $\Omega(K^N)$ is less conservative than $\sup C_{L(G)}(K^S)$.

**Theorem 4:** $\sup C_{L(G)}(K^S) \subseteq \Omega(K^N) \subseteq L(S^C) = \sup C_{L(G)}(K^S)$. 

**Remark 1:** As shown in [17], $L^*_C$ guarantees to construct $S^C$ recognizing $\sup C_{L(G)}(K^S)$ with a computational time bounded by a polynomial in $|Q_N|$ and the number of events contained in the longest counterexample provided by the oracle when answering equivalence queries. The complexity of transforming full observation supervisor $S^C$ to partial supervisor $S^N_2$ is bounded by $O(|Q_N||Q_N|)$.

**C. Learning-based Active Fault-tolerance**

In Section III, a diagnoser $G_d$ is designed to monitor the behaviors of the closed-loop system $G^N_d$. When no faults occur, the supervised system is $S^N \| G^N$ and is guaranteed to satisfy $K^N_d$; when a fault occurs, the system continues its operation until the diagnoser detects the fault’s occurrence, which is enforced by $S^N$ to be accomplished before $G^N_d$ generates any traces violating $K^S$. After the fault detection, we stop the operation of the nominal supervisor $S^N$ and switches to a post-fault supervisor $S$ to attain a post-fault specification. This section investigates the problem of re-configuring the supervisor to achieve active fault-tolerance following the fault detection and diagnosis.

Assume that under the supervision of nominal supervisor $S^N$, a trace $s \in L(G^N_d)$ is generated when the fault $f$ is detected and diagnosed by the diagnoser $G_d$, then $s = ut$ where $u \in \Psi(f)$ and $t$ can be viewed as the diagnosis delay. Following the detection and diagnosis of the fault, the generation of appropriate post-fault specification $K^F = K^F(s)$ is considered at this point, in this paper, we require that

- $pr(s) \subseteq K^F(s)$.
- $K^F(s) \subseteq K^S(s)$.
- $K^F(s) \setminus s \subseteq K^S \setminus s$. 


The intuition of the above requirements basically asserts that the post-fault specification should always tolerate the behavior generated by the plant before the fault diagnosis, and moreover, the post-fault specification should also obey the safety constraints.

The following theorem states the solvability of the reconfiguration problem, whose proof is omitted due to space consideration.

**Theorem 5:** For any trace \( s = ut \in K^F \), \( u \in \Psi(f) \) and \( G_d(P_0(s)) = Y \), the reconfiguration problem is solvable if and only if

1) there does not exist a string \( v \in \Sigma_u \) such that \( sv \in L(G_d^N) \setminus K^F \).

2) there exists a sublanguage of \( K^F \) that is controllable and observable with respect to \( G_d^N \).

Theorem 5 basically states that, one can re-synthesize a post-fault supervisor \( S \) such that \( L(S) \| G = K^F \) if and only if any string that contains a fault event and leads to an undesired behavior can be interrupted by disabling at least one controllable event.

Note that for any trace \( s \in L(G_d^N) \), \( K^F(s) = pr(s) \cup (K^F(s) \setminus s) \), in this section, we only concentrate on the synthesis of \( K^F(s) \setminus s \), where \( s \) is of the form in Theorem 5. In fact, after stopping operation of \( S \), and building up an appropriate \( K^F \), we can following the procedure in the previous section to synthesize the post-fault supervisor to obtain \( \Omega(K^F \setminus s) \).

**V. AN ILLUSTRATIVE EXAMPLE**

We examine the effectiveness of the proposed fault-tolerant control framework by revisiting the example in [15]. Due to the limited space, we have to omit the details of the observation tables based on application of the proposed algorithms. Consider the hydraulic system \( G \) in Fig. 4, which is composed of a water tank \( T \), a pump \( P \), three valves \( V_1 \), \( V_2 \), and \( V_r \), and associated pipes. The pump is used to move fluid from the tank through the pipe and must coordinate with the valves, and the pipe is equipped with a pressure sensor for monitoring use. Valves \( V_1 \) and \( V_2 \) is equipped with a switch such that only one of them can be used at the same time. The nominal part of the system is depicted by \( G^N \).

![Fig. 4: The hydraulic system.](image)

The events of the system are defined as follows: for the valve \( V_i \) for \( i = 1, 2 \), events \( opi \) and \( cl_i \) are used to open and close valve \( V_i \); event \( rec \) is used to open the reserved valve \( V_r \); for the pump, event \( start \) is used to start the pump while \( stop \) is used to stop it; for the pressure sensor, \( P \) means an over-pressure in the pipe and \( NP \) means no over-pressure in the pipe. We assume that all these events are controllable and observable. The automaton representation of \( G^N \) is given in Fig. 5 (a).

In nominal operation, the pump is coordinating with either one of \( V_1 \) and \( V_2 \) (with the help of \( V_r \)). We now consider the fault \( f \) that valve \( V_1 \) may get stuck closed due to possible malfunction, and the automaton representing \( G \) is then given in Fig. 5 (b). Note that we omit events \( P \) and \( NP \) for space consideration. The nominal specification requires coordination between the pump and \( V_1 \), and is given by \( K^N = pr((op_1.start.stop.cl_1)^*) \). The safety specification requires that the pump should not be working with the closed \( V_1 \), i.e., \( L(G) - K^S = pr((op_1.start.stop.cl_1)^*).f.op1.start. \)

Since all the events except for \( f \) are controllable and observable, the nominal specification \( K^N \) is controllable and observable with respect to \( G^N \). We use Algorithm \( L_C \) to help build up the safe nominal specification, and the supervisor \( S^N \) as well as the closed-loop plant \( G^N_{cl} \) are depicted respectively in Fig. 6, in red solid lines.

![Fig. 6: The safe nominal supervisor \( S^N \).](image)

Now for closed-loop plant \( G^N_{cl} \), we apply Algorithm 1 and construct an online fault diagnoser \( G_d \) as shown in Fig. 7.

![Fig. 7: The fault diagnoser \( G_d \).](image)

From Fig. 7, the fault diagnoser detects the fault \( f \) by observing the trace \( s = pr((op_1.start.stop.cl_1)^*).pp \). When the fault \( f \) is diagnosed, we stop the operation of \( S^N \), and consider the generation of post-fault specification \( K^F \). In this example, we require that rather than the malfunctioned \( V_1 \), the pump should coordinate with the alternative valve \( V_2 \), whose operation recalls the reserved valve \( V_r \). Hence the “partial” post-fault specification is given by \( K^F \setminus s = pr(rec.(op_2.start.stop.cl_2)^*) \), which is obviously controllable and observable with \( G \), despite the fact that \( G \) is not
given *a priori*. The post-fault supervisor $S$ that drives the system to satisfy $R^F \setminus s$ after the trace $s$ is then given in Fig. 8.

![Fig. 8: The post-fault supervisor $S$.](image)

**VI. CONCLUSION**

In this paper, the active fault-tolerance control of discrete-event systems is studied. Based on the uncontrolled plant possessing both non-faulty and faulty behaviors, the contributions of this paper are as follows. (i) A learning-based approach is proposed to synthesize a nominal supervisor in an effort to satisfy the nominal specification when the plant model is not given *a priori*. (ii) We apply a second learning-based approach to design a safe diagnoser to monitor the system under nominal supervision, whose task is to detect and diagnose the fault before the system generates any traces violating the safety specification. (iii) Given the online generated post-fault specification, we propose a control reconfiguration framework that incorporates with the safe diagnoser to solve the active fault-tolerance problem. (iv) Effectiveness of our approach is illustrated through an example. Future study directions of this work may include examining the scalability of the proposed approach, and extending the current fault-tolerant control framework to a distributed/decentralized architecture.

**REFERENCES**