Fault Diagnosis of Hierarchical Discrete-Event Systems Based on State-Tree Structures

Deguang Wang, Xi Wang, *Member, IEEE*, Jing Yang, and Zhiwu Li, *Fellow, IEEE*

***Abstract*—In this paper, fault diagnosis of hierarchical discrete event systems (HDES) is investigated using state-tree structures (STS). As a structured formalism, an STS provides a compact representation for an HDES model and utilizes binary decision diagrams (BDDs) for efficient symbolic computation. Building on the above advantages of STS, with "on-the-fly" analysis technique addressed, the issue of offline diagnosability verification can be efficiently tackled. A symbolic approach of diagnoser construction is presented based on predicates and predicate transformers. With the BDD representation of predicates, the state space of the diagnoser is compressed and managed such that the occupied computer memory space is greatly reduced. Besides, instead of flattening an STS model to its equivalent monolithic model at first and then constructing the entire diagnoser for such a model, a heuristic on-the-fly algorithm based on the depth-first search, which unfolds the STS model gradually, is proposed for the level-by-level diagnosability analysis. Finally, several case studies are provided for evaluating the effectiveness and the scalability of the proposed method.**

***Index Terms*—Hierarchical discrete-event system, fault diagnosis, state-tree structure, level-by-level analysis, symbolic diagnoser.**

I. Introduction

Fault diagnosis plays a crucial role for guaranteeing reliable and safe operations of complex automated systems, such as manufacturing systems, transportation systems, and aerospace systems. Once a fault occurs, the system behavior deviates from its normal or intended operation. In recent years, diagnosis approaches in the framework of discrete-event systems (DES) [1]–[3] have been widely studied in the literature. By extracting the high level logical behavior of a system under diagnosis, a DES method provides diagnostic information using the discrete-state and event-driven model. Diagnosability analysis and online diagnosis are two main issues to be tackled. Online diagnosis aims to infer the occurrence of predetermined faults based on the observed event sequences, while diagnosability refers to the ability that provides a precise diagnosis verdict.

An industrial system usually consists of multiple subsystems that operate concurrently and each sub-system has different abstract levels of functionality at different levels of modelling, namely, an industrial system is naturally equipped with hierarchy and concurrency structures. The number of states of an industrial system grows exponentially with respect to the number of sub-systems. Therefore, the fault diagnosis of such a system is computationally difficult resulting in solutions that require large computer memory space. One way of mitigating complexity is to take advantage of the hierarchical structure without expanding it. In this work, the diagnosis problem of an enhanced class of DES, i.e., hierarchical discrete-event systems (HDES), is considered in the framework of state-tree structures (STS) [4]–[10]. Adapted from Statecharts [11], an STS can model an HDES in a compact way. Besides, based on predicates and binary decision diagrams (BDDs) [12], [13], a complete symbolic approach for efficient computation has been developed in STS, which lays a solid foundation for addressing the diagnosis problem of HDES.

A. Related Work

In the DES community, fault diagnosis has been one of the hot topics. Starting with the state-based formalism [14] and the event-based framework [15], a series of work has been published dealing with different fault scenarios. For efficient diagnosis, the abstraction-based, modular, distribute, decentralized, and symbolic approaches are proposed in [16]–[20], respectively. In [21]–[25], the diagnosis method is investigated in hierarchical, timed, stochastic, and fuzzy DES settings, respectively. In [26], a state-based paradigm of asynchronous diagnosis and diagnosability under full observation is provided and extended to the case of partial observation in [27]. Besides, plenty of work [28]–[32] addressing diagnosis and diagnosability of DES using Petri nets ensues. In [33], a detailed review of the state-of-the-art fault diagnosis techniques and tools can be found.

A regular or flat DES is a special type of HDES, i.e., a regular DES is an HDES with one layer. Finite automata are used to model a flat DES, but it is infeasible for expressing the hierarchical decomposition of a complex system. Moreover, in a flat modelling scheme, adding new information will increase the computational burden dramatically and the search state space size will be definitely larger. The use of BDDs contributes to reduce the state space and tackle the computational complexity hurdle. Extended finite automata (EFA) are introduced in [34], which endow a compact and comprehensible representation for a DES model and give rise to EFA-based synthesis approaches utilizing BDDs [35]–[37]. The combination of EFA and BDDs overcomes the

This work was supported in part by the Alexander von Humboldt Foundation, Guizhou Provincial Science and Technology Projects under Grant No. Qiankehejichu [ZK[2022]Yiban103], Innovation group of Guizhou Education Department under Grant No. Qianjiaohe KY[2021]012, and the Science and Technology Fund of Guizhou University under Grant No. Guidateganghezi [2021]04. (*Corresponding author: Xi Wang*).

Deguang Wang is with the School of Electrical Engineering, Guizhou University, Guiyang 550025, China (e-mail: dgwang@gzu.edu.cn).

Xi Wang is with the School of Mechano-Electronic Engineering, Xidian University, Xi'an 710071, China. (e-mail: wangxi@xidian.edu.cn).

Jing Yang is with the School of Electrical Engineering, Guizhou University, Guiyang 550025, China (e-mail: jyang7@gzu.edu.cn).

Zhiwu Li is with the Institute of Systems Engineering, Macau University of Science and Technology, Taipa, Macau and also with the School of Mechano-Electronic Engineering, Xidian University, Xi'an 710071, China (e-mail: zhwli@xidian.edu.cn).

following drawbacks of the standard approach: 1) modeling complex systems by finite automata makes the model large and intractable; 2) exploring all reachable states explicitly is computationally expensive in terms of both time and memory. However, the BDD-based approaches in [34]–[37] mainly focus on the supervisory control problem. In addition, the EFA formalism cannot model hierarchy and concurrency structures of a complex system.

The aforementioned studies have the following two limitations:

1) The automata-based diagnosis approaches are not suitable for large-scale DES with hierarchical structures. Generally, the diagnosis relies on the global system model, where the state size is exponential with respect to the number of parallel and nested sub-systems. Although a modular method can get rid of a global model, several assumptions (e.g., shared events among components are observable) are essential. Moreover, a diagnosable failure may become undiagnosable by modular diagnosis.
2) In the most existing work, the states in a diagnoser are presented in the form of state subsets and explicitly stored in the computer memory. The state compression technology based on predicates and BDDs is rarely used. In [20], BDDs are utilized to compact and manage the state space of a diagnoser. However, encoding efficiency is limited, especially for a synchronous product system because of the lack of a hierarchical organization.

To the best of our knowledge, none of the existing results, in the literature relevant to fault diagnosis of HDES, combines a hierarchical modeling formalism, symbolic computation, and an on-the-fly analysis technique in the same approach: model an HDES by an STS, compress the state space by symbolic computation, and verify diagnosability hierarchically by a heuristic on-the-fly algorithm. In [38], fault diagnosis is investigated in the contexts of STS and timed STS, which follows the same idea as [21]. The difference is that in [21] a hierarchical finite state machine is used as the modelling tool. The proposed approaches in [21] and [38] adapt and extend the state-based diagnosis work in [26]. By taking the advantage of system structure, the computer memory space for fault diagnosis in [21] and [38] is reduced compared with that in [26]. However, such a reduction depends on several additional constraints (e.g., the boundary events of a holon are assumed to be observable). Besides, the advanced techniques, such as symbolic computation and on-the-fly analysis, are not involved.

B. Our Contribution

This paper studies the fault diagnosis problem of HDES modeled by STS. The main contributions are summarized as follows:

1) In the worst case, the diagnoser construction is subject to the exponential complexity with respect to the system's state size. To reduce the demanding of the diagnoser for the computer memory space, a symbolic approach in the framework of STS, which is based on predicates and BDDs, is proposed to encode and compress a diagnoser. With the help of the structural information embedded in an STS model, the encoding efficiency of the diagnoser is much higher than that using automata-based diagnosis approaches.
2) The diagnosability verification presented in [15] relies on the search of cycles in both the diagnoser and the system model. A systematic procedure in which only the diagnoser is used to check the condition violating the diagnosability is designed.
3) To avoid the complete expansion of an STS into its equivalent monolithic model at first and then the entire diagnoser construction for such a model, an efficient heuristic on-the-fly algorithm based on the depth-first search is developed to explore the state space of the diagnoser and check diagnosability simultaneously. By prioritizing the branches to be explored, diagnosability analysis is performed in a hierarchical way from top to bottom. In the case of non-diagnosability, the algorithm significantly speeds up the verification process.
4) The developed algorithms have been realized in a software package, STSLib, which can easily handle the diagnosis problem of industrial applications.

The remainder of this paper is structured as follows. Section II provides the preliminaries of fault diagnosis using automata, STS, and symbolic computation. Section III details the diagnoser construction of HDES using STS, which lays a solid foundation for diagnoser symbolication and diagnosability analysis in Section IV. Section V develops a heuristic on-the-fly symbolic algorithm for verifying diagnosability hierarchically. Section VI evaluates the proposed approach by several examples. Conclusions are drawn in Section VII.

II. Notations and Preliminaries

This section reviews the preliminaries of fault diagnosis using automata, state-tree structures (STS), and symbolic computation, summarized from [4], [9], [15], and [32].

A. Fault Diagnosis Using Automata

An automaton $\mathbf{P}$ for modeling a DES to be diagnosed is a five-tuple $\mathbf{P} = (Q, \Sigma, \delta, q_0, Q_m)$, where Q is the finite *state* set; Σ is the finite *event* set; δ is the *partial state transition function*; $q_0 \in Q$ is the *initial* state; and $Q_m \subseteq Q$ is the set of marker states. The event set is partitioned into the *observable* subset Σ_o and the *unobservable* subset Σ_{uo} with $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. A sequence of events in Σ is called a *string*. Σ^* is the set of all finite strings over Σ, including the *empty string* ϵ. We write $\delta(q, \sigma)!$ to mean that $\delta(q, \sigma)$ is defined with $q \in Q$ and $\sigma \in \Sigma$. The *closed behavior* of $\mathbf{P}$ is represented by $L(\mathbf{P}) = \{s \in \Sigma^* \mid \delta(q_0, s)!\}$.

The diagnosis of an observable fault is trivial; thus faults are generally represented as unobservable events. Except fault events, there may exist other unobservable events due to the lack of sensors. An *observer* $\mathbf{P}_o$ is an automaton that can be constructed using the approach proposed in [3]. Each state in $\mathbf{P}_o$ is an estimate (a state subset) of the true system state. A *diagnoser* $\mathbf{P}_d$ is an automaton built on $\mathbf{P}_o$ by tagging each state within the estimate in $\mathbf{P}_o$ with fault labels and/or normal label.

Let $\Sigma_f \subseteq \Sigma_{uo}$ denote the set of fault events. The set Σ_f can be partitioned into disjoint sets corresponding to different failure types $\Sigma_f = \Sigma_{f_1}\dot{\cup}\cdots\dot{\cup}\Sigma_{f_m}$. The notion of F_i-indeterminate $(1 \leq i \leq m)$ cycle helps propose the diagnosability condition. An F_i-indeterminate cycle in the diagnoser is one corresponds to two cycling traces in the original system with the same observation such that events in Σ_{f_i} appear in one trace but not in the other.

Theorem 1. *[15] A DES modeled by an automaton $\mathbf{P}$ is diagnosable w.r.t. Σ_f if and only if there is no F_i-indeterminate cycle in $\mathbf{P}_d$ for any fault type Σ_{f_i}.*

The above result relies on the assumption that no cycles of unobservable events exist in $\mathbf{P}$. The detailed proof can be found in [15].

B. System model

This subsection briefly introduces the STS model, summarized from [4] and [9].

State collection is defined to describe the structure of a state set. Let X be a finite collection of states, $x \in X$, and $Y = \{x_1, x_2, \ldots, x_n\}(n \geq 1)$ be a proper subset of X with $x \notin Y$. With the abuse of set operations, we follow the notations in [4]. If x is the disjoint union [1] (resp., Cartesian product [2]) of states in Y, write $x = \dot{\bigcup}_{x_i \in Y} x_i$ (resp., $x = \prod_{x_i \in Y} x_i$), x is called an OR (resp., AND) superstate and each x_i an OR (resp., AND) component of x. The states other than the superstates in X are called SIMPLE states. Superstates are the aggregation (or abstraction) of the SIMPLE states and represented by arc rectangular boxes graphically.

A *holon* H assigned to an OR superstate x is an automaton with both boundary and internal transitions, defined as a five-tuple $H^x = (X^x, \Sigma^x, \delta^x, X_0^x, X_m^x)$, where X^x is the nonempty state set, divided into the *external state set* X_E^x (possibly empty) and *internal state set* X_I^x with $X^x = X_E^x \dot{\cup} X_I^x$; Σ^x is the event set, structured as the disjoint union of *boundary event set* Σ_B^x and *internal event set* Σ_I^x, i.e., $\Sigma^x = \Sigma_B^x \dot{\cup} \Sigma_I^x$; $\delta^x : X^x \times \Sigma^x \rightarrow X^x$ is the partial transition function, composed of two disjoint transition structures, i.e., the *internal transition structure* $\delta_I^x : X_I^x \times \Sigma_I^x \rightarrow X_I^x$ and the *boundary transition structure* $\delta_B^x = \delta_{BI}^x \dot{\cup} \delta_{BO}^x$ with $\delta_{BI}^x : X_E^x \times \Sigma_B^x \rightarrow X_I^x$ (*incoming boundary transitions*) and $\delta_{BO}^x : X_I^x \times \Sigma_B^x \rightarrow X_E^x$ (*outgoing boundary transitions*); $X_0^x \subseteq X_I^x$ is the set of *initial states*; and $X_m^x \subseteq X_I^x$ is the set of *terminal states*.

The state space of DES is structured as a *state-tree* $\boldsymbol{ST}$ and it is a four-tuple $\boldsymbol{ST} = (X, x_0, \mathcal{T}, \mathcal{E})$, where X is a finite structured state set consisting of three kinds of states: AND, OR, and SIMPLE; $x_0 \in X$ is a special state called the *root state*; $\mathcal{T} : X \rightarrow \{\text{AND}, \text{OR}, \text{SIMPLE}\}$ is the *type function*; and $\mathcal{E} : X \rightarrow Pwr(X)$ is the *expansion function*, where $Pwr(X)$ is the power set of X. A *sub-state-tree*, acting as a state subset in an automaton, is the result of removing the branches (but not all) rooted by OR superstates in $\boldsymbol{ST}$. A *basic state-tree* is a special type of sub-state-trees with only one component at each OR superstate. The notations $\mathcal{ST}(\boldsymbol{ST})$ and $\mathcal{B}(\boldsymbol{ST})$ are used to denote the sets of all sub-state-trees and all basic state-trees of $\boldsymbol{ST}$, respectively.

With holons and state-trees defined, a *state-tree structure* $\mathbf{G}$ for modeling a DES is a six-tuple $\mathbf{G} = (\boldsymbol{ST}, \mathcal{H}, \Sigma, \Delta, \boldsymbol{ST}_0, \mathcal{ST}_m)$, where $\boldsymbol{ST}$ is the *state-tree*; $\mathcal{H}$ is the set of *holons*; $\Sigma = \Sigma_o \cup \Sigma_{uo}$ is the finite set of events; $\Delta : \mathcal{ST}(\boldsymbol{ST}) \times \Sigma \rightarrow \mathcal{ST}(\boldsymbol{ST})$ is the *global transition function*; $\boldsymbol{ST}_0$ is the *initial state-tree*; and $\mathcal{ST}_m$ is the *marker state-tree* set. Write $\Delta(b, s)!$ if $\Delta(b, s)$ is defined for $b \in \mathcal{B}(\boldsymbol{ST})$ and $s \in \Sigma^*$. For the diagnosis problem, the component $\mathcal{ST}_m$ in an STS $\mathbf{G}$ can be omitted since the diagnosis does not rely on it. For simplicity, we use a five-tuple $\mathbf{G} = (\boldsymbol{ST}, \mathcal{H}, \Sigma, \Delta, \boldsymbol{ST}_0)$ to represent an STS model of an HDES under diagnosis.

Both automaton and synchronous product models are special STS. An automaton model can be converted as an STS with one holon and its state space is organized as a state-tree by introducing an OR superstate as the root and assigning all states of the automaton as its children. A synchronous product model with n component automata can be converted into an STS by the following steps:

1) Introduce an OR superstate for each component automaton and assign all states of the automaton as its children.
2) Introduce an AND superstate as the root and assign all of the OR superstates created in step 1) as its children.

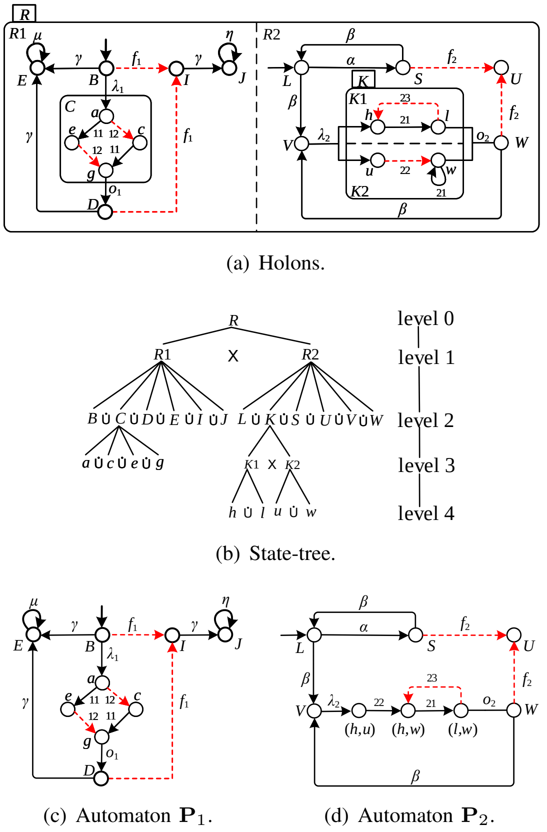
By exploding OR superstates and replacing AND superstates by the parallel composition of their components, an STS model can be converted into the equivalent flat automaton.

Example 1. *Consider an STS model $\mathbf{G}$ in Fig. 1, where the red dotted lines represent the transitions labeled by unobservable events. The sets of AND and OR superstates are $\{R, K\}$ and $\{R1, R2, B, K1, K2\}$, respectively. There are five holons H^{R1}, H^{R2}, H^B, H^{K1}, and H^{K2} matched to OR superstates. The equivalent flat automata $\mathbf{P}_1$ and $\mathbf{P}_2$ of holons H^{R1} and H^{R2} can be obtained respectively by the expansion of H^B and the parallel composition of H^{K1} and H^{K2}, as shown in Figs. 1(c) and 1(d). There are 72 states and 203 transitions in the equivalent flat automaton $\mathbf{P} = \mathbf{P}_1 || \mathbf{P}_2$ of $\mathbf{G}$, which is complicated and thus is not given. The sub-state-tree depicted in Fig. 2(a) is equivalent to the state subset $\{(c, (h, u))$, $(c, (h, w))$, $(e, (h, u))$, $(e, (h, w))\}$ in $\mathbf{P}$ and the basic state-tree depicted in Fig. 2(b) is equivalent to the state $(e, (h, w))$ in $\mathbf{P}$.*

One compact representation of a sub-state-tree is to use an *active state set* [4]. The size of an active state set is less than that of a sub-state-tree. Let $subST = (Y, x_0, \mathcal{T}', \mathcal{E}')$ be a sub-state-tree of $\mathbf{ST}$. Let $z \in Y$, $\mathcal{T}'(z) = \text{OR}$, and $x \in \mathcal{E}'(z)$. Then, x is said to be *active* if $\mathcal{E}'^*(x) = \mathcal{E}^*(x)$ and $\mathcal{E}'^*(z) \subset \mathcal{E}^*(z)$, i.e., all the descendants of x on $\mathbf{ST}$ are on the $subST$ but at least one descendant of z is not on the $subST$. Let $\mathcal{V}(z)$ be the set of all active states expanding z. Then, a proper sub-state-tree is uniquely represented by the active state set $\mathcal{V} = \bigcup_{\forall z \in Z} \mathcal{V}(z)$, where Z is the set of OR superstates having

[1] The semantics of x is the *exclusive-or* of x_i, $i = 1, 2, \ldots, n$, i.e., to be at state x the system must be at exactly one state of Y.

[2] The semantics of x is the *and* of x_i, $i = 1, 2, \ldots, n$, i.e., to be at state x the system must be at all states of Y simultaneously.



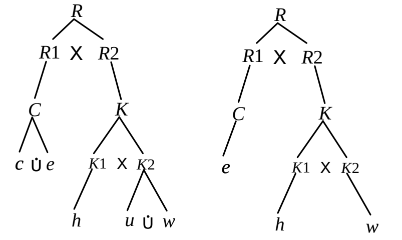
(a) Holons.

(b) State-tree.

(c) Automaton $\mathbf{P}_1$.

(d) Automaton $\mathbf{P}_2$.

Fig. 1: An STS $\mathbf{G}$ and flat automata $\mathbf{P}_1$ and $\mathbf{P}_2$.



(a) A sub-state-tree.

(b) A basic state-tree.

Fig. 2: Sub-state-trees.

active children. For example, the active state sets $\{c,e,h\}$ and $\{e,h,w\}$ represent the sub-state-tree in Fig. 2(a) and the basic state-tree in Fig. 2(b), respectively. In the rest of this paper, for convenience, a sub-state-tree is expressed by its corresponding active state set.

C. Predicate and BDD

A *predicate* P defined on $\mathcal{B}(\mathbf{ST})$ is a function $P : \mathcal{B}(\mathbf{ST}) \to \{0,1\}$. The truth-value 1 (resp., 0) represents logical *true* (resp., *false*). The predicate *true* (resp., *false*) is identically 1 (resp., 0). A predicate P can be identified by a set B_P of basic state-trees $B_P := \{b \in \mathcal{B}(\mathbf{ST}) | P(b) = 1\}$. The satisfaction relation $P(b) = 1$ is written as $b \models P$ (b satisfies P). For a sub-state-tree $subST \in \mathcal{ST}(\mathbf{ST})$, $subST \models P$ if and only if $(\forall b \in \mathcal{B}(subST))\ b \models P$. The initial state-tree $\mathbf{ST}_0$ is represented by the predicates P_0 with $B_{P_0} := \{b \in \mathcal{B}(\mathbf{ST}) | b \models P_0\} = \mathcal{B}(\mathbf{ST}_0)$. An STS can be defined as $\mathbf{G} = (\mathbf{ST}, \mathcal{H}, \Sigma, \Delta, P_0)$. Write $Pred(\mathbf{ST})$ for the set of all predicates on $\mathcal{B}(\mathbf{ST})$. The partial order $\preceq$ is introduced on $Pred(\mathbf{ST})$, defined by $P \preceq P'$ iff $P \wedge P' = P$, i.e., $P \preceq P'$ if $b \models P \Rightarrow b \models P'$ for every $b \in \mathcal{B}(\mathbf{ST})$. A *state variable* for a holon H (or an OR superstate x) is a variable whose range is the internal state set of H (or the set of all children of x). Assign a state variable to each OR superstate on the state-tree $\mathbf{ST}$. Denote by v_x the state variable for the OR superstate x. The function Θ encodes a sub-state-tree to its corresponding predicate.

Definition 1. *[5] Let $\mathbf{ST}_1 = (X_1, x_0, \mathcal{T}_1, \mathcal{E}_1)$ be a sub-state-tree of $\mathbf{ST}$. Define $\Theta : \mathcal{ST}(\mathbf{ST}) \to Pred(\mathbf{ST})$ recursively by $\Theta(\mathbf{ST}_1) :=$*

$$\begin{cases} \bigwedge_{y \in \mathcal{E}_1(x_0)} \Theta(\mathbf{ST}_1^y), & \text{if } \mathcal{T}(x_0) = \textit{AND}; \\ \bigvee_{y \in \mathcal{E}_1(x_0)} ((v_{x_0} = y)) \wedge \Theta(\mathbf{ST}_1^y)), & \text{if } \mathcal{T}(x_0) = \textit{OR}; \\ 1, & \text{if } \mathcal{T}(x_0) = \textit{SIMPLE}. \end{cases}$$

In Definition 1, the operator "=" in $(v_{x_0} = y)$ returns value 1 iff v_{x_0} has been assigned value y. Notice that if x_0 is an OR superstate, the tautology [3]

$$\left(\bigvee_{y \in \mathcal{E}_1(x_0)} (v_{x_0} = y) \right) \equiv 1$$

can simplify $\Theta(\mathbf{ST}_1)$.

BDD representation of predicates: The states in the state space X^x of a holon H^x are encoded by BDD nodes (variables). Each element y in X^x is encoded as a vector of n binary values with $n = \lceil \log_2 |X^x| \rceil$, where $|X^x|$ is the state size of X^x. The encoding process is denoted by a function $f : X^x \to \{0,1\}^n$ that maps each element y in X^x to a distinct n-bit binary vector. According to [4], the n variables are denoted by x_j with $0 \leq j < n$. Let M_1 be the number of holons in an STS model and N_1 be the largest number of BDD nodes to encode a holon. The number of BDD nodes to encode an STS model is not exceeding $M_1 \times N_1$. For a synchronous product system with M_2 component automata, BDDs need to encode at most $N_2^{M_2}$ states after obtaining the monolithic model by the operation of parallel composition, where N_2 is the largest state size among the component automata. Obviously, the encoding efficiency of the monolithic model is far lower that of the STS hierarchical and modular encoding policy. As an illustration, we list the states and their BDD encoding vectors of holons in Example 1 in Table I.

[3] The predicate $\Theta(\mathbf{ST}_1)$ is independent of the state variable v_{x_0} if all descendants of x_0 are on the state tree.

TABLE I: BDD vectors encoding states of holons in Example 1

H^{R1}&H^{R2}		H^{B}&H^{K1}&H^{K2}	
States	BDD vectors	States	BDD vectors
B	<R1_0:0, R1_1:0, R1_2:0>	a	<B_0:0, B_1:0>
C	<R1_0:1, R1_1:0, R1_2:0>	c	<B_0:1, B_1:0>
D	<R1_0:0, R1_1:1, R1_2:0>	e	<B_0:0, B_1:1>
E	<R1_0:1, R1_1:1, R1_2:0>	g	<B_0:1, B_1:1>
I	<R1_0:0, R1_1:0, R1_2:1>	h	<K1_0:0>
J	<R1_0:1, R1_1:0, R1_2:1>	l	<K1_0:1>
L	<R2_0:0, R2_1:0, R2_2:0>	u	<K2_0:0>
K	<R2_0:1, R2_1:0, R2_2:0>	w	<K2_0:1>
S	<R2_0:0, R2_1:1, R2_2:0>	-	-
U	<R2_0:1, R2_1:1, R2_2:0>	-	-
V	<R2_0:0, R2_1:0, R2_2:1>	-	-
W	<R2_0:1, R2_1:0, R2_2:1>	-	-

III. Diagnoser Construction of HDES using STS

In this section, the diagnoser construction of an HDES modeled by an STS is investigated, which lays a solid foundation for symbolizing a diagnoser and diagnosability analysis later.

Before discussing the construction of a diagnoser, we need to figure out the characteristics of fault propagation in an HDES. Fault events may be present at all of the system levels. For instance, a failure in a sensor value (stuck-closed or stuck-open) may occur in lower levels of the system. On the other hand, software breakdown, planning failure, and scheduling error are some kind of failures that usually occur in higher levels of the system. Due to the system hierarchy, faults will be propagated horizontally and vertically, i.e., a fault can be propagated not only in the same level but also from the high level to the low level or vice versa. Hence, a holon may contain faulty states while the corresponding fault events appear in another holon.

Definition 2. [*Unobservable Reachability Function*] *Let* $\mathbf{G} = (\boldsymbol{ST}, \mathcal{H}, \Sigma, \Delta, \mathbf{ST}_0)$ *be an STS and* b *a basic state-tree of* $\boldsymbol{ST}$. *The unobservable reachability function* $\mathit{UR} : \mathcal{B}(\boldsymbol{ST}) \to Pwr(\mathcal{B}(\boldsymbol{ST}))$ *maps a basic state-tree* b *to a set of basic state-trees that can be reached from* b *via an unobservable string in* Σ_{uo}^*, *as defined as follows:*

$$\mathit{UR}(b) := \{\Delta(b, s) \in \mathcal{B}(\boldsymbol{ST}) \mid (\exists s \in \Sigma_{uo}^*) \Delta(b, s)!\}.$$

Definition 2 can be generalized to a basic state-tree subset $B' \subseteq \mathcal{B}(\boldsymbol{ST})$ by defining $\mathsf{UR}(B') = \bigcup_{b \in B'} \mathsf{UR}(b)$. For the basic state-tree $b = \{e, h, w\}$ in Fig. 2(b), we have $\mathsf{UR}(b) = \{\{e, h, w\}, \{g, h, w\}\}$.

Definition 3. [*Basic State-Tree Aggregation*] *Let* $\mathbf{G}$ *be an STS and* b *be a basic state-tree of* $\mathbf{ST}$. *A basic state-tree aggregation* A *is a non-empty set of basic state-trees such that* $b \in A$ *implies* $\mathit{UR}(b) \subseteq A$.

Let $\mathcal{L} = \{N\} \cup 2^{\mathcal{F}}$ be the set of *condition labels* with $\mathcal{F} = \{F_1, F_2, \ldots, F_m\}$ being the set of *fault labels* and N being the label for normal system operation. The label F_i $(1 \leq i \leq m)$ associates with the fault events in Σ_{f_i}. The fault condition labels are tracked and propagated via the fault propagation function $\nabla : \mathcal{L} \times \Sigma^* \to \mathcal{L}$ as $\nabla(\ell, s) :=$

$$\begin{cases} \{N\}, \text{ if } \ell = \{N\} \text{ and } \forall \sigma_f \in \Sigma_f, \sigma_f \notin s; \\ \{F_i \in F \mid F_i \in \ell \text{ or } \exists \sigma_f \in \Sigma_{f_i}, \sigma_f \in s\}, \text{otherwise.} \end{cases}$$

With the abuse of notation, $\sigma_f \in s$ indicates that the event σ_f exists in the string s.

The states $\mathcal{A}_d \subseteq 2^{\mathcal{B}(\boldsymbol{ST}) \times \mathcal{L}}$ in the diagnoser are in the form of $A_d = \{(b_1, \ell_1), \ldots, (b_k, \ell_k)\}$, where the pairs $(b_j, \ell_j) \in A_d$ capture the state estimations $b_j \in \mathcal{B}(\boldsymbol{ST})$ of the system under diagnosis associated with their condition labels, $\ell \subseteq \mathcal{L}$.

Let $A_d = \{(b_1, \ell_1), \ldots, (b_k, \ell_k)\} \in \mathcal{A}_d$. Then A_d is said to be

1) normal if $(\forall j \in [1, k])\ \ell_j = \{N\}$;
2) F_i-certain if $(\forall j \in [1, k])\ F_i \in \ell_j$;
3) F_i-uncertain if $(\exists n, r \in [1, k])\ F_i \in \ell_n$ & $F_i \notin \ell_r$.

Formally, the diagnoser $\mathbf{G}_d$ is defined by $\mathbf{G}_d = (\mathcal{A}_d, \Sigma_o, \Delta_d, A_{d0})$, where $\mathcal{A}_d$ is the set of basic state-tree aggregations (BSTAs) associated with condition labels; Σ_o is the observable event set; $\Delta_d : \mathcal{A}_d \times \Sigma_o \to \mathcal{A}_d$ is the (partial) global transition function; and A_{d0} is the initial BSTA associated with condition labels.

The system under diagnosis is initially normal and starts from the initial basic state-tree b_0. Let $b_c = (b, \ell)$. Extend the function UR by $\mathsf{UR}(b_c) = \{(b', \ell') \mid b' = \Delta(b, s), s \in \Sigma_{uo}^*, \ell' = \nabla(\ell, s)\}$. Let $A_{c0} = \{(b_0, \{N\})\}$. We have $A_{d0} = \mathsf{UR}(A_{c0})$. Starting from A_{d0}, the BSTAs and transition relation of the diagnoser can be recursively constructed. To this end, for any $A_d \in \mathcal{A}_d$ and $\sigma_o \in \Sigma_o$, $A_c = \bigcup_{(b,\ell) \in A_d} \{(\Delta(b, \sigma_o), \ell)\}$ and $A_d' = \mathsf{UR}(A_c)$. Add the transition (A_d, σ_o, A_d') to the list of admissible transitions of the diagnoser.

Example 2. *For the STS model with* $\Sigma_{uo} = \{12, 22, 23, f_1, f_2\}$ *and* $\Sigma_f = \{f_1, f_2\}$ *in Fig. 1, its diagnoser* $\mathbf{G}_d$ *can be built based on the above procedure. There are 42 BSTAs with condition labels and 110 transitions in* $\mathbf{G}_d$ *and we do not depict it here for saving space.*

Diagnosability is verified by checking the existence of an F_i-indeterminate cycle in the diagnosers $\mathbf{G}_d$. If there is no such a cycle in $\mathbf{G}_d$, then the system under diagnosis is diagnosable. Otherwise, the system is not diagnosable.

Theorem 2. *An HDES modeled by an STS* $\mathbf{G}$ *is diagnosable w.r.t.* Σ_f *if and only if there is no* F_i*-indeterminate cycle in* $\mathbf{G}_d = (\mathcal{A}_d, \Sigma_o, \Delta_d, A_{d0})$ *for any failure type* Σ_{f_i}.

Proof: Found in the Appendix. □

According to Theorem 2, we can infer that the HDES modeled by the STS $\mathbf{G}$ in Fig. 1 is not diagnosable since the F_1-uncertain cycle $A_{d2} \xrightarrow{\alpha} A_{d3} \xrightarrow{\beta} A_{d2}$ in $\mathbf{G}_d$ is F_1-indeterminate with $A_{d2} = \{(\{D, H\}, \{N\}), (\{F, H\}, \{F_1\})\}$ and $A_{d3} = \{(\{D, G\}, \{N\}), (\{D, M\}, \{F_2\}), (\{F, G\}, \{F_1\}), (\{F, M\}, \{F_1, F_2\})\}$. The BASTs $A_{d2} = \Delta_d(A_{d0}, s_1)$ and $A_{d3} = \Delta_d(A_{d0}, s_2)$ with $A_{d0} = (\{A, H\}, \{N\}), (\{E, H\}, \{F_1\})$, $s_1 = f_1\gamma(\alpha\beta)^*$, and $s_2 = \gamma(\alpha\beta)^*$.

IV. Symbolic Diagnoser and Diagnosability Analysis

In this section, we aim to present a symbolic approach to encode a diagnoser of an HDES for offline diagnosability analysis later. With the increasing scale and complexity of an HDES, the existing diagnosis methods face the computational complexity hurdle caused by the state explosion problem. In the worst case, the diagnoser construction suffers from the exponential complexity with respect to the state space of a system, which further aggravates the state explosion problem. To partially overcome the mentioned issues, predicates and predicate transformers are utilized to symbolically encode a diagnoser. Furthermore, a powerful data structure called binary decision diagrams (BDDs) is used for representing predicates. Owing to a hierarchical system structure, the encoding efficiency of symbolic computation is high such that the computer memory space for storing the state space of a diagnoser can be greatly saved.

A. Diagnoser Symbolization of HDES

In this part, we provide a symbolic approach to encode a diagnoser for an HDES modelled by an STS based on predicates. Let $P \in Pred(\mathbf{ST})$. The predicate transformer $\langle\cdot\rangle : Pred(\mathbf{ST}) \rightarrow Pred(\mathbf{ST})$ is defined according to the inductive definition:

1) $b \models P \Rightarrow b \models \langle P\rangle$;
2) $b \models \langle P\rangle$ & $b' \neq \emptyset$ & $\sigma \in \Sigma_{uo}$ & $\Delta(b,\sigma) = b' \Rightarrow b' \models \langle P\rangle$;
3) no other basic state-trees b satisfy $\langle P\rangle$.

In effect, the predicate $\langle P\rangle$ holds on all the basic state-trees that can be reached via P by unobservable paths only. We illustrate $\langle\cdot\rangle$ in Fig. 3, where $\sigma_i \in \Sigma_{uo}$ ($i \in [1,k]$) and $\sigma_o \in \Sigma_o$. Evidently, we can conclude that:

1) $P \preceq \langle P\rangle$, i.e., a basic state-tree satisfying P must satisfy $\langle P\rangle$;
2) $\Delta(b,s) \models \langle P\rangle$ if $\Delta(b,s)$ is defined for a basic state-tree $b \models P$ and a string $s \in \Sigma_{uo}^*$.

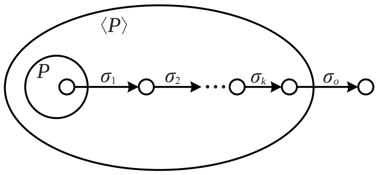


Fig. 3: The predicate $\langle P\rangle$.

The normal predicate $\langle P\rangle_N$ is defined as $\Delta(b,s) \models \langle P\rangle_N$ if $\Delta(b,s)$ is defined for a basic state-tree $b \models P$ and a string $s \in (\Sigma_{uo} - \Sigma_f)^*$. The fault predicate $\langle P\rangle_{F_i}$ is defined as $\Delta(b,s) \models \langle P\rangle_{F_i}$ if $\Delta(b,s)$ is defined for a basic state-tree $b \models P$, a string $s \in \Sigma_{uo}^*$, and $(\exists\sigma_f \in \Sigma_{f_i})\sigma_f \in s$.

In the process of diagnosability verification, each fault type is treated independently. Therefore, we construct a diagnoser for each fault type Σ_{f_i} and check its diagnosability. During fault propagation, There are two kinds of basic state-trees, i.e., normal and fault basic state-trees, generated. Let $\mathcal{P} = (P_N, P_{F_i}) \in Pred(\mathbf{ST}) \times Pred(\mathbf{ST})$ be a predicate pair with P_N being a normal predicate and P_{F_i} a fault predicate. The system condition $\kappa^i(\mathcal{P})$ of $\mathcal{P}$ is defined as

$$\kappa^i(\mathcal{P}) = \begin{cases} 0, & \text{if } P_{F_i} \equiv false; \\ i, & \text{if } P_N \equiv false; \\ -1, & \text{otherwise.} \end{cases}$$

A predicate pair $\mathcal{P}$ is said to be

- normal if $\kappa^i(\mathcal{P}) = 0$;
- F_i-certain if $\kappa^i(\mathcal{P}) = i$;
- F_i-uncertain if $\kappa^i(\mathcal{P}) = -1$.

Formally, the symbolic diagnoser $\mathbf{G}_d^i$ of an STS $\mathbf{G}$ for verifying F_i-diagnosability is defined as a five-tuple $\mathbf{G}_d^i = (\mathcal{P}_d^i, \Sigma_o, \Delta_d^i, \mathcal{P}_{d0}^i, \kappa^i)$, where $\mathcal{P}_d^i = \{(\mathcal{P}, \kappa^i(\mathcal{P})\}$ is the set of predicate pairs $\mathcal{P}$ associated with system conditions $\kappa^i(\mathcal{P})$; $\Sigma_o \subseteq \Sigma$ is the observable event set; $\Delta_d^i : \mathcal{P}_d^i \times \Sigma_o \rightarrow \mathcal{P}_d^i$ is the (partial) global transition function; $\mathcal{P}_{d0}^i = (\mathcal{P}_0, \kappa^i(\mathcal{P}_0))$ is the initial predicate pair associated with system condition; and κ^i is the system condition function.

A symbolic diagnoser is constructed recursively. Let P_0 be the predicate identified by the initial basic state-tree b_0. Since the system under diagnosis is initially normal, $\kappa^i(P_0) = 0$. Let $P_{0N} = P_0$ and $P_{0F_i} \equiv false$. Update P_{0N} by $P_{0N} = \langle P_{0N}\rangle_N$ and P_{0F_i} by $P_{0F_i} = \langle P_{0N}\rangle_{F_i}$; thus $\mathcal{P}_0 = (P_{0N}, P_{0F_i})$ and $\mathcal{P}_{d0}^i = (\mathcal{P}_0, \kappa^i(\mathcal{P}_0))$. Note that $P_{0N} \vee P_{0F_i} \preceq \langle P_0\rangle$ due to the existence of other fault types. For any observable event $\sigma_o \in \Sigma_o$, $P_{1N} = \langle\Delta(P_{0N}, \sigma_o)\rangle_N$ and $P_{1F_i} = \langle\Delta(P_{0F_i}, \sigma_o)\rangle \vee \langle\Delta(\langle P_0\rangle, \sigma_o)\rangle_{F_i}$. Then $\mathcal{P}_1 = (P_{1N}, P_{1F_i})$ and $\mathcal{P}_{d1}^i = (\mathcal{P}_1, \kappa^i(\mathcal{P}_1)$. The predicate P_{1F_i} is composed of the predicate $\langle\Delta(P_{0F_i}, \sigma_o)\rangle$ and the predicate $\langle\Delta(\langle P_0\rangle, \sigma_o)\rangle_{F_i}$ because of the characteristic of fault propagation. Add the transition $(\mathcal{P}_{d0}^i, \sigma_o, \mathcal{P}_{d1}^i)$ to the list of admissible transitions of the diagnoser $\mathbf{G}_d^i$. The relation between $\mathcal{P}_0 = (P_{0N}, P_{0F_i})$ and $\mathcal{P}_1 = (P_{1N}, P_{1F_i})$ is illustrated in Fig. 4, where $\sigma_f \in \Sigma_{f_i}$. In Example 1, we have $P_0 = \Theta(b_0) = \Theta(\{A, H\})$ initially. According to the procedure, $P_{0N} = P_0$ and $P_{0F_1} = \langle P_{0N}\rangle_{F_1} = \Theta(\{E, H\})$ can be computed. Let $\sigma_o = \gamma$. Then we have $P_{1N} = \Theta(\{D, H\})$ and $P_{1F_1} = \Theta(\{F, H\})$.

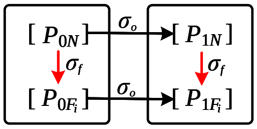


Fig. 4: The relation between $\mathcal{P}_0 = (P_{0N}, P_{0F_i})$ and $\mathcal{P}_1 = (P_{1N}, P_{1F_i})$.

On the basis of a symbolic diagnoser $\mathbf{G}_d^i$, the necessary and sufficient condition for diagnosability verification of an HDES modeled by an STS $\mathbf{G}$ is provided below.

Theorem 3. *An HDES modeled by an STS $\mathbf{G}$ is diagnosable w.r.t. Σ_f if and only if there is no F_i-indeterminate cycle in the diagnosers $\mathbf{G}_d^i = (\mathcal{P}_d^i, \Sigma_o, \Delta_d^i, \mathcal{P}_{d0}^i, \kappa^i)$ for any failure type Σ_{f_i}.*

Proof: Similar to the proof of Theorem 2. □

B. Diagnosability Verification

The detection of an F_i-indeterminate cycle is realized by a double-checking process [15]: 1) check whether an F_i-uncertain cycle exists or not in the diagnoser; 2) if it exists, check whether it corresponds to a faulty cycle and a non-faulty one in an intermediate model. Besides, each cycle analysis is performed from the initial state of the intermediate model. To improve the computing efficiency, we design a systematic procedure in which only the F_i-uncertain cycle under verification is sufficient to provide a verdict in the framework of STS.

Proposition 1. *Let* $cl = (\mathcal{P}_1, -1) \xrightarrow{\sigma_{o1}} (\mathcal{P}_2, -1) \xrightarrow{\sigma_{o2}} \cdots \xrightarrow{\sigma_{o(n-1)}} (\mathcal{P}_n, -1) \xrightarrow{\sigma_{on}} (\mathcal{P}_1, -1)$ [4] *with* $\mathcal{P}_k = (P_{kN}, P_{kF_i})$ *be an* F_i*-uncertain cycle in* $\mathbf{G}_d^i$*. Then there exists at least one fault-free cycle in* $\mathbf{G}$ *that has the same observation* $(\sigma_1\sigma_2\cdots\sigma_n)^*$.

Proof: Found in the Appendix. □

A fault-free cycle in $\mathbf{G}$ is a cycling trace that does not contain events in Σ_f. Proposition 1 indicates that if an F_i-uncertain cycle cl in $\mathbf{G}_d^i$ corresponds to a faulty cycle in $\mathbf{G}$, then the cycle cl is F_i-indeterminate.

Proposition 2. *Let* cl *be an* F_i*-uncertain cycle in* $\mathbf{G}_d^i$*. If* $(\forall k \in [1, n], \forall \sigma_f \in \Sigma_{f_i})\ \Delta(P_{kN}, \sigma_f) \equiv \text{false}$*, then* cl *is an* F_i*-indeterminate cycle.*

Proof: Found in the Appendix. □

Proposition 2 means that if there exist no faulty transitions from the normal predicate P_{kN} to the fault predicate P_{kF_i} in an F_i-uncertain cycle cl, then the cycle cl is F_i-indeterminate. Proposition 2 can be used as a sufficient condition for non-diagnosability, i.e., if the condition in Proposition 2 is satisfied, then the system is not diagnosable.

Definition 4. [*Sequence* S^{cl}] *Let* cl *be an* F_i*-uncertain cycle in* $\mathbf{G}_d^i$*. Sequence* $S^{cl} = S_1^{cl}, S_2^{cl}, \ldots$ *associated with* cl *is defined as follows:*

$$S_k^{cl} = \begin{cases} P_{1F_i}, & k = 1; \\ \langle \Delta(S_{k-1}^{cl}, \sigma_{(k-1)mod_n}) \rangle, & k > 1. \end{cases}$$

The role of S^{cl} is to find the actual faulty cycles corresponding to a given F-uncertain cycle if such a cycle exists. Except S_1^{cl}, sequence S^{cl} records the fault subpredicates $S_k^{cl} \preceq P_{1F_i}$ obtained via strings in $\sigma_k \Sigma_{uo}^*$ from S_{k-1}^{cl}. Let $S'^{cl} = S_1^{cl}, S_{(1+n)}^{cl}, \ldots, S_{(1+jn)}^{cl}, S_{(1+(j+1)n)}^{cl}, \cdots$ be a subsequence of S^{cl}. Obviously, sequence S'^{cl} is extracted from S^{cl} by considering sample predicates with n steps (n is the length of cl). Moreover, sequence S'^{cl} preserves some properties of suquence S^{cl} (e.g., convergence).

Proposition 3. $(\forall k \geq 1)$ $S'^{cl}_{k+1} \preceq S'^{cl}_k$*, i.e.,* $S_{(1+kn)}^{cl} \preceq S_{(1+(k-1)n)}^{cl}$.

Proof: Found in the Appendix. □

Proposition 3 indicates the subset containment relationship between predicates of sequence S'^{cl}. Based on Proposition 3, we can conclude that S'^{cl} can reach a fixed point, i.e., $(\exists j \geq 1)$ $S_{(1+jn)}^{cl} = S_{(1+(j-1)n)}^{cl}$.

Theorem 4. *An* F_i*-uncertain cycle* cl *in* $\mathbf{G}_d^i$ *is* F_i*-indeterminate if and only if the fixed point reached by* S'^{cl} *is not* false.

Proof: Found in the Appendix. □

Derived from the above theoretical results, a systematic procedure to check an F_i-indeterminate cycle is given as follows:

1) Determine whether the found cycle cl in $\mathbf{G}_d^i$ is F_i-uncertain or not. If so, go to Step 2);
2) Check whether the condition in Proposition 2 is satisfied or not. If so, then cl is an F_i-indeterminate cycle and the procedure stops. If not, go to Step 3);
3) Compute the successive predicates of sequence S^{cl}, and for each predicate check the conditions below:
    a) If $S_k^{cl} \equiv$ *false*, then cl is not an F_i-indeterminate cycle and the procedure stops;
    b) If $S_k^{cl} = S_{k-n}^{cl}$ $(k = 1 + jn, j \geq 1)$, then cl is an F_i-indeterminate cycle and the procedure stops; otherwise continue.

Once a cycle is detected in $\mathbf{G}_d^i$, the above procedure starts. We emphasize that the procedure terminates well since a fixed point can be found within a finite delay.

Example 3. *For the STS* $\mathbf{G}$ *in Fig. 1, the partial diagram of* $\mathbf{G}_d^1$ *is depicted in Fig. 5. Each* $\mathcal{P}_{dj}$ $(0 \leq j \leq 19)$ *is in the form of* $(\mathcal{P}_j, \kappa^1(\mathcal{P}_j))$ *with* $\mathcal{P}_j = (P_{jN}, P_{jF_1})$*, as shown in Table II. In* $\mathbf{G}_d^1$*, the cycle* $cl = \mathcal{P}_{d1} \xrightarrow{\alpha} \mathcal{P}_{d5} \xrightarrow{\beta} \mathcal{P}_{d1}$ *is* F_1*-uncertain. Then we move to Step 2) and find that the condition in Proposition 2 is satisfied, which means that* cl_1 *is an* F_1*-indeterminate cycle. Therefore, the system is not diagnosable. As an illustration for Step 3), we compute predicates* $S_1^{cl} = P_{1F_1} = \Theta(\{F, H\})$*,* $S_2^{cl} = \Theta(\{D, G, M\})$*, and* $S_3^{cl} = S_1^{cl}$ *successively. The result indicates that the cycle* cl_1 *is* F_1*-indeterminate. Hence, the system is not diagnosable.*

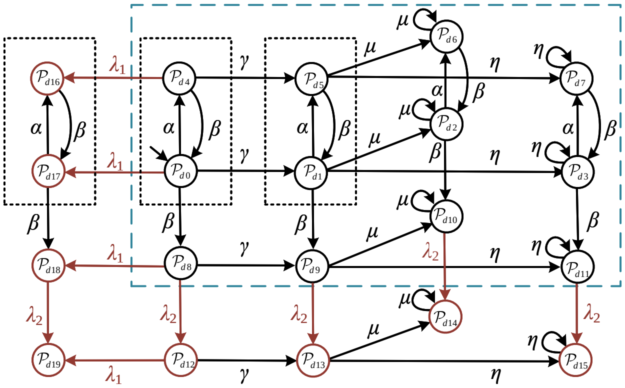


Fig. 5: Partial diagram of $\mathbf{G}_d^1$.

V. Heuristic On-the-fly Symbolic Algorithms for Diagnosability Verification

In this section, an efficient heuristic on-the-fly algorithm based on the depth-first search is developed for performing the level-to-level diagnosability analysis. Owing to symbolic computation and on-the-fly techniques, the memory cost and the overall running time for fault diagnosis can be significantly reduced.

Generally, no rules are given to heuristically explore an F_i-indeterminate cycle, i.e., the search direction is random.

[4] $(\mathcal{P}_k, -1) \xrightarrow{\sigma_k} (\mathcal{P}_{k+1}, -1)$ if $\Delta_d^i((\mathcal{P}_k, -1), \sigma_k) = (\mathcal{P}_{(k+1)mod_n}, -1)$ $(1 \leq k \leq n)$.

TABLE II: Partial predicate pairs and their system conditions in $\mathbf{G}_d^1$

Index j	Predicate pairs $\mathcal{P}_j$	Conditions κ_j^1
0	$(\Theta(\{B,L\}),\Theta(\{I,L\}))$	-1
1	$(\Theta(\{E,L\}),\Theta(\{J,L\}))$	-1
2	$(\Theta(\{E,L\}),false)$	0
3	$(false,\Theta(\{J,L\}))$	1
4	$(\Theta(\{B,S,U\}),\Theta(\{I,S,U\}))$	-1
5	$(\Theta(\{E,S,U\}),\Theta(\{J,S,U\}))$	-1
6	$(\Theta(\{E,S,U\}),false)$	0
7	$(false,\Theta(\{J,S,U\}))$	1
8	$(\Theta(\{B,V\}),\Theta(\{I,V\}))$	-1
9	$(\Theta(\{E,V\}),\Theta(\{J,V\}))$	-1
10	$(\Theta(\{E,V\}),false)$	0
11	$(false,\Theta(\{F,I\}))$	1
12	$(\Theta(\{B,h,u\}),\Theta(\{I,h,u\}))$	-1
13	$(\Theta(\{E,h,u\}),\Theta(\{J,h,u\}))$	-1
14	$(\Theta(\{E,h,u\}),false)$	0
15	$(false,\Theta(\{J,h,u\}))$	1
16	$(\Theta(\{a,S,U\}),false)$	0
17	$(\Theta(\{a,L\}),false)$	0
18	$(\Theta(\{a,V\}),false)$	0
19	$(\Theta(\{a,h,u\}),false)$	0

Owing to the hierarchical structure of an STS model, we can prioritize the branches to be explored during diagnosability verification. In this way, a diagnosability verdict can be quickly obtained. To hierarchically verify diagnosablity, events are selected successively according to their priorities from the current enabled event set. The event priority rule is that the higher level, the higher priority. Particularly, events in the same level have the same priority and thus the picked order of them is arbitrary. Note that the events labeling the incoming transitions of a holon are chosen at the end of the enabled event list with the same priority. The level of an event is defined as follows.

Definition 5. [*Event Level*] *Let σ be an internal event in a holon H^x. The level of σ is defined to be equal to that of an OR superstate x.*

Remark 1. *For diagnosability analysis, two worst cases exist requiring the entire space exploration of the symbolic diagnoser, as indicated below:*

1) *The system is diagnosable;*
2) *An F_i-indeterminate cycle is detected at the end of the exploration in the case of non-diagnosability.*

The flowchart of the proposed method is presented in Fig. 6. A detailed version with the developed algorithm is updated to the link https://github.com/gzudgwang/STS-fault-diagnosis.git. As an illustration for the algorithm, we verify F_1-diagnosability of the STS model $\mathbf{G}$ illustrated in Fig. 1. The partial predicates and transitions in $\mathbf{G}_d^1$ are shown in Fig. 5. Particularly, the predicates inside the blue dashed rectangular box will be explored first according to the rule of event priority. There are three F_1-uncertain cycles in Fig. 5 surrounded by the black dotted rectangular boxes and two of them are located in the blue box. Besides, the states represented by red cycles are those where the events in lower levels are enabled. By prioritizing the search direction, the exploration for the state space of a symbolic diagnoser is carried out level-by-level. In the best case, the F_1-uncertain cycle $\mathcal{P}_{d0} \xrightarrow{\alpha} \mathcal{P}_{d4} \xrightarrow{\beta} \mathcal{P}_{d0}$ in Fig. 5 can be found after recording two predicate pairs. Once an F_1-uncertain cycle is detected, we apply the procedure in Section IV.B to check whether this cycle is F_1-indeterminate or not. If the events are selected randomly, the unnecessary exploration for lower levels is inevitable, which usually leads to the waste of computer memory space and computing time.

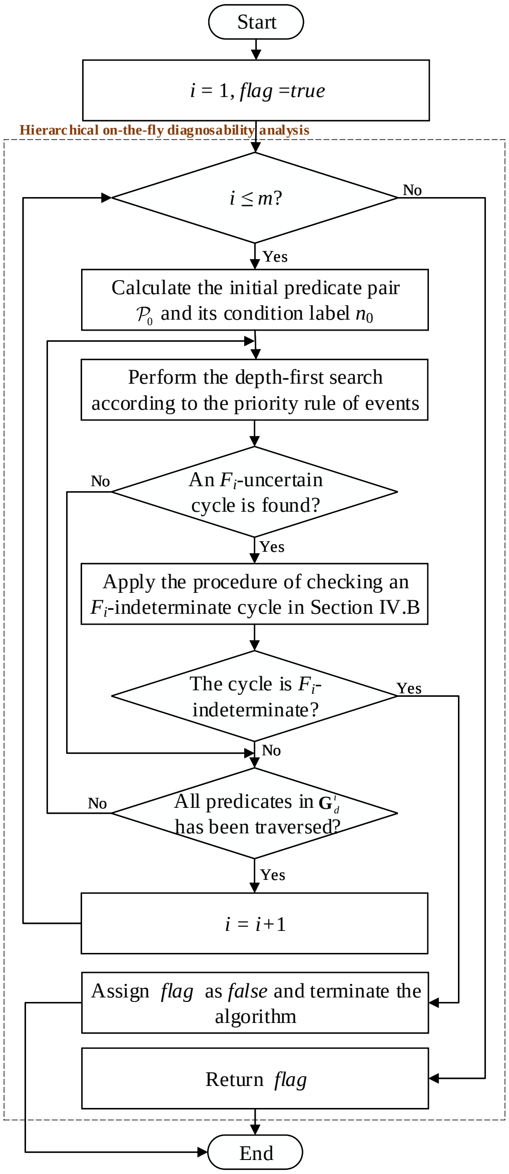

Start
i = 1, $flag$ =$true$
Hierarchical on-the-fly diagnosability analysis
$i \le m$?
No
Yes
Calculate the initial predicate pair $\mathcal{P}_0$ and its condition label n_0
Perform the depth-first search according to the priority rule of events
An F_i-uncertain cycle is found?
No
Yes
Apply the procedure of checking an F_i-indeterminate cycle in Section IV.B
The cycle is F_i-indeterminate?
Yes
No
All predicates in $\mathbf{G}_d^i$ has been traversed?
No
Yes
i = i+1
Assign $flag$ as $false$ and terminate the algorithm
Return $flag$
End

Fig. 6: The flowchart of the developed method.

Instead of constructing and storing a whole diagnoser directly, on-the-fly analysis based on depth-first search only needs to manage the traversed states. Besides, owing to state-tree structures (STS) and symbolic computation, the proposed approach has significant advantages over the existing diagnosis methods, as stated below.

1) For an HDES, the automata-based diagnosis methods needs to first convert an STS model into the equivalent automaton and then analyze diagnosability by a diagnoser or a verifier. However, the state size of a system

is exponential with respect to the number of its components; thus it is infeasible for a system with enormous state space. Owing to the structural organization of an STS model, diagnosability verification in this work is performed hierarchically from top to bottom and thus the STS model is unfolded level-by-level.

2) Without structural information, using BDDs alone cannot handle large-scale systems comfortably. An automaton is a special STS with one holon and there is only one state variable with its range over the entire state space. Hence, the symbolic representation of a flat model is not promising. However, it will be different if a model has a structure. In [5], it has been demonstrated that using BDDs has an advantage for systems modelled with structure.

In Table III, we list the main features of the proposed work and compare it with several existing methods. Here, "Hierarchy?" means the ability of modelling an HDES, "Memory" refers to the demanding of the computer memory space for diagnosability verification, "Symb?" denotes the use of predicates and BDDs, which can economically represent the state space, and "Conv?" indicates whether a hierarchical model is converted into its equivalent automaton first or not when dealing with the diagnosis problem. For a synchronous product system (a single layer STS), the computational complexity of diagnosability verification in [39] and [40] is of polynomial order in the model's state size. However, its state size increases exponentially with the number of components. Although the symbolic computation based on BDDs is utilized in [20], the hierarchical model needs to be converted into an equivalent automaton first before diagnosability analysis; thus the encoding efficiency is limited and it is infeasible for a large-scale system. In [32], a semi-symbolic diagnoser is constructed to analyze diagnosability of bounded labeled Petri nets. For an HDES, a petri net may not be a suitable modeling formalism. In [21], fault diagnosis is investigated for an HDES modelled by an HFSM. The limitations are that symbolic computation is not used and several assumptions are necessary. In [17], a modular diagnosis approach of DES is proposed, which avoids the global model. Nevertheless, shared events among components are assumed to be observable. Besides, a diagnosable system may become undiagnosable one using modular method.

TABLE III: Comparisons of relevant literature

References	Models	Hierarchy?	Memory	Symb?	Conv?
Our work	STS	Yes	Low	Yes	No
[21]	HFSM	Yes	High	No	No
[39], [40]	automata	No	High	No	Yes
[15]	automata	No	High	No	Yes
[17]	automata	No	High	No	Yes
[20]	automata	No	High	Yes	Yes
[32]	Petri net	No	Low	Yes	N/A

VI. Simulation Evaluation and Result Analysis

The developed algorithms are implemented and integrated in a software package STSLib. Several examples are presented to evaluate the effectiveness and the scalability of the proposed approach.

A. Ozone Generation Plant

The ozone plant [21] depicted in Fig. 7 has two parts: an oxygen supply unit (OSU) and an ozone generator unit (OGU).

- The OSU is used for generating the required oxygen and it is composed of five types of components, including a liquid oxygen tank, a pulse generator, a liquid oxygen inlet valve VT, two vaporizer outlet valves VP1 and VP2, and two vaporizers. Normally, two vaporizers operate alternatively and they are set to be in duty or standby by opening or closing the valves VP1 and VP2. The pulse generator is responsible for controlling the valves VP1 and VP2.
- The OGU is used for producing ozone from oxygen and it consists of five kinds of components, including an oxygen gas inlet valve V2, a cooling water valve V1, a power supply unit PSU, an ozone generation element OG, and an ozone gas outlet valve V3. Several sensors are equipped in the OGU. The changes of ozone concentration are measured by an ozone concentration analyzer OCA, marked as AM in Fig. 7. The sensors PS1 and PS2 measure the pressures at P1 and P2, respectively.

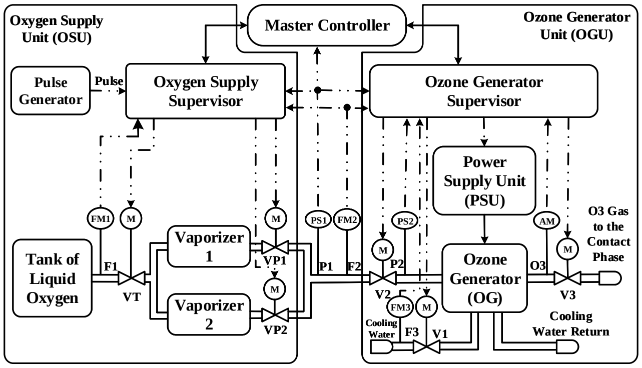


Fig. 7: A simplified ozone generation plant.

The units OSU and OGU are under the supervision of local supervisors or controllers issuing appropriate commands. A master controller or coordinator is used for avoiding the deadlocks among the subsystems and managing the command sequences. The commands "Start-Plant" and "Stop-Plant" are sent by the master controller to start and stop the plant.

1) The STS model of the plant: The STS model of the ozone plant has an AND superstate **ozone** at the top level. There are three components **OSU**, **OGU**, and **Master Controller** for the AND superstate **ozone**, as depicted in Fig. 8.

All sensors are assumed to be reliable or fault free. Only two types of valve faults, i.e., stuck-closed and stuck-open faults, are considered. Besides, suppose that valves may become stuck-closed (resp., stuck-open) only when they are closed (resp., open) for simplicity. The controller commands and events generated by the sensors are observable and all fault events are unobservable. Generally, the probability of simultaneous

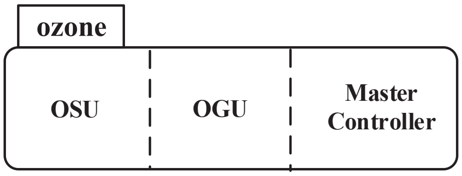


Fig. 8: High-level diagram of the plant.

failures is small in the case of independent failure modes. Hence, a single-failure scenario is assumed in each unit.

In each sensor, there are two output values: low/high for the pressure sensors, and flow/no-flow for the flowmeter. The holons describing dynamics of some typical system components are shown in Fig. 9.

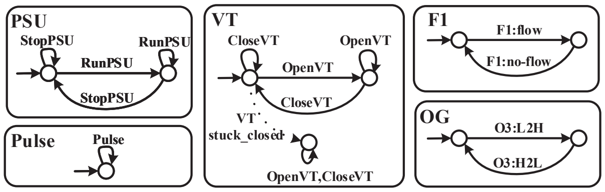


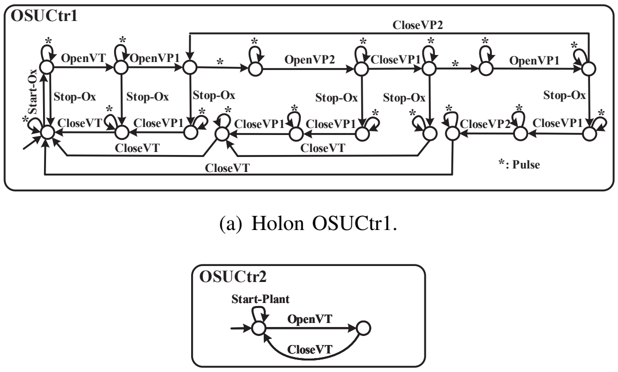
Fig. 9: Holons of several typical components.

(1) Controllers and interactions of **OSU**: There are two parts in the local controller. The first one is OSUCtr1 (Fig. 10(a)), which restricts the operation procedures in the unit, including start-up and shut-down sequences and the alternative running of two vaporizers. The second one is OSUCtr2 (Fig. 10(b)), ensuring that the plant initiates when the OSU resides at the shut-down state. In the OSU, the interactions among the components are shown as follows.

1) When valve VT is open or stuck-open (resp., closed or stuck-closed), there is "flow" (resp., "no-flow") at F1. (Holon OSUInt1)
2) When either valves VP1 or VP2 is open and the output value at F1 is high, the pressure becomes from low to high at P1. When either valve VT is closed or valves VP1 and VP2 are closed, and FM1 is showing a flow, the pressure becomes from high to low at P1. (Holon OSUInt2)

(2) Controllers and interactions of **OGU**: There are four parts in the local controller. Holon OGUCtr1 in Fig. 12 supervises the running process of the OGU. Besides, holons OGUCtr2, OGUCtr3, and OGUCtr4 enforce several specifications related to system safety, which are not given for saving space. In the OGU, the interactions among the components are shown as follows.

1) When the pressure is high at P2, the PSU is running, and the value is 'flow" at F3, the ozone concentration in the ozone generator becomes from low to high; otherwise, it becomes low. (Holon OGUInt1)
2) When valve V2 is open and the pressure is high at P1, the pressure becomes from low to high at P2. When either valve V2 is closed and valve V3 is open or valve V2 is open and the pressure is low at P1, the pressure becomes from high to low at P2. (Holon OGUInt2)



(a) Holon OSUCtr1.

(b) Holon OSUCtr2.

Fig. 10: Holons OSUCtr1 and OSUCtr2 of the OSU controller.

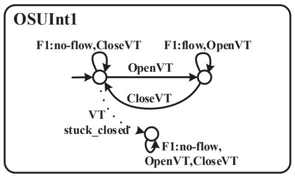


Fig. 11: Holon OSUInt1 for the interaction between F1 and VT.

(3) **Master Controller**: There are two parts in the master controller. The first one controls the operation sequences in the plant and coordinates the units OSU and OGU (holon MCtr1 in Fig. 13(a)). The second one is to ensure that the OGU runs when the pressure is high at P1 (holon MCtr2 in Fig. 13(b)).

After putting the holons of all the components, the interactions among the components, and the unit controllers together, we can obtain the STS model of the ozone generation plant. Here, we only give the sketch of state-tree of the plant, as depicted in Fig. 14. In the fault-free case, the size of state space that the state-tree represents is approximately 1.70×10^{10}. In practice, the reachable state size of the controlled system is 986. Nevertheless, the diagnoser design and diagnosability

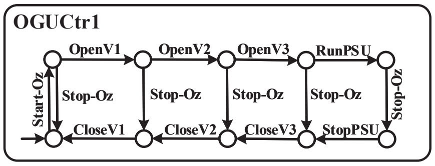
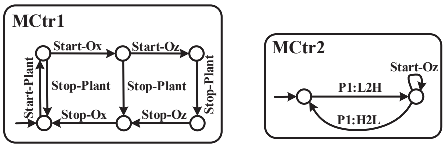


Fig. 12: Holon OGUCtr1 of the OGU sequence controller.



(a) Holon MCtr1.

(b) Holon MCtr2.

Fig. 13: Holons MCtr1 and MCtr2 of the master controller.

analysis of such a system remains a challenging task.

For convenience, we use F_i $(i \in [1,7])$ to represent fault labels of VT_Stuck_Closed, VT_Stuck_Open, VP1_Stuck_Closed, VP1_Stuck_Open, V1_Stuck_Closed, V1_Stuck_Open, and PSU_Fail, respectively. The simulation results of diagnosability analysis in different fault cases are listed in Table IV, where $|\mathbf{G}|$ is the reachable state size of the system, $|\mathbf{G}_d|$ is the state size of the symbolic diagnoser of the undiagnosable fault type, $|\mathcal{P}|$ is the number of visited predicate pairs, and "Diag?" is the result of diagnosablity.

TABLE IV: Simulation results of diagnosability analysis

Index	Failure Types	$\|\mathbf{G}\|$	$\|\mathbf{G}_d\|$	$\|\mathcal{P}\|$	Diag?
1	F_1	1925	1787	132	No
2	F_3	1972	987	41	No
3	F_7	1972	987	41	No
4	$F_{5,6}$	2920	993	41	No
5	$F_{3,4}$	2950	987	41	No
6	$F_{1,7}$	3850	1787	132	No
7	$F_{1,2,4}$	5796	2319	132	No
8	$F_{2,5,7}$	7872	1519	134	No
9	$F_{3,4,7}$	8166	987	41	No
10	$F_{1,2,5,6}$	8628	2325	132	No
11	$F_{2,3,4,5}$	11784	1519	134	No
12	$F_{3,4,6,7}$	11588	993	41	No
13	$F_{1,3,4,5,6}$	13459	1792	132	No
14	$F_{2,3,4,5,6}$	17472	1525	134	No
15	$F_{3,4,5,6,7}$	17488	993	41	No
16	$F_{1,2,3,4,5,6}$	25848	2325	132	No
17	$F_{1,3,4,5,6,7}$	34240	1793	132	No
18	$F_{2,3,4,5,6,7}$	48312	1525	134	No
19	$F_{1,2,3,4,5,6,7}$	71388	2325	132	No

Binary decision diagrams (BDDs) are efficient representations of subsets of the model's state space, especially with the help of structural information of STS. Owing to BDDs, the occupied memory resource is far less than an explicit state enumeration. The state space size of the ozone plant in Table IV and the number of BDD nodes in use are visualized in Fig. 15. As seen, the number of BDD nodes in use increases at a much slower rate than the state space size of the ozone plant. This is due to the fact that BDDs are particularly efficient for large sets of data.

B. Other Examples

Other examples include a system of automatic guided vehicles (AGVs) in a manufacturing workcell and a large-scale industrial system–Cluster Tool (CL), which are described in Table V. Here, we only present the results of fault diagnosis, as shown in Table VI. If interested, relevant models and diagnosis settings can be available from the authors.

TABLE V: Description of case studies

Examples	Description
AGVs	The coordination of a system of automatic guided vehicles (AGVs) serving a manufacturing workcell [2].
Cluster Tool (CL)	An integrated semiconductor system used for wafer processing [41].

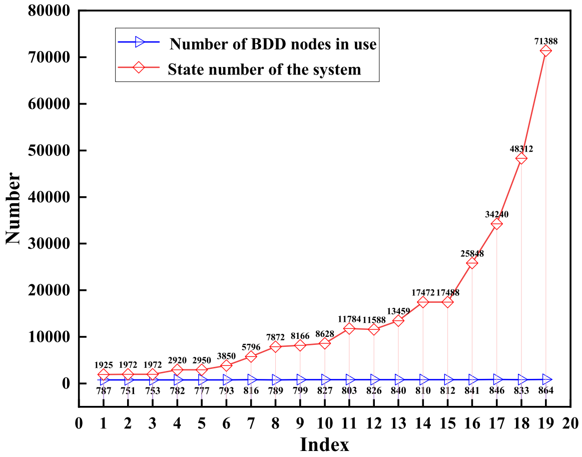


Fig. 15: The state space size of the ozone plant and the number of BDD nodes in use.

TABLE VI: Simulation results of diagnosability analysis for AGVs and Cluster Tool

Examples	$\|\mathbf{G}\|$	$\|\mathbf{G}_d\|$	$\|\mathcal{P}\|$	Diag?
AGVs	61440	62208	49920	Yes
AGVs	61440	62208	90	No
CL	2.3324×10^7	2.0887×10^7	1.6014×10^7	Yes
CL	7.8032×10^7	1.91977×10^7	84	No

C. Analysis on Simulation Results

The simulation results indicate that the proposed method is efficient for fault diagnosis of large-scale DES. The time cost of producing the diagnosis result is reasonable. Specially speaking, the verification of a diagnosable system results in a heavy computing burden because of the space exploration of the entire diagnoser. In addition to the system size, the number of loops in a diagnoser has a great impact on the diagnosis speed. Moreover, the ratio of observable and unobservable transitions affects the diagnosis efficiency. If the ratio is high, more computing cost is required. In detail, more computing time is consumed and more computer memory space is occupied caused by the traversed sequences becoming longer. Moreover, the symbolic diagnoser converges to a standard diagnoser in terms of the state size such that the symbolic technique has a limited role. If the ratio is low, the computing cost is significantly reduced since the memory space is greatly saved owing to the high encoding efficiency of BDDs.

For an HDES or a synchronous product system, most existing diagnosis methods in the literature, such as classical diagnoser-based methods [15], [20] and verifier-based methods [39], [40], first need a global model computed by the synchronous product of the component models and then perform diagnosability analysis based on it. Modular diagnosis methods [17], [21] can get rid of a global model, but extra assumptions are needed. To sum up, the existing diagnosis methods only work for the systems with small state space sizes. In addition, the developed software tools in the DES community, such as Supremica [42], TCT [43],

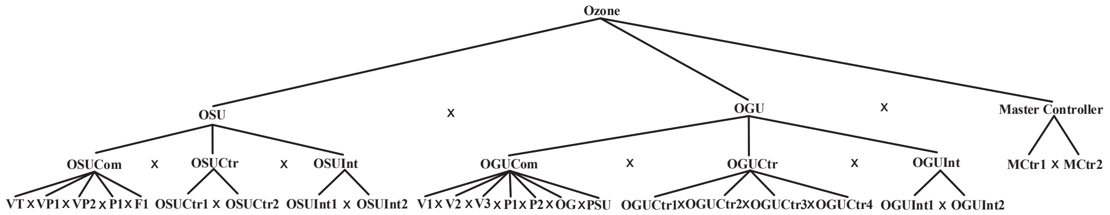


Fig. 14: The sketch of state-tree of ozone plant.

DESUMA [44], and libFaudes [45], either only focus on supervisor synthesis or will be crashed with a high probability when diagnosing a large-scale system. Hence, the Cluster Tool example cannot be handled by the aforementioned software tools. For the AGVs example and the ozone plant example, both the global models and their diagnosers or verifiers must be constructed completely and stored in the computer memory by the existing methods and software tools. Besides, the whole traversal is necessary when the systems are diagnosable, which will spend a large amount of computing time. Based on the developed method, it takes about three hours for the Cluster Tool example and several minutes for the AGVs example to output the diagnosis results in diagnosable cases, while only several seconds for both examples are needed in non-diagnosable cases. In practice, the status of a large-scale system can be estimated on-line once it is verified to be diagnosable. Therefore, the demanding for computing resources in the on-line diagnosis implementation is not high.

The presented results in Tables IV and VI can well support the claims mentioned before, which are also emphasized as follows.

1) In the case of non-diagnosability, the occupied computer memory space and computing time for diagnosability analysis using the proposed approach are far less than classical methods, which first construct the whole diagnosers or verifiers before diagnosability analysis. The on-the-fly technique is of great use, which stops the exploration of the state space of the diagnoser once the diagnosability condition is violated. The avoidance of exploring the whole diagnoser not only greatly saves the computer memory space but also considerably reduces the computing time.
2) Compared with the explicit state enumeration, symbolic computation based on predicates and binary decision diagrams (BDDs) can significantly compress the state space of the diagnoser. In the classical diagnoser, each state estimate is a state subset of the system. In the proposed approach, the diagnoser is symbolized and an estimate is represented by a BDD. Generally, the number of BDD nodes is far less than the size of state subset that the BDD represents.
3) Owing to the hierarchy of an STS model, the encoding efficiency of predicates and BDDs is higher than that in a naive monolithic model. Besides, diagnosabilty analysis can be performed in a level-by-level way, which can avoid the unnecessary exploration of successive layers if fault events are not diagnosable at the current layer.

VII. Conclusion and Future Work

Fault diagnosis of large-scale DES suffers from the double exponential complexity, i.e., its state size increases exponentially with respect to the number of system components and the diagnoser construction is subject to the exponential complexity with respect to the system state's size. This paper deals with the fault diagnosis problem of HDES in the framework of STS. The efforts of two aspects have been made to reduce the occupied computer memory space and consumed computing time during the diagnosis. First, a symbolic approach based on predicates and binary decision diagrams is presented for encoding a diagnoser, which avoids the explicit state enumeration. Second, a heuristic on-the-fly algorithm is proposed for testing diagnosability in a hierarchical way from top to bottom, which greatly lessens the calculative burden and improves the verification efficiency. From theoretical and practical viewpoints, this paper takes the requirement of safety-critical systems for fault diagnosis and computing efficiency of diagnosability analysis into account, aiming to provide a promising methodology and a software package to efficiently handle the diagnosis problem of industrial applications.

One limitation in this study is the use of entire STS models without fully considering structural decomposition. In the future work, we aim to solve the diagnosis problem in the framework of STS based on nest decomposition, which first disassembles an STS model into a group of nest STS and then analyze the diagnosability.

Appendix

We need the following definitions and lemmas for the proofs later.

Definition 6. [*Observation-Adjacency*] *For any two basic state-trees* b, $b' \in \mathcal{B}(\mathbf{ST})$ *and two condition labels* $\ell, \ell' \in \mathcal{L}$, (b', ℓ') *is said to be observation-adjacent to* (b, ℓ), *written as* $(b, \ell) \overset{\sigma}{\rightarrowtail} (b', \ell')$, *if there exists a string* $s\sigma t$ *in which* $s, t \in \Sigma_{uo}^*$ *and* $\sigma \in \Sigma_o$ *such that* $b' = \Delta(b, s\sigma t)$ *and* $\ell' = \nabla(\ell, s\sigma t)$. $\Diamond$

Assume that in the diagnoser $\mathbf{G}_d = (\mathcal{A}_d, \Sigma_o, \Delta_d, A_{d0})$ $cl = A_{d1} \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{n-2}} A_{d(n-1)} \xrightarrow{\sigma_{n-1}} A_{dn} \xrightarrow{\sigma_n} A_{d1}$ with $n \geq 1$ is an F_i-indeterminate cycle ($1 \leq i \leq m$). A cycle $cl' = (b_1, \ell_1) \overset{\sigma_1}{\rightarrowtail} \cdots \overset{\sigma_{n-2}}{\rightarrowtail} (b_{n-1}, \ell_{n-1}) \overset{\sigma_{n-1}}{\rightarrowtail} (b_n, \ell_n) \overset{\sigma_n}{\rightarrowtail} (b_1, \ell_1)$ is called an *underlying faulty cycle* of cl if $(b_j, \ell_j) \in A_{dj}$ and $F_i \in \ell_j$ ($1 \leq j \leq n$). Intuitively, if there is an F_i-indeterminate cycle, then the system has a cycle in the faulty condition F_i such that when it evolves on the cycle, it will generate the event sequence periodically. The cycle in the F_i and the corresponding event sequence keep the diagnoser in the F_i-uncertain cycle indefinitely, and in this case the system is not diagnosable.

Lemma 1. *Let* $p = A_{d1} \xrightarrow{\sigma_1} \cdots \xrightarrow{\sigma_{n-2}} A_{d(n-1)} \xrightarrow{\sigma_{n-1}} A_{dn}$ *(*$n \geq 2$*) be a path in the diagnoser* $\mathbf{G}_d$ *and all* A_{dj} *be* F_i*-uncertain (*$1 \leq j \leq n$*). For any* $(b_n, \ell_n) \in A_{dn}$*, there exist* $(b_k, \ell_k) \in A_{dk}$ *(*$1 \leq k \leq n-1$*) such that* $(b_k, \ell_k) \overset{\sigma_k}{\rightarrowtail} (b_{k+1}, \ell_{k+1})$*.*

A. Proof of Theorem 2 in Section III

Proof: (only if): Suppose that $\mathbf{G}$ is diagnosable, but there exists an F_i-indeterminate cycle cl in the diagnoser $\mathbf{G}_d = (\mathcal{A}_d, \Sigma_o, \Delta_d, A_{d0})$. Since $\mathbf{G}_d$ is reachable, there exists an event sequence that can take the diagnoser into A_{dk} belonging to cl. Let $(b_n, \ell_n) \in A_{dn}$ belong to an underlying faulty cycle of cl. By Lemma 1, there exist pairs $(b_1, \ell_1), \cdots, (b_{n-1}, \ell_{n-1})$ such that $(b_j, \ell_j) \overset{\sigma_i}{\rightarrowtail} (b_{j+1}, \ell_{j+1})$ $(1 \leq j \leq n-1)$. After reaching b_n with condition label ℓ_n, the system may sojourn on the underlying faulty cycle causing the diagnoser to stay on the F_i-indeterminate cycle indefinitely. Therefore, there exists a trajectory for the system leading to basic state-trees with fault label F_i such that the corresponding event sequence throws the diagnoser into a cycle of F_i-uncertain BSTAs and keeps it there indefinitely. Hence, the system is not diagnosable, which leads to a contradiction.

(if): Assume that no F_i-indeterminate cycle exists in the diagnoser $\mathbf{G}_d$. After the occurrence of F_i and the generation of a new observable event, the diagnoser reaches either F_i-certain or F_i-uncertain BSTA. If it is an F_i-certain BSTA, then it will remain F_i-certain (because fault is permanent) and the system is diagnosable. If it is an F_i-uncertain BSTA, then the number of F_i-uncertain BSTAs is bounded. After the generation of a bounded number of observable events, the diagnoser will reach an F_i-certain BSTA (the diagnoser gets trapped indefinitely in a cycle of F_i-uncertain BSTAs only if the cycle is F_i-indeterminate).

Let n denote the number of events that it takes the diagnoser to detect and isolate. After the occurrence of fault events in Σ_{f_i}, the diagnoser can visit an F_i-uncertain BSTA A_d at most $|N_{A_d}|$ times, where $|N_{A_d}|$ is the number of basic state-trees with fault labels F_i. Then we have $n \leq c \times M + M$, where $c = \sum\limits_{A_d \in \mathcal{A}_d} |N_{A_d}|$ and M is the length of the longest path of faulty basic state-trees. Since $M \leq |\mathcal{A}_d|$ and $c \leq |\mathcal{A}_d| \cdot |\mathcal{A}_d|$, $n \leq c \times M + M \leq |\mathcal{A}_d| \cdot |\mathcal{A}_d| \cdot |\mathcal{A}_d| + |\mathcal{A}_d| = |\mathcal{A}_d|(|\mathcal{A}_d|^2 + 1)$. Consequently, the system is diagnosable with a finite delay $n = |\mathcal{A}_d|(|\mathcal{A}_d|^2 + 1)$, which proves the sufficiency.

B. Proof of Proposition 1 in Section IV.B

Proof: Suppose that no fault-free cycle exists in $\mathbf{G}$. Since faults are permanent, a cycle in $\mathbf{G}$ composed of several faulty basic state-trees and normal basic state-trees cannot exist. Hence, at least one faulty cycle exists in $\mathbf{G}$, which leads to the existence of F_i-uncertain cycle cl. In this case, event σ_n is not eligible at normal basic state-trees satisfying P_{nN}. Hence, after the occurrence of σ_n the successor predicate of P_{nN} must be faulty, which leads to a contradiction.

C. Proof of Proposition 2 in Section IV.B

Proof: From Proposition 1, there exists at least one fault-free cycle formed by basic state-trees in $\mathbf{G}$ that has the same observation $(\sigma_1\sigma_2 \cdots \sigma_n)^*$. Then, we only need to show that a corresponding faulty cycle formed by basic state-trees in $\mathbf{G}$ also shares the same observation as cl. Suppose $(\forall k \in [1, n], \forall \sigma_f \in \Sigma_{fi})$ $\Delta(P_{kN}, \sigma_f) \equiv$ *false*. Let P_k be the predicate satisfied by the state estimation after occurring event σ_k. Then, we have $P_{(k+1)mod_n F_i} = \langle \Delta(P_{kF_i}, \sigma_k) \rangle \vee \langle \Delta(P_k, \sigma_k) \rangle_{F_i}$. Based on Lemma 1, for any $b_{n+1} \models P_{1F_i}$, there exist $b_k \models P_{kF_i}$ $(1 \leq k \leq n)$ such that $(b_k, \ell_k) \overset{\sigma_i}{\rightarrowtail} (b_{k+1}, \ell_{k+1})$. Let $b_{n+1} = b_1$. Then $b_1, \cdots, b_n$ form an underlying faulty cycle, and we can infer that a corresponding faulty cycle formed by basic state-trees in $\mathbf{G}$ with the same observation as cl exists. Hence, the cycle cl is F_i-indeterminate as well.

D. Proof of Proposition 3 in Section IV.B

Proof: It is proved using mathematical induction.
Basis step: For $k = 1$, $S^{cl}_{n+1} \preceq S^{cl}_1$ is true due to $S^{cl}_1 = P_{1F_i}$ and $S^{cl}_2 = \langle \Delta(S^{cl}_1, \sigma_1) \rangle \preceq P_{2F_i} = \langle \Delta(P_{1F_i}, \sigma_1) \rangle \vee \langle \Delta(P_1, \sigma_1) \rangle_{F_i}$. With the same reasoning along the event sequence $\sigma_1, \ldots, \sigma_n$, we have $S^{cl}_n = \langle \Delta(S^{cl}_{n-1}, \sigma_{n-1}) \rangle \preceq P_{nF_i} = \langle \Delta(P_{(n-1)F_i}, \sigma_{n-1}) \rangle \vee \langle \Delta(P_{n-1}, \sigma_{n-1}) \rangle_{F_i}$. Hence, $S^{cl}_{n+1} = \langle \Delta(S^{cl}_n, \sigma_n) \rangle \preceq P_{1F_i} = S^{cl}_1$.
Inductive step: Suppose $S^{cl}_{1+kn} \preceq S^{cl}_{1+(k-1)n}$. We need to show $S^{cl}_{1+(k+1)n} \preceq S^{cl}_{1+kn}$. By $S^{cl}_{1+kn} = \langle \Delta(S^{cl}_{kn}, \sigma_n) \rangle$ and $S^{cl}_{(k+1)n} \preceq S^{cl}_{kn}$, $S^{cl}_{(k+1)n} = \langle \Delta(S^{cl}_{(k+1)n}) \rangle \preceq \langle \Delta(S^{cl}_{kn}, \sigma_n) \rangle = S^{cl}_{1+kn}$.

E. Proof of Theorem 4 in Section IV.B

Proof: (only if): Suppose that cl is an F_i-indeterminate cycle. Then we need to show that the fixed point reached by sequence S'^{cl} associated with cl is no-empty.

Since cl is an F_i-indeterminate cycle, at least one faulty cycle formed by basic state-trees in $\mathbf{G}$ exists. Assume that there exist exactly M faulty cycles $(M \geq 1)$. There exist a string s^j_l in Σ^*_{uo} and a basic state-tree b^j_l satisfying P_{lF_i} such that $b^j_{(l+1)_{mod_n}} = \Delta(b^j_l, s^j_l \sigma_l)$ and $b^j_1 = \Delta(b^j_n, s^j_n \sigma_n)$ $(1 \leq l \leq n,\ 1 \leq j \leq M)$. Thus, $(\forall k \in \mathbb{N}^*)$ $b^j_l \models S^{cl}_{l+nk}$, indicating that all the terms of S'^{cl} are non-empty. Clearly, the reached fixed point is also non-empty.

(if): Suppose that sequence S'^{cl} associated with cl has a non-empty fixed point. Now, we need to show that cl is an F_i-indeterminate cycle. From Proposition 1, the existence of a faulty cycle sharing the same observation with cl is sufficient.

We know that there exists an integer $k \in \mathbb{N}^*$ such that $S^{cl}_{1+kn} = S^{cl}_{1+(k-1)n}$. Due to $S^{cl}_{1+kn} \not\equiv false$, we assume that the predicate S^{cl}_{1+kn} holds exactly on the basic state-tree subset $B_{S^{cl}_{1+kn}} = \{b_1, \ldots, b_N\}$. According to the definition of sequence S^{cl}, there exist $b_r, b_j \in B_{S^{cl}_{1+kn}}$, and $t = s_1\sigma_1 s_2\sigma_2 \ldots s_{n-1}\sigma_{n-1} s_n\sigma_n$ with $s_l \in \Sigma^*_{uo}$ such that $b_r = \Delta(b_j, t)$ $(1 \leq l \leq n, 1 \leq r, j \leq N)$. By repeating this procedure to b_r at least N times, we can infer that b_r is certainly visited twice, which indicates the existence of at least one faulty cycle. Therefore, the cycle cl is F_i-indeterminate.

References

[1] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete-event processes," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, 1987.

[2] W. M. Wonham and K. Cai. *Supervisory Control of Discrete-Event Systems*, Springer, 1st edition, 2019.

[3] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*, Springer, 2nd edition, 2008.

[4] C. Ma and W. M. Wonham, *Nonblocking Supervisory Control of State Tree Structures*. Berlin, Germany: Springer-Verlag, LNCIS, vol. 317, 2005.

[5] C. Ma and W. M. Wonham, "Nonblocking supervisory control of state tree structures," *IEEE Trans. Autom. Control*, vol. 51, no. 5, pp. 782–793, 2006.

[6] C. Ma and W. M. Wonham, "STSLib and its application to two benchmarks," in *Proc. Workshop Discrete-Event Syst*, Göteborg, Sweden, 2008, pp. 119–124.

[7] C. Gu, X. Wang, Z. Li, and N. Wu, "Supervisory control of state-tree structures with partial observation," *Inf. Sci.*, vol. 465, pp. 523–544, 2018.

[8] C. Gu, X. Wang, and Z. W. Li, "Synthesis of supervisory control with partial observation on normal state-tree structures," *IEEE Trans. Autom. Sci. Eng.*, vol. 16, no. 2, pp. 984–997, 2019.

[9] D. Wang, X. Wang, and Z. Li, "Nonblocking supervisory control of state-tree structures with conditional-preemption matrices," *IEEE Trans. Ind. Inf.*, vol. 16, no. 6, pp. 3744–3756, 2020.

[10] X. Wang, Z. Li, and W. M. Wonham, "Real-time scheduling based on supervisory control of state-tree structures," *IEEE Trans. Autom. Control*, vol. 66, no. 9, pp. 4230–4237, 2020.

[11] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comp. Programming*, vol. 8, no. 3, pp. 231–274, 1987.

[12] S. B. Akers, "Binary Decision Diagrams," *IEEE Trans. Comput.*, vol. C-27, no. 6, pp. 509–516, 1978.

[13] R. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, 1986.

[14] F. Lin, "Diagnosability of discrete event systems and its applications," *Discrete Event Dyn. Syst.*, vol. 4, no. 2, pp. 197–212, 1994.

[15] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete-event systems," *IEEE Trans. Autom. Control*, vol. 40, no. 9, pp. 1555–1575, 1995.

[16] K. Schmidt, "Abstraction-based failure diagnosis for discrete event systems," *Syst. Control Lett.*, vol. 59, no. 1, pp. 42–47, 2010.

[17] R. Debouk, R. Malik, and B. Brandin, "A modular architecture for diagnosis of discrete event systems," in *Proc. 41st IEEE Conf. Dec. Cont.*, Las Vegas, USA, 2002, pp. 417–422.

[18] X. Yin and Z. Li, "Decentralized fault prognosis of DES with guaranteed performance bound," *Automatica*, vol. 69, pp. 375–379, 2016.

[19] R. Su and W. M. Wonham, "Global and local consistencies in distributed fault diagnosis for discrete-event systems," *IEEE Trans. Autom. Control*, vol. 50, no. 12, pp. 1923–1935, 2005.

[20] A. Boussif, M. Ghazel, and K. Klai, "Fault diagnosis of discrete-event systems based on the symbolic observation graph," *Int. J. Crit. Comput. Sys.*, vol. 8, no. 2, pp. 141–168, 2018.

[21] A. Mohammadi-Idghamishi and S. Hashtrudi-Zad, "Hierarchical fault diagnosis: application to an ozone Plant," *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 37, no. 5, pp. 1040–1047, 2007.

[22] A. Paoli and S. Lafortune, "Diagnosability analysis of a class of hierarchical state machines," *Discrete Event Dyn. Syst.*, vol. 18, no. 3, pp. 385–413, 2008.

[23] S. H. Zad, R. H. Kwong, and W. M. Wonham, "Fault diagnosis in discrete-event systems: Incorporating timing information," *IEEE Trans. Autom. Control*, vol. 50, no. 7, pp. 1010–1015, 2005.

[24] D. Thorsley and D. Teneketzis, "Diagnosability of stochastic discrete-event systems," *IEEE Trans. Autom. Control*, vol. 50, no. 4, pp. 476–492, 2005.

[25] M. Luo, Y Li, F. Sun, and H Liu, "A new algorithm for testing diagnosability of fuzzy discrete event systems," *Inf. Sci.*, vol. 185, no. 1 pp. 100–113, 2012.

[26] S. H. Zad, R. H. Kwong, and W. M. Wonham, "Fault diagnosis in discrete-event systems: Framework and model reduction," *IEEE Trans. Autom. Control*, vol. 48, no. 7, pp. 1199–1212, 2003.

[27] D. Wang, X. Wang, and Z. Li, "State-based fault diagnosis of discrete-event systems with partially observable outputs," *Inf. Sci.*, vol. 529, pp. 87–100, 2020.

[28] M. P. Cabasino, A. Giua, and C. Seatzu, "Fault detection for discrete event systems using Petri nets with unobservable transitions," *Automatica*, vol. 46, no. 9, pp. 1531–1539, 2010.

[29] X. Yin, "Verification of prognosability for labeled Petri nets," *IEEE Trans. Autom. Control*, vol. 63, no. 6, pp. 1828–1834, 2018.

[30] D. Lefebvre, "On-line fault diagnosis with partially observed Petri nets," *IEEE Trans. Autom. Control*, vol. 59, no. 7, pp. 1919–1924, 2014.

[31] B. Liu, M. Ghazel, and A. Toguyéni, "On-the-fly and incremental technique for fault diagnosis of discrete event systems modeled by labeled Petri nets," *Asian J. Control*, vol. 19, no. 5, pp. 1659–1671, 2017.

[32] A. Boussif, M. Ghazel, and K. Klai, "A semi-symbolic diagnoser for fault diagnosis of bounded labeled petri nets," *Asian J. Control*, vol. 23, no. 2, pp. 648–660, 2021.

[33] C. Basilio, C. Hadjicostis, and R. Su, "Analysis and control for resilience of discrete event systems: fault diagnosis, opacity and cyber security," *Found. Trends Syst. Control*, vol. 8, no. 4, pp. 285–443, 2021.

[34] M. Sköldstam, K. Åkesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," in *Proc. 46th IEEE Conf. on Dec. and Cont.*, New Orleans, USA, 2007, pp. 3387–3392.

[35] S. Miremadi, B. Lennartson, and K. Åkesson, "A BDD-based approach for modeling plant and supervisor by extended finite automata," *IEEE Trans. Control Syst. Technol.*, vol. 20, no. 6, pp. 1421–1435, 2012.

[36] L. Ouedraogo, R. Kumar, R. Malik, and K. Åkesson, "Nonblocking and safe control of discrete-event systems modeled as extended finite automata," *IEEE Trans. Autom. Sci. Eng.*, vol. 8, no. 3, pp. 560–569, 2011.

[37] R. Devaraj, A. Sarkar, and S. Biswas, "Supervisory control approach and its symbolic computation for power-aware RT scheduling," *IEEE Trans. Ind. Inf.*, vol. 15, no. 2, pp. 787–799, 2019.

[38] A. Saadatpoor, "Timed state tree structures: supervisory control and fault diagnosis," *PhD thesis*, University of Toronto, Ontario, ON, Canada, 2009.

[39] S. Jiang, Z. Huang, V. Chandra, and R. Kumar, "A polynomial algorithm for testing diagnosability of discrete-event systems," *IEEE Trans. Autom. Control*, vol. 46, no. 8, pp. 1318–1321, 2001.

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

[41] R. Su, J. Schupen, and J. Rooda, "Aggregative synthesis of distributed supervisors based on automaton abstraction," *IEEE Trans. Autom. Control*, vol. 55, no. 7, pp. 1267–1640, 2010.

[42] R. Malik, K. Akesson, H. Flordal, and M. Fabian, "Supremica–an efficient tool for large-scale discrete event systems," *IFACPapersOnLine*, 2017, pp. 5794–5799.

[43] L. Feng and W. M. Wonham, "TCT: a computation tool for supervisory control synthesis," in *Proc. 8th Int. Workshop Discrete Event Syst.*, Ann Arbor, USA, 2006, pp. 388–389.

[44] L. Ricker, S. Lafortune, and S. Genc, "DESUMA: a tool integrating GIDDES and UMDES," in *Proc. 8th Int. Workshop Discrete Event Syst.*, Ann Arbor, USA, 2006, pp. 392–393.

[45] T. Moor, K. Schmidt, and S. Perk, "libFaudes—an open source C++ library for discrete event systems," in *Proc. 9th Int. Workshop Discrete Event Syst.*, Göteborg, Sweden, 2008, pp. 125–130.

Deguang Wang received the B.S. degree in Automation and the Ph.D. degree in Control Science and Engineering from Xidian University, Xi'an, China, in 2014 and 2019, respectively. From 2016 to 2017, he was a visiting Ph.D student at the Systems Control Group, Department of Electrical and Computer Engineering, University of Toronto, Canada. He is currently a lecturer in Guizhou University. His current research interests include supervisory control of discrete-event systems, supervisory control of state-tree structures, fault diagnosis.

Xi Wang received the B.S. degree in Automation from Liren College, Yanshan University, Qinhuangdao, China, in 2008, and the M.S. and Ph.D degrees in Mechanical-Electronic Engineering from Xidian University, Xi'an, China, in 2011 and 2016, respectively. He joined Xidian University as a Lecturer in 2016. During 2013 and 2015, he was a visiting Ph.D student at the Systems Control Group, Department of Electrical and Computer Engineering, University of Toronto, Canada. Dr. Wang has been awarded a 24-month Humboldt Research Fellowship for Postdoctoral Researchers during February 2018 and January 2020. His research interests include dynamic reconfiguration and scheduling of real-time systems and supervisory control of discrete-event systems.

Jing Yang received the B.S. Degree in Mechanical and Electrical Integration from Guizhou University of Technology, Guiyang, China, in 1997, the M.S. degree in Control Science and Engineering from Guizhou University of Technology, Guiyang, China, in 2004, and the Ph.D degree in Control Science and Engineering from Jiangnan university, Wuxi, China in 2010. He is currently a professor in Guizhou University. His research interests include Internet of Things, Swarm optimization and Control theory and its application.

Zhiwu Li (Fellow, IEEE) received the B.S., M.S., and Ph.D. degrees all from Xidian University, Xi'an, China, in 1989, 1992, and 1995, respectively. Dr. Li held visiting professor positions at the University of Toronto, Technion (Israel Institute of Technology), Martin-Luther University at Halle (supported by Alexander von Humboldt Foundation), University of Cagliari, Politecnico di Bari, Conservatoire National des Arts et Métiers (Cnam, supported by the program of Research in Paris), King Saud University, and Meliksah University. He has published three monographs in Springer (2009; 2023) and CRC Press (2013). His research was cited by leading business giants including IBM, HP, ABB, Volvo, GE, GM, Mitsubishi, Ford Car, and Huawei. His current research interests include Petri net theory & applications, supervisory control of discrete event systems, data modeling, and production automation. Dr. Li serves (served) as Associate Editor of IEEE Transactions on Automation Science and Engineering, IEEE Transactions on Systems, Man, and Cybernetics: Systems and Human Beings, IEEE Transactions on Systems, Man, and Cybernetics: Systems, Information Sciences (Elsevier), IEEE Access (Senior Editor), and Scientific Reports. He is a Fellow of IEEE (2016) and was selected as Thomson Reuters Highly Cited Researchers in the category of Engineering from 2014—2018.