Optimized diagnosability of distributed discrete event systems through abstraction

Lina Ye

Optimized Diagnosability of Distributed Discrete Event Systems Through Abstraction

THÈSE

UNIVERSITÉ PARIS SUD 11

présentée pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ PARIS SUD 11

Spécialité: INFORMATIQUE

Par

Lina YE

Directeur de thèse

Prof. Philippe DAGUE

Thèse soutenue le 7 Juillet 2011 devant le jury composé de:

Mme. Marie-Odile CORDIER, professeur, Université de Rennes 1	Rapporteur
M. Philippe DAGUE, professeur, Université Paris Sud	Directeur de thèse
M. Paul GASTIN, professeur, ENS de Cachan	Examinateur
M. Stéphane LAFORTUNE, professeur, University of Michigan	Rapporteur
Mme. Fatiha ZAÏDI, Maître de conférence, Université Paris Sud	Examinateur

Acknowledgment

I would like to express my deep sense of gratitude to Prof. Philippe DAGUE for his invaluable help and patient guidance during the course of my Ph.D research study and to Prof. Yuhong YAN for her useful discussions. I am highly indebted to them for constantly encouraging me by giving their critics on my work as well as giving me the support and confidence. I would like to thank Professor Stéphane LAFORTUNE and Professor Marie-Odile CORDIER to give me instructive comments that help me to improve my approaches. I am grateful to professor Paul GASTIN and professor Fatiha ZAÏDI for their participation in my defense. Then I also want to thank the team Leo that reunites INRIA researchers and faculty from IASI team of the LRI lab, providing me a friendly work atmosphere and many nice memories.

My deepest thanks are due to my parents, Xiaoru Tang and Huifu YE, who support and believe in me all the time. My sincere thanks to my husband, Zhihai ZHOU, for his infinite support and encouragement. my achievements are the result of the love, sacrifice and concerns from my family. You are and always will be present in my heart.

Lina YE

July 2011

University of Paris South 11, Lri

Abstract

Over the latest decades, much research work has been done on automatic fault diagnosis. However, it is imperative to analyze at system design stage how correctness and efficiency any diagnosis algorithm can achieve. Thus many studies were interested in analyzing and characterizing the properties of diagnosability of a system. Diagnosability is the property of a system ensuring that it generates observations for detecting and discriminating faults in finite time after their occurrence.

In this thesis, we investigate how to optimize distributed diagnosability analysis by abstracting necessary and sufficient information from local objects to decide global diagnosability decision. The algorithm efficiency can be greatly improved by synchronization of abstracted local objects compared to that of non abstracted local ones.

Then we extend the distributed diagnosability algorithm from fault event first to simple pattern and then to general pattern, where pattern can describe more general objects in the diagnosis problem, e.g. multiple faults, multiple occurrences of the same fault, ordered occurrence of significant events, etc. In the distributed framework, the pattern recognition is first incrementally performed normally in a subsystem and then pattern diagnosability can be determined by adjusting abstracted method used in fault event case. We prove the correctness and efficiency of our proposed algorithm both in theory through proof and in practice through implementation.

Finally we study joint diagnosability problem in systems with autonomous components, i.e. observable information is distributed instead of centralized. In other words, each component can only observe its own observable events. We give joint diagnosability definition. And then we discuss the undecidability of joint diagnosability in the general case, i.e., communication events are not observable, before proposing an algorithm to test its sufficient condition. In addition, we also get a decidability result and algorithm when communications are observable.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the latest decades, with the advancement of technologies, systems are becoming more and more complex since more performance requirements are imposed on them and thus more errors that they are subject to. However, it is not realistic to detect faults manually for complex systems. Automated diagnosis mechanisms are therefore required to monitor large distributed applications such as transportation systems, communication networks, manufacturing systems, web services, spatial systems and power systems. For example, some industrial disasters could have been prevented by well designed diagnosis and repair devices, like total blackouts of important big cities in the world, nuclear power plant accidents, etc. Thus the high reliability and quality of services are required even in faulty situations. In other words, it is crucial for a complex system to perceive that it is not operating correctly and then without human intervention, to detect and isolate original faults, which will be restored by repair plans to normality.

1.1 Motivation

In the literature, three types of systems are under investigation for diagnosis problem: continuous systems, discrete event systems and hybrid systems ([6], [7], [24], [26], [31], [28], [29], [36], [37], [38]). In this thesis, the dynamic systems studied are discrete event systems (abbreviated DES hereafter). Given a system, if its state space is naturally described by a discrete set and if state transitions are only observed at discrete points in time, we associate these transitions with events and this system is called a DES. The reason why we choose DES for investigation is that most of the man-made systems are DES and that continuous systems can be abstracted to be DES. Nowadays lots of works have been studied on control of DES, including diagnosis algorithm,

diagnosability analysis, predictability analysis, etc ([5], [8], [12], [14], [21], [22]).

Generally speaking, diagnosis reasoning is to detect possible faults that can explain the observations. The possibility to achieve such a diagnosis reasoning depends on the diagnosability of the system. Diagnosability is an important property that determines at design stage how accurate any diagnosis algorithm can be on a partially observable system and thus has significant economic impact on the improvement of performance and reliability of complex systems. The diagnosability analysis problem has received considerable attention in the literature. Now let us review the existing works concerning diagnosability for DES and analyze their possible improvable aspects.

1. Some existing works analyze diagnosability in a centralized way ([61], [44] and [15]), i.e., the knowledge of the monolithic model of a given system is hypothesized, which is the very powerful information for diagnosability analysis. However, real systems, e.g. telecommunication networks, water distribution networks, transportation systems, are steadily growing in terms of sizes, complexity and interactions. The centralized diagnosability approach requires an unrealistic combinatorial explosion of the search space.

2. Very recently the distributed approach for diagnosability began to be investigated ([53], [52] and [64]), relying on local objects. More precisely, in these distributed approaches, original diagnosability information can be obtained from the components where the fault may occur and then the global decision is calculated by checking its global consistency. However, even using local objects, during global checking procedure, the abstraction level of local information is not enough high such that in the worst case, the final state space is either the same as in the centralized approach ([53]) or reduced compared to the centralized one ([64]) but still quite large.

3. Some recent works have generalized the property usually checked in diagnosability, i.e., the occurrence of a fault event, to the recognition of a pattern that can represent more general objectives such as multiple faults, ordered occurrence of significant events, multiple occurrences of the same fault, etc ([34] and [43]). Actually the single fault event case is one special case of the pattern one. All works about pattern case adopt centralized framework, which, as said above, is not realistic due to the combinatorial explosion of the search state space.

4. All above approaches assume that each observable event in the system can be observed by all components in the system, i.e., globally observed. However, there are some cases where

it is not possible to assume the presence of global information. For example, networked control systems are characterized by that multiple distributed components possess their own part of available information instead of global knowledge. Thus some concerned works about distributed observations are investigated ([30], [72] and [56]). But they assume there is no communication between different sites. In other words, they separate several sites from the monolithic model of the system and each site can observe one subset of observable events set of the whole system. Then each site decides its own local decision from its own observations. With some merged rules, these local decisions are combined to get global decision. Clearly, these approaches are based on the monolithic model, which is not practical for real systems.

From above, we know that the study about diagnosability analysis for DES in literature develops quite a lot and there are whereas lots of aspects we can improve. The next section will show our contributions to diagnosability problem considering the insufficiency of the current works.

1.2 Contribution

There are several contributions in this thesis to diagnosability problem, which are described as follows.

- We optimize distributed diagnosability algorithm based on the approach of [53] by improving the abstraction level of local objects to reduce the final search state space.
- Then we extend pattern diagnosability analysis from centralized framework to distributed one, where the high abstraction level of local objects is adopted.
- We also investigate distributed diagnosability without global knowledge, which means that the available observations are distributed into local components and there is no assumption about the monolithic model for the considered system.

Next we describe our major contributions in detail.

1.2.1 Optimization of distributed diagnosability through abstraction

In chapter 3, we describe the major steps concerning how to optimize distributed diagnosability through abstraction.

- First we gear the definition of classical diagnosability for an entire system to that of regional diagnosability for a subsystem, which leads to defining a diagnosable subsystem.
- Then we describe how to improve diagnosis algorithm in terms of observation reduction with a given diagnosable subsystem in a formal way.
- And we provide a new distributed theoretical framework to check regional diagnosability and thus diagnosability of distributed systems. Instead of performing diagnosability verification on global object or local objects, we abstract necessary and sufficient diagnosability information from local objects and then distribute the search on these abstracted local ones. This algorithm is optimized in the sense that with our abstracted diagnosability information, the search state space is reduced to be as small as possible.
- We also discuss the strategy of next component selection for further exploitation during global consistency checking such that the returned diagnosable subsystem being a minimal diagnosable subsystem is more possible when the system is diagnosable.
- And then the diagnosability result that we obtain can help in improving the diagnosis algorithm when the system is diagnosable, in which case the algorithm returns a diagnosable subsystem. Otherwise, the algorithm provides some helpful information about indistinguishable behaviors that can be used to upgrade the diagnosability level of the system when the system is verified to be not diagnosable.

This major contribution, to some extent, fills up the gaps of the first and the second points described in the section 1.1. In other words, this approach not only takes into account the distributed nature of real systems but also performs a higher level of abstraction from local objects, which greatly reduces the final search space compared to the current existing works.

1.2.2 Distributed pattern diagnosability

The important steps concerning distributed pattern diagnosability are shown as below.

- First we extend pattern diagnosability problem from centralized framework to distributed one.
- Then pattern recognition can be checked by constructing pattern recognizers for incrementally extended subsystems. More precisely, the subsystem is extended by synchronizing the

diagnosability relative part of the current subsystem with next selected component. In this way we may avoid global model construction considering that normally the diagnosability relative part of the subsystem is a small subpart of the whole subsystem.

- Furthermore, we propose a way to abstract necessary and sufficient diagnosability information from regional object, which we call pattern verifier in chapter 4, that is constructed from the subsystem where the pattern is completely recognized. Then the global consistency checking is based on the abstracted local objects to check pattern diagnosability. In this way, we avoid constructing global objects both for pattern recognition and for pattern diagnosability verification.

- Finally some important information about the reasons why the system is not pattern diagnosable is provided by our algorithm when the system is not diagnosable, which, to some extent, can help the designer to improve the diagnosability level of the system by rearranging sensor placement, reconfiguration, etc.

This pattern diagnosability algorithm is in a distributed way. Furthermore, it adopts the high level of abstraction from local objects as described in the section 1.2.1. Thus it fills up the gap of the third point in the section 1.1. The idea is to find an equivalent alternative to the centralized pattern diagnosability checking that is more efficient in order to improve the scalability of the problem. In chapter 4, we theoretically prove the correctness and the efficiency of this distributed algorithm. Then in chapter 5, we also implement and evaluate this algorithm with results consistent with the theoretical analysis.

1.2.3 Distributed diagnosability for systems with autonomous components

The crucial points of distributed diagnosability for systems with autonomous components are presented as the following.

- We first describe systems with autonomous components, where each observable event can only be observed by its own component. In other words, there is no global knowledge available about the system, like globally observable events. And then we define communication compatibility that is identical to reconstructibility in trace theory.

- Then we define joint diagnosability definition for systems with autonomous components, which is proved to be undecidable when communication events are unobservable. We then

give an algorithm to sufficiently but not necessarily test joint diagnosability. And then we provide another algorithm to test joint diagnosability in a decidable case, where communication events are assumed to be observable.

Clearly, this approach is to deal with the drawback of the fourth point in the section 1.1. In other words, here we consider that each component is autonomous, i.e., each component can only observe its own observable events and thus there is no global knowledge.

1.3 Organization

This thesis is organized as follows. In chapter 2, we review some existing methods of diagnosis and diagnosability, especially the diagnosability methods that are relative to ours. Then we describe how to optimize the existing distributed diagnosability by heightening the level of abstraction from local objects in chapter 3. And then the pattern diagnosability is extended from centralized framework to distributed one, including the high level of abstraction from local objects, which is described in chapter 4. Then in chapter 5, the implementation of distributed pattern diagnosability algorithm is presented with its test case. Since the case of single fault event is a special case of the pattern one, thus the implementation is also suitable for the approach described in chapter 3. Here we can see the search space is really reduced in practice. And then we define joint diagnosability before discussing about its undecidable case and decidable case and the corresponding algorithms are detailed in chapter 6. Finally conclusion and perspectives are presented in chapter 7.

Chapter 2

Diagnosis and diagnosability methodologies

In this section, we review some major diagnosis approaches and then modeling formalisms for DES before describing some important diagnosability algorithms for DES, which will help in understanding the contributions of this thesis.

2.1 Diagnosis approaches

In recent decades, the design and implementation of diagnosis systems have received considerable attention in the literature. Many approaches with different frameworks have been proposed. Most of the diagnosis approaches that rely on explicit knowledge (we do not consider here black-box approaches like statistical learning, etc.) can be divided into the following four classes:

- fault tree analysis;
- analytical redundancy methods;
- expert systems and knowledge-based methods;
- model-based reasoning methods.

2.1.1 Fault tree analysis

Fault tree analysis ([70], [49] and [50]) is top-down deductive analytical method where the effects of initiating faults and events on a complex system are analyzed. It can be qualitative or quanti-

tative, depending on whether fault event probabilities are unknown or known. To construct fault tree, the first step is to define the undesired event to study, which is taken as root of the fault tree. Then all causes (with probabilities) of the undesired event are studied and analyzed. Finally the fault tree is constructed based on AND and OR gates which define the major characteristics of the fault tree. From its construction, obviously, the fault tree is used to reason backwards until the root cause of the fault is found when an observation indicates an abnormality of a system. However, assembling a fault tree can be a costly and cumbersome experience and thus limits its applicability in practice. Moreover, a fault tree is used to analyze a single fault event, and that one and only one event can be analyzed during a single fault tree. In other words, one undesired event for one fault tree and no two undesired events will be used to make one fault tree.

2.1.2 Analytical redundancy methods

Most of the approaches for fault diagnosis proposed by Control Community are based on analytical redundancy techniques ([32], [69], [73]). The main principle is to use a residual, a symptom of process faults to facilitate the diagnosis tasks. Residuals are quantities that represent the inconsistency between the actual system variables and the mathematical model. In other words, residual signals are generated by comparing predicted values of system variables with the actual observed values, where the predicted values come from the available mathematical model of the system. And then the evaluation of the residuals for the likelihood of faults using for instance likelihood ratio functions can lead to the final decision and fault isolation. In other words, residuals are ideally zero and some residuals become non-zero if the actual system differs from the ideal one, which may be due to faults, disturbances, noise. However, the approaches based on analytical redundancy techniques are very sensitive to modeling errors and to complex problems of detailed on-line modeling of system behaviors.

2.1.3 Expert systems and knowledge-based methods

Expert system methods and knowledge-based methods [63] for fault diagnosis are mostly for systems that are difficult to model. The terms expert system and knowledge-based system are often used synonymously. In expert system methods, based on experience with the system, heuristic knowledge of experts is captured in a set of rules that efficiently associate the observations to the corresponding diagnoses. There are several drawbacks of expert system methods. First, it is quite difficult to acquire the expertise, which is only available after a long period of use of the system

in most cases. Then, when a previously unseen behavior occurs leading to undesired observation, it is impossible to decide a diagnosis. In other words, the acquired expert knowledge can never be guaranteed to be complete. Moreover, in the case where a very small modification is made on the system, the expert system must be constructed again.

2.1.4 Model-based reasoning methods

Model-based diagnosis ([45], [57], [3], [4], [13], [16], [19], [54], [27], [40], [46], [47], [65], [74], [2]) is based on an explicit behavioral model of the system to be diagnosed and offers a continuum from consistency-based reasoning to abductive reasoning. A behavioral model is a functional representation of the system where its behavior can be predicted purely from the internal states of the model and the values of the input variables. More precisely, a behavioral model of the system is composed of a list of the component models and of their connections. For each component model, it is characterized by a set of variables, a set of modes with ok mode and sometimes a set of fault modes and a set of relations to describe the behavior of the component in such a mode.

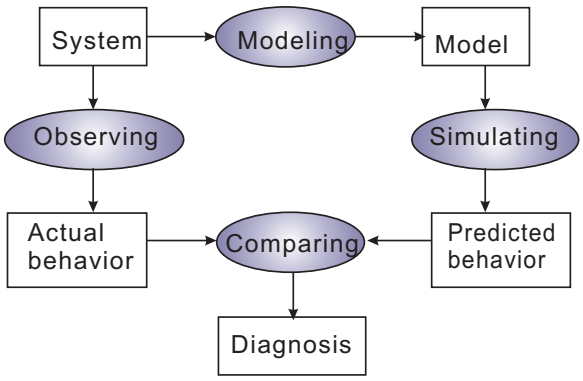

Figure 2.1: The procedure of model-based diagnosis.

Figure 2.1 depicts the principle of model-based diagnosis. Given observations of the system, i.e., actual behavior of the system, the model is used to simulate the system to gain the predicted behavior, and then the diagnosis can be obtained by comparing the actual behavior with the predicted behavior. The Artificial Intelligence approaches are normally based on qualitative models, where the domain of each variable is a finite set of values. Such an abstraction is simpler than quantitative models and has proven to be quite powerful for diagnostic purposes.

The main advantages of model-based diagnosis are its generality, reliability, flexibility and capability to explain diagnoses. For example, it decides diagnosis without dependance on infor-

mation about previously experienced faults, which is at the contrary the fundamental requirement of expert systems and knowledge-based methods. The difficulty lies in general in model acquisition.

2.2 Modeling formalisms for DES

Since DES is the type of systems studied in this thesis, we first look at its modeling formalisms. Many formalisms have been proposed to model DES, such as finite state machines (FSM) or automata ([61], [59], [48], [41]), Petri nets ([1], [33]), process algebra ([17], [18]) and so on. The most classical ones are FSM and Petri nets. They both use a state transition structure, that is, by specifying what the possible events are in each state of the system. This highlights structural information about the system behavior, which is convenient to manipulate when addressing analysis and controller synthesis issues. As far as distributed or very complex systems are concerned, the component oriented approach is clearly preferred. In other words, to model such a system, it is necessary to consider the system as a set of communicating components, where each component is modeled separately with its communication information. For the formalism of Petri nets, communication is represented by common places or transitions between several components. Thus to obtain the global model of the whole system, the different nets are merged over the shared places and transitions. As far as FSM are concerned, communication is modeled by common events, which are also called communication events. The global model is obtained by the synchronization of all the system components. In this thesis, we choose the FSM as the modeling formalism for DES for the sake of simplicity and its amenability to analysis for answering various questions about the behavior of DES.

2.3 Diagnosability for DES

In some cases, diagnosis decision could be necessarily ambiguous, and thus running a diagnosis engine does not make sense. So it is very important to decide at design stage how accurate any diagnosis algorithm can be on a given system based on system model. This problem is called diagnosability analysis and is the basic question that underlies diagnosis.

2.3.1 Centralized and distributed approaches

The approaches for diagnosability analysis can also be classified into centralized and distributed methods. For the former one, there is always a global system model from which the diagnosability property is tested directly or indirectly ([61], [44], [80], etc.). For distributed methods, there are two types. One is for systems where the information used for diagnosability analysis is centralized, i.e., the observations being globally observed. The other one is for systems where the information is distributed, i.e., several work stations having access to their own local observable information. In distributed approaches with centralized information such as [53], [52] and [64], a global model is implicitly defined as the synchronization of the set of system components. For each of these components, the local diagnosability information is computed and later combined to obtain the global diagnosability result. Due to the underlying global system model, all events emitted from the system are globally ordered, which allows reasoning about global dependencies among faults. For the latter, the distributed methods with distributed information ([30], [56] and [66]) assume that only the observations from the same subsystem, also referred to as work station or as site, are ordered. Mostly, each site has its own local diagnoser associated to it. This differs from the distributed approaches with centralized information, where there is a centralized coordination of the local diagnosability analysis. Actually, in the existing distributed approaches with distributed information, the system is not modeled as a set of communicating components, which means that the observations are distributed into different sites that are divided from an entire system, i.e. the monolithic model of the system is assumed. In comparison to centralized approaches, distributed ones require less space. In fact, due to the high space requirements of centralized methods, they can hardly be applied to large scale systems.

Next we describe the most popular centralized approaches for diagnosability analysis of DES to understand their essential idea. We will present relatively important distributed diagnosability approaches in the following chapters to compare with our proposed distributed approaches.

2.3.2 Centralized diagnosability for DES

Informally speaking, the existence of two indistinguishable behaviors, i.e., holding the same enough observations, with exactly one of them containing one given fault violates diagnosability property. The classical and centralized diagnosability analysis methods check the existence of such indistinguishable behaviors with the assumption that the knowledge about the system is a monolithic model.

Definition 1 *(System model). A system is modeled as a FSM, denoted by* $G = (Q, \Sigma, \delta, q^0)$*, where*

- Q *is a finite set of states;*
- Σ *is a finite set of events;*
- $\delta \subseteq Q \times \Sigma \times Q$ *is a finite set of transitions;*
- q^0 *is the initial state.*

The events set Σ is partitioned into three subsets: $\Sigma = \Sigma_o \biguplus \Sigma_u \biguplus \Sigma_f$, where Σ_o denotes the set of observable events, Σ_u denotes the set of unobservable normal events and Σ_f denotes the set of unobservable fault events. For the transition set, it is easy to extend $\delta \subseteq Q \times \Sigma \times Q$ to $\delta \subseteq Q \times \Sigma^* \times Q$ in the following way:

- $(q, \epsilon, q) \in \delta$, where ϵ is the null event;
- $(q, se, q1) \in \delta$ if $\exists q\prime \in Q, (q, s, q\prime) \in \delta$ and $(q\prime, e, q1) \in \delta$, where $s \in \Sigma^*, e \in \Sigma$.

Given a system model G, the prefix-closed language $L(G)$, which describes the normal and faulty behaviors of the system, is a subset of the Kleene closure of Σ: $L(G) \subseteq \Sigma^*$. Formally, the language $L(G)$ is the set of words produced by FSM G:

$$L(G) = \{s \in \Sigma^* | \exists q \in Q, (q^0, s, q) \in \delta\}.$$

Sometimes there is a set F of final states in the FSM. In such a FSM, we denote the marked language generated by G by:

$$L_m(G) = \{s \in L(G) | \exists q \in F, (q^0, s, q) \in \delta\}.$$

In the following, we call a word from $L(G)$ a trajectory in G and a sequence $q_0\sigma_0q_1\sigma_1...$ a path in G, where $\sigma_0\sigma_1...$ is a trajectory in G and for all i, we have $(q_i, \sigma_i, q_{i+1}) \in \delta$. Given $s \in L(G)$, we denote the post-language of $L(G)$ after s by $L(G)/s$, formally defined as: $L(G)/s = \{t \in \Sigma^* | s.t \in L(G)\}$. The projection of the trajectory s to observable events is denoted by $P(s)$. And the inverse projection of an observation sequence s, denoted by $P^{-1}(s)$, returns the set of all trajectories whose observable projection is s.

Two composition operations are defined as follows. For the sake of simplicity, they are presented for two deterministic FSMs. It is easy to generalize them for a set of FSMs using the associativity properties and nondeterministic FSMs can be composed with the same rules.

Definition 2 *(Synchronization). Given two FSMs* $G_1 = (Q_1, \Sigma_1, \delta_1, q_1^0)$ *and* $G_2 = (Q_2, \Sigma_2, \delta_2, q_2^0)$*, their synchronization is* $G_1 \|_{\Sigma_s} G_2 = (Q_1 \times Q_2,\ \Sigma_1 \cup \Sigma_2,\ \delta_{1\|2},\ (q_1^0, q_2^0))$*, where* $\Sigma_s = \Sigma_1 \cap \Sigma_2$ *is the set of shared events, which can be omitted when there is no ambiguity in the context, and* $\delta_{1\|2}$ *is defined as follows:*

- $((q_1, q_2), \sigma, (q_1', q_2')) \in \delta_{1\|2}$*, if* $\sigma \in \Sigma_s$*,* $(q_1, \sigma, q_1') \in \delta_1$ *and* $(q_2, \sigma, q_2') \in \delta_2$*;*
- $((q_1, q_2), \sigma, (q_1', q_2)) \in \delta_{1\|2}$*, if* $\sigma \in \Sigma_1 \backslash \Sigma_s$ *and* $(q_1, \sigma, q_1') \in \delta_1$*;*
- $((q_1, q_2), \sigma, (q_1, q_2')) \in \delta_{1\|2}$*, if* $\sigma \in \Sigma_2 \backslash \Sigma_s$ *and* $(q_2, \sigma, q_2') \in \delta_2$*.*

Definition 3 *(Product). Given two FSMs* G_1 *and* G_2*, their product is* $G_1 \times G_2 = (Q_1 \times Q_2,\ \Sigma_1 \cup \Sigma_2,\ \delta_{1\times 2},\ (q_1^0, q_2^0))$*, where* $\delta_{1\times 2}((q_1, q_2), \sigma) = (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))$ *if both* $\delta_1(q_1, \sigma)$ *and* $\delta_2(q_2, \sigma)$ *are defined in* G_1, G_2 *respectively. Otherwise,* $\delta_{1\times 2}((q_1, q_2), \sigma)$ *is undefined in* $G_1 \times G_2$*.*

Definition 4 *(Delay Closure). Given a FSM* $G = (Q, \Sigma, \delta, q^0)$*, its delay closure with respect to* Σ_d*, where* $\Sigma_d \subseteq \Sigma$*, is* $\complement_{\Sigma_d}(G) = (Q, \Sigma_d, \delta_d, q^0)$*, where* $\delta_d(q, \sigma) = q\prime$ *with* $\sigma \in \Sigma_d$ *if* $\exists s \in (\Sigma \backslash \Sigma_d)^*, \delta(q, s\sigma) = q\prime$ *in* G*.*

The operation of product is sometimes called complete synchronization. The main difference between the two operations is how the private events, i.e., the events not in $\Sigma_1 \cap \Sigma_2$, are handled. In the product, the transitions of the two FSMs must always be synchronized on a shared event, $\sigma \in \Sigma_1 \cap \Sigma_2$. In other words, an event in the product occurs iff it occurs in both FSMs. In the synchronization, the two FSMs are still synchronized on the shared events but the private events can independently be executed whenever possible. So if $\Sigma_1 = \Sigma_2$, then the synchronization reduces to product because all events are forced to be synchronized. The standard way of building models of entire systems from models of individual system components is by synchronization. As for the operation of delay closure with respect to Σ_d, we preserve the information about the events in Σ_d while abstracting away irrelevant parts.

Figure 2.2 depicts a simple system example, where the events Oi denote observable events, the event F denotes unobservable fault event, the events Ui denote unobservable normal events. Two assumptions are made on the system under investigation during the diagnosability analysis:

- The language of system $L(G)$ is live, which means that there is at least one transition defined at each state in Q.
- There does not exist any cycle of unobservable events in G.

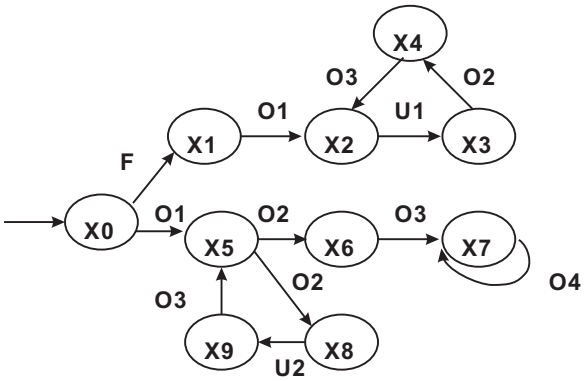

Figure 2.2: A system example.

A fault F is diagnosable in a system G iff its occurrence is determinable when enough events are observed from the system after the occurrence of F, which is formally defined as follows ([61]), where s^F denotes a trajectory in G ending with F.

Definition 5 *(Diagnosability). A fault F is diagnosable in a system G iff*

$$\exists k \in N, \forall s^F \in L(G), \forall t \in L(G)/s^F, |t| \geq k \Rightarrow$$
$$(\forall p \in L(G),\ P(p) = P(s^F.t) \Rightarrow F \in p).$$

The above definition states that for each trajectory s^F in G, for each t that is an extension of s^F in G with sufficient events, every trajectory p in G that is observation equivalent to $s^F.t$ should contain in it F. In other words, the (non-)diagnosability checking consists in searching for a pair of trajectories p and $p\prime$ satisfying the following conditions:

- p contains F and $p\prime$ does not;
- p has arbitrarily long observations after the occurrence of F;
- $P(p) = P(p\prime)$.

Such a pair is called a **critical pair** [15], which violates definition 5 and thus witnesses non-diagnosability. Next we will recall the most popular approaches to check diagnosability of DES in the centralized way. Since the purpose here is to illustrate the essential idea of these approaches, for the sake of simplicity, we assume that there is only one fault event, denoted by F, which can be directly extended to the case of a set of fault events.

2.3.2.1 Deterministic diagnoser approach

In the literature, the first way proposed to verify the diagnosability of DES is to construct a deterministic FSM, called a deterministic diagnoser ([61]). Before deterministic diagnoser construction, we show a nondeterministic generator construction based on system model, which we call pre-diagnoser in the following.

Definition 6 *(Pre-diagnoser). The pre-diagnoser of the system G is a FSM, denoted by $D = (Q_D, \Sigma_D, \delta_D, q_D^0)$ where*

- $Q_D \subseteq Q \times 2^{\Sigma_f}$ *is the set of states;*
- $\Sigma_D = \Sigma_o$ *is the set of events;*
- $\delta_D \subseteq Q_D \times \Sigma_D \times Q_D$ *is the set of transitions;*
- $q_D^0 = (q^0, \emptyset)$ *is the initial state.*

The transitions of δ_D are those $((q, \ell), e, (q\prime, \ell\prime))$ with (q, ℓ) reachable from the initial state q_D^0 and satisfying the following condition: there is a transition path $p = (q \xrightarrow{u_1} q_1 ... \xrightarrow{u_m} q_m \xrightarrow{e} q\prime)$ in G, with $u_k \in \Sigma_u, \forall k \in \{1, ..., m\}$, $e \in \Sigma_o$ and $\ell\prime = \ell \cup (\{u_1, ...u_m\} \cap \Sigma_f)$.

From the top part of figure 2.3, which is the pre-diagnoser of the system depicted in figure 2.2, we can see that it is designed to preserve all observable information from the original system model and to append to every state an estimate of failure information, which is called fault label. Then the deterministic diagnoser can be built from the pre-diagnoser, formally defined as follows:

Definition 7 *(Deterministic diagnoser). The deterministic diagnoser of the system G is a FSM, denoted by $D_d = (Q_{D_d}, \Sigma_{D_d}, \delta_{D_d}, q_{D_d}^0)$ where*

- $Q_{D_d} \subseteq 2^{Q_D}$ *is the set of states;*
- $\Sigma_{D_d} = \Sigma_o$ *is the set of events;*
- $\delta_{D_d} \subseteq Q_{D_d} \times \Sigma_{D_d} \times Q_{D_d}$ *is the set of transitions;*
- $q_{D_d}^0 = (q^0, \emptyset)$ *is the initial state.*

Since the state space Q_{D_d} is a subset of the powerset of Q_D, each state q_{D_d} of D_d is of the form: $q_{D_d} = \{q_d^1, ..., q_d^n\}$, where $q_d^k \in Q_D, \forall k \in \{1, ..., n\}$. For each transition $\delta_{D_d}(q_1, \sigma) = q_2$, q_2 is obtained as follows:

$$q_2 = \bigcup_{\{q_d | q_d \in q_1 \wedge \exists q'_d, \delta_D(q_d,\sigma)=q'_d\}} \{\delta_D(q_d, \sigma)\}$$

A deterministic diagnoser state, $q_{D_d} \in Q_{D_d}$, is called a F-certain state, if $\forall q_d = (q, \ell) \in q_{D_d}, F \in \ell$, or a certain normal state, if $\forall q_d = (q, \ell) \in q_{D_d}, \ell = \{\}$. If there exists $q_d^1 = (q, \ell), q_d^2 = (q\prime, \ell\prime) \in q_{D_d}$ such that $F \in \ell$ and $F \notin \ell\prime$, then q_{D_d} is called a F-uncertain state. A cycle in the deterministic diagnoser is called a F-indeterminate cycle if it satisfies the following conditions:

1. There exists a corresponding cycle in its pre-diagnoser involving only states whose fault label contains F;

2. There exists a corresponding cycle in its pre-diagnoser involving only states whose fault label does not contain F;

3. All the states in the cycle are F-uncertain states.

Clearly, the third condition is the result of the first and the second conditions. Each F-indeterminate cycle is corresponding to a critical pair with respect to F, which violates diagnosability property. So diagnosability verification consists in checking the existence of F-indeterminate cycles in the deterministic diagnoser of the system.

Theorem 1 *A fault F is diagnosable in a system G iff there is no F-indeterminate cycle in the deterministic diagnoser of G.*

The bottom part of figure 2.3 is the deterministic diagnoser of system depicted in figure 2.2. For all deterministic diagnoser states, we attach the top labels $S0, ..., S4$, which are used as their identifiers. In this diagnoser, the states $S1, S2, S3$ are F-uncertain states and $S0, S4$ are normal states. From the definition of F-indeterminate cycle, it is seen that the cycle containing the states $(S2, S3)$ is a F-indeterminate cycle since there is a corresponding cycle $(X2\{F\}, X4\{F\})$ in the pre-diagnoser (top part of figure 2.3) involving only states whose fault label contains F and there is a corresponding cycle $(X5\{\}, X8\{\})$ in the pre-diagnoser involving only states whose fault label is empty. So the fault F is not diagnosable in this system.

Afterward some studies for diagnosability analysis were based on this deterministic diagnoser approach. One recent is [11], where the intermittent sensor failures leading to loss of observability are taken into account. In other words, the authors of [11] assume the presence of intermittent sensor failures, which includes permanent sensor failures that can be viewed as a forever lasting intermittent failure. To consider the influence of intermittent sensor failures on the system behavior, the events set of the system is divided into three disjoint subsets: the set of observable events

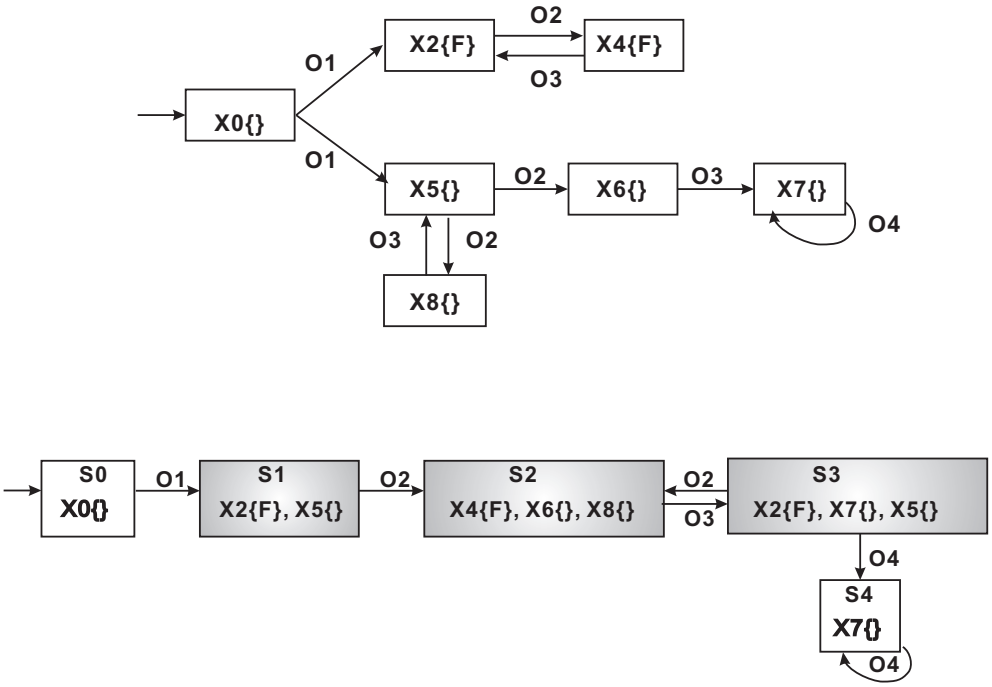

Figure 2.3: The pre-diagnoser (top) and the deterministic diagnoser (bottom).

associated with intermittent sensor failures; the set of observable events associated with sensors without failures and the set of unobservable events. A new language operation, called language dilation, is proposed to model systems of both normal behavior and subject to sensor failures. An algorithm to construct an automaton modeling the observed behavior of system with intermittent sensor failures from the original system model is described. Then robust diagnosability of DES subject to intermittent sensor failures is defined. Finally they show how to construct a robust diagnoser to verify robust diagnosability with the proposed necessary and sufficient condition.

2.3.2.2 Twin plant approach

The main drawback of deterministic diagnoser approach is its exponential space complexity in the number of system states. Then the authors of [44] proposed a new algorithm called twin plant method with polynomial complexity in the number of system states, based on the construction of nondeterministic automata and on the search for cycles with a given property.

In the twin plant method, given a system model G, first the pre-diagnoser is constructed based on G and then a twin plant is obtained by synchronizing the pre-diagnoser with itself based on the observable events to obtain all pairs of trajectories with the same observations. Since the events

set of pre-diagnoser is the set of observable events, then such a synchronization is equal to the product of the pre-diagnoser with itself.

Definition 8 ***(Twin Plant)*** *The twin plant of the system G, denoted by T, is the FSM: $T = D \times D$, where D is the pre-diagnoser of the system G.*

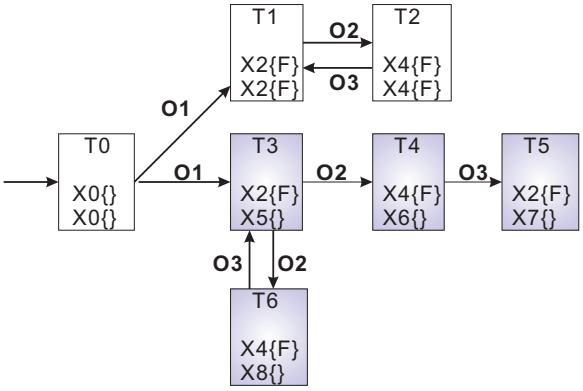

Figure 2.4: Part of the twin plant of system depicted in figure 2.2.

Each state of a twin plant is a pair of pre-diagnoser states that provide two possible diagnoses with the same observations. Given a twin plant state, if the fault F is contained in only one pre-diagnoser state, which means that the occurrence of F is not certain up to this twin plant state with the same observations, this twin plant state is called an ambiguous state with respect to F. An ambiguous state cycle is a cycle containing only ambiguous states. It has been proved that a path in the twin plant containing an ambiguous state cycle with at least one observable event corresponds to a critical pair in the system. We call this kind of path a critical path. So the diagnosability verification in the twin plant method is to check the existence of critical paths. From here we can see that the maximum states and the maximum transitions of the pre-diagnoser are $(|Q| \times 2^{|\Sigma_f|})$ and $(|Q|^2 \times 2^{2|\Sigma_f|} \times |\Sigma_o|)$, respectively. And the states and the transitions of the twin plant are at most $(|Q|^2 \times 2^{2|\Sigma_f|})$ and $(|Q|^4 \times 2^{4|\Sigma_f|} \times |\Sigma_o|)$. Thus the complexity of the twin plant method is $O(|Q|^4 \times 2^{4|\Sigma_f|} \times |\Sigma_o|)$ i.e. with polynomial complexity in the number of system states and exponential complexity in the number of faults.

Figure 2.4 depicts a part of twin plant T of the system G, where its state labels (top) are composed of a pair of pre-diagnoser state labels (middle and bottom). The gray nodes represent ambiguous states with respect to F, which form one ambiguous state cycle $(T3, T6)$. So the fault is not diagnosable in this system.

We have the following fundamental theorem for diagnosability verification [44].

Theorem 2 *A fault is diagnosable in a system G iff there is no critical path in the twin plant of G.*

2.3.2.3 Other approaches

Some people use process algebra as modeling formalism for diagnosability analysis. For instance, the authors of [17] propose to adopt a stochastic process algebra called Performance Evaluation Process Algebra (PEPA) for modeling physical systems and diagnosability verification without considering time issues. PEPA is a parsimonious language with only four combinators but is very expressive to define high-level complex systems. It can be used to study both quantitative properties and qualitative properties of the system. It has been demonstrated that it is easy to model a complex system as two parts with PEPA: a model of the behavior of each component type and a model of system structure. And then the semantics of PEPA model is represented by a Labeled Transition System (LTS), which describes all possible evolutions of the components individually as well as cooperatively. At the same time, the observations can be expressed as equations in terms of PEPA. Thus the diagnosis can be defined as an equation with system description and observations, all in terms of PEPA. More precisely, the diagnosis is the synchronization of system description and a set of observations based on the set of actions that the sensors can witness. Finally they rephrase the diagnosability definition and provide the necessary and sufficient condition to verify diagnosability in their very own framework by introducing d-equivalent paths, which are proved to correspond to the same minimal candidate diagnosis. One advantage of their approach is that there is no assumptions on which parameters can be observed. The observations are represented by algebraic expressions and simply synchronized with the system itself. Furthermore, with PEPA, diagnosis and diagnosability analysis can be automatically computed using a prototype tool called PEPA Workbench [35] that supports modeling and analysis with PEPA.

Another way to improve the efficiency of diagnosability analysis is described in [15], where the authors propose to reduce the diagnosability problem to a model checking problem. First, they define a critical pair as a pair of executions that are observationally indistinguishable but cause situations required to be distinguished, e.g. fault and normal behavior of the system. They model the system as a FSM. In their assumption, the input and output of the plant are observable and internal evolution of the plant is unobservable. Specifically, their system model is a structure $P = (Q, I, O, \delta, \lambda)$, where Q denotes the state space, I, O denote input space and output space respectively, $\delta \subset Q \times I \times Q$ is the transition relation and $\lambda \subset Q \times O$ is the observation relation.

Then very similarly to twin plant construction, they build a FSM called coupled twin plant from their system model and then they prove that a critical pair in the system is equivalent to a feasible execution in this coupled twin plant. Next the model checking technique is employed. First step is to associate to the system model a Kripke structure, which is a nondeterministic transition system, where each state is assigned a valuation to the state variables of the structure. The idea is that the state, input an output spaces of the system model can be encoded into the state space of the Kripke structure. Then in the same way, the Kripke structure corresponding to their coupled twin plant is constructed, denoted by K_{p^2}. The second step is to use the symbolic representation by defining K_{p^2} with a vector of variables. Atomic propositions over such variables are expressed. The third step is to characterize behaviors of the system over time through the temporal logics. In this way, the necessary and sufficient condition of diagnosability is expressed by Kripke structure with temporal logic formula, which can be verified directly by the maturely developed symbolic model checker.

The diagnosability analysis in the above approach is quite narrow in the sense that the delay is 1, which means that only one further event has to be observed before being certain that a failure has taken place. Then the authors of [58] propose another extensible approach to solve diagnosability testing by reducing it to the satisfiability problem of the classical propositional logic. In their framework, the system states are represented in terms of Boolean state variables and the relations corresponding to events in terms of changes to the values of the state variables. To improve the efficiency of SAT-based technique used for diagnosability analysis, the diagnosability definition with non-interfering simultaneous events is also discussed. In their logic formula, the events at each time point t are described with a parameter t. Then a formula to find a pair of infinite executions with the same observations but only one of them contains a failure is defined. They have proved that this formula is satisfiable iff the system is not diagnosable, which can be efficiently tested by SAT tools. One weakness of this SAT-based approach is that the diagnosability test is better suited to detect non-diagnosability than diagnosability since it is often easy to detect the presence of paths in transition systems but difficult to detect the absence of paths with a given property without length restrictions.

2.3.2.4 Pattern diagnosability for DES

All above approaches assume that the fault is a predefined event resulting in unexpected system behavior. However, sometimes the fault can be a sequence of some important events while any

single one of them is not the fault by itself. For example, the action of driving a car followed by the action of opening the car door without stopping driving between these two actions will cause an abnormal situation for the car, which should be considered as a fault. In this case, any single action is legal. A new proposal is provided by the authors of [43], who formally introduce the notion of supervision pattern, simply called pattern, that is general enough to cover an important class of diagnosis objectives, e.g. diagnosing multiple faults, repeating faults, sequences of significant events, etc. A fault event is a special case of pattern.

Since the pattern is actually the sequences of events, then in the system model, there is no fault event in the system events set.

Definition 9 *(Pattern). A pattern is a FSM with final states set* F_Ω, $\Omega = (Q_\Omega, \Sigma_\Omega, \delta_\Omega, q_\Omega^0, F_\Omega)$, *which satisfies the following conditions:*

- $\forall q \in Q_\Omega, \forall \sigma \in \Sigma_\Omega$, *if* $(q, \sigma, q_1) \in \delta_\Omega$ *and* $(q, \sigma, q_2) \in \delta_\Omega$, *then* $q_1 = q_2$
- $\forall q \in Q_\Omega, \Sigma_\Omega(q) = \Sigma_\Omega$ *where* $\Sigma_\Omega(q) = \{\sigma \in \Sigma_\Omega | \exists q\prime \in Q_\Omega, (q, \sigma, q\prime) \in \delta_\Omega\}$
- $F_\Omega \subseteq Q_\Omega$ *and* $\delta_\Omega(F_\Omega, \Sigma_\Omega) \subseteq F_\Omega$ *where* $\delta_\Omega(F_\Omega, \Sigma_\Omega) = \bigcup_{q \in F_\Omega, \sigma \in \Sigma_\Omega} \{q\prime \in Q_\Omega | (q, \sigma, q\prime) \in \delta_\Omega\}$

The first two conditions describe the pattern as a deterministic and complete FSM. The third condition characterizes that the final states set F_Ω is stable. Then it can be deduced that its marked language is "extension-closed", formally described as

$$\forall s \in L_m(\Omega), \forall s\prime \in \Sigma_\Omega^*, ss\prime \in L_m(\Omega)$$

which means that once the pattern arrives in a final state, it will be always in a final state in the future. Note that for all $s \in L_m(\Omega)$, $\exists e \in s$ such that e is unobservable, otherwise, the diagnosability problem with respect to the pattern would be trivial.

Given a system $G = (Q, \Sigma, \delta, q^0)$ and a pattern $\Omega = (Q_\Omega, \Sigma_\Omega, \delta_\Omega, q_\Omega^0, F_\Omega)$, we assume $\Sigma = \Sigma_\Omega$, $\Sigma_o = \Sigma_{\Omega_o}$, $\Sigma_u = \Sigma_{\Omega_u}$. A trajectory $s \in L(G)$ is recognized by Ω iff $s \in L_m(\Omega)$. The property of pattern diagnosability concerns the ability of a system to detect any trajectory recognized by a pattern with certainty, based on a sequence of observations. For example, figure 2.5 depicts an example of such a system and a pattern. Here in the pattern (bottom part) $\Sigma = \{U1, U2, O1, O2, O3\}$, which is the same events set as that of the system (top part). For both the system and the pattern, we have the set of unobservable events $\{U1, U2\}$ and the set of observable events $\{O1, O2, O3\}$. And the final states set of the pattern is $\{P2\}$. We can see that

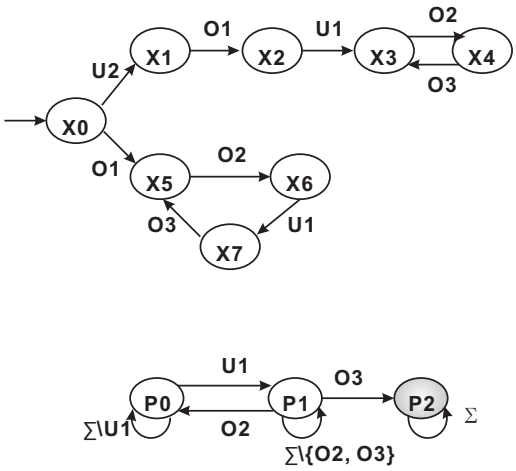

Figure 2.5: The system without fault event (top) and the considered pattern (bottom).

the pattern is actually the ordered occurrence of the events $U1, O3$ without the event $O2$ between them but can be with any other events between them.

Definition 10 *(Pattern Diagnosability). A pattern Ω is diagnosable in a system G, iff*

$$\exists n \in N, \forall s \in L(G) \cap L_m(\Omega), \forall t \in L(G)/s,$$
$$\textit{if } |t| \geq n, \textit{ then } P^{-1}P(s.t) \subseteq L_m(\Omega).$$

If Ω is diagnosable in G, then for any trajectory s in G that is recognized by the pattern, for any extension t of s with enough events, any trajectory with the same observations as $s.t$ is also recognized by the pattern. A critical pair p, $p\prime$ of system G with respect to the pattern Ω should satisfy the following conditions:

- $p \in L_m(\Omega)$ and $p\prime \notin L_m(\Omega)$;
- p is of arbitrarily long length after pattern recognition;
- $P(p) = P(p\prime)$.

The existence of such a critical pair states that Ω is not diagnosable in G. So similar to the case of fault event, pattern diagnosability checking is to search for critical pairs. The idea in [43] is to reuse twin plant method in the pattern case with some modifications.

Definition 11 *(Pattern Recognizer). Given a system $G = (Q, \Sigma, \delta, q^0)$ and a pattern $\Omega = (Q_\Omega, \Sigma_\Omega, \delta_\Omega, q^0_\Omega, F_\Omega)$, then the pattern recognizer of G is $R_G = (Q_{R_G}, \Sigma_{R_G}, \delta_{R_G}, q^0_{R_G}, F_{R_G}) = G \times \Omega$, where the initial state is $q^0_{R_G} = (q^0, q^0_\Omega)$, $F_{R_G} = (Q \times F_\Omega) \cap Q_{R_G}$ is the set of final states.*

Since Ω is a complete FSM, we have $L(\Omega) = \Sigma^*$ and thus $L(R_G) = L(G) \cap L(\Omega) = L(G)$. So the pattern recognizer shows which part of the pattern can be recognized after any trajectory in the system. The top part of figure 2.6 shows the pattern recognizer for the system and the pattern depicted in figure 2.5. Each state of the recognizer is composed of two parts with the left part being the system state and the right part being the pattern state. The set of three recognizer states $((X5, \{P2\}), (X6, \{P2\}), (X7, \{P2\}))$ is the set of final states, which forms a cycle containing only final states.

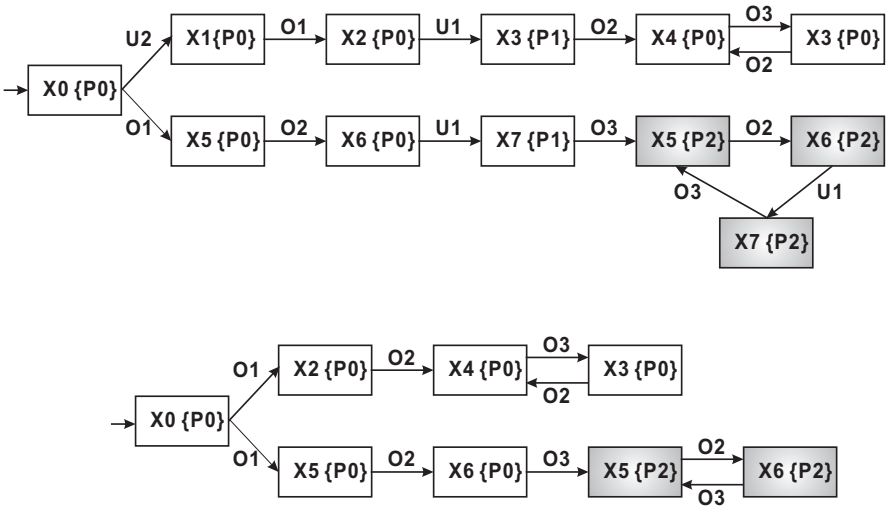

Figure 2.6: The pattern recognizer R_G (top) and its $\mathsf{C}_{\Sigma_o}(R_G)$ (bottom).

Before using the twin plant method, for the sake of simplicity, the delay closure with respect to the set of observable events is performed on the pattern recognizer: $\mathsf{C}_{\Sigma_o}(R_G)$. The bottom part of figure 2.6 is the result of performing this delay closure on the pattern recognizer shown in the top part of figure 2.6. Then for the sake of clarity, we rename the twin plant in the pattern case as the pattern verifier.

Definition 12 *(Pattern verifier). Given a pattern recognizer R_G, the corresponding pattern verifier, denoted by V, is obtained by $V = \mathsf{C}_{\Sigma_o}(R_G) \times \mathsf{C}_{\Sigma_o}(R_G)$.*

Similar to twin plant defined in definition 8, each state of V is a pair of pattern recognizer states that provide two possible pattern recognitions with the same observations. Given a pattern verifier state, if it has only one pattern recognizer state that is a final state, which means that the occurrence of the pattern is not certain up to this verifier state with the same observations, then this verifier state is called an ambiguous state. An ambiguous state cycle is a cycle containing

only ambiguous states. Then a path in V containing an ambiguous state cycle corresponds to a critical pair in the system, which is called a pattern critical path, simply critical path if there is no ambiguity in the context. So the pattern diagnosability verification consists in checking the existence of critical paths in the pattern verifier.

Theorem 3 *A pattern is diagnosable in a system G iff there is no critical path in the pattern verifier of G.*

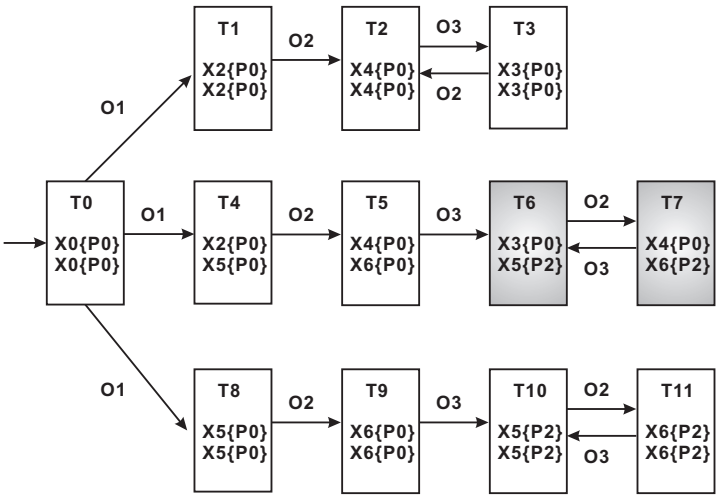

Figure 2.7: Part of twin plant for pattern recognizer.

Figure 2.7 depicts a part of the pattern verifier constructed from the pattern recognizer shown in the bottom part of figure 2.6, where its state labels (top) are composed of a pair of pattern recognizer state labels (middle and bottom). The gray nodes represent ambiguous states, which form one ambiguous state cycle $(T6, T7)$. So the pattern is not diagnosable in this system.

Chapter 3

Optimized diagnosability algorithm through abstraction

The classical and centralized diagnosability checking methods are to check the existence of critical pairs with the assumption that the knowledge about the system is a monolithic model. This hypothesis is normally unrealistic when dealing with real complex systems due to the combinatorial explosion of the search space. So we propose here a new formal framework for checking diagnosability of distributed DES, where the problem is described as a distributed search problem to avoid calculating global objects.

Our proposed approach makes several contributions to the diagnosability problem. First, we gear classical diagnosability definition for an entire system to regional diagnosability for a subsystem, which leads to defining a diagnosable subsystem. And we describe how to improve diagnosis algorithm in terms of observation reduction with a given diagnosable subsystem in a formal way. Second, we provide a new distributed theoretical framework to check regional diagnosability as well as diagnosability of the whole system. Instead of performing diagnosability verification on the global twin plant or local twin plant, i.e., searching for critical paths, we abstract necessary and sufficient diagnosability information from the local twin plant and then perform the search for local critical paths on the abstracted one before checking their global consistency. Our algorithm is optimized in the sense that with the abstracted diagnosability information, the search space is reduced to be as small as possible. Third, the diagnosability results we obtain can possibly help in the improvement of diagnosis algorithm when the system is diagnosable, in which case the algorithm returns a diagnosable subsystem. Otherwise, the algorithm provides some helpful information about indistinguishable behaviors that can be used to upgrade the diagnosability level of

the system when the system is verified to be not diagnosable.

3.1 Distributed system model

We consider a distributed DES composed of a set of components $G_1, ..., G_n$ that can communicate with each other by communication events. Such a system is modeled by a set of FSMs, each of them modeling one component.

Definition 13 ***(Local Model)*** *A component G_i is modeled as a FSM, denoted by $G_i = (Q_i, \Sigma_i, \delta_i, q_i^0)$, where*

- *Q_i is the set of states;*
- *Σ_i is the set of events;*
- *$\delta_i \subseteq Q_i \times \Sigma_i \times Q_i$ is the set of transitions;*
- *q_i^0 is the initial state.*

The set of events Σ_i is divided into four disjoint parts: Σ_{i_o}, the set of observable events in G_i, Σ_{i_f}, the set of unobservable fault events in G_i, Σ_{i_u}, the set of unobservable normal events in G_i and Σ_{i_c}, the set of unobservable communication events in G_i that are shared by at least one other component. Then we have $\Sigma_i = \Sigma_{i_o} \biguplus \Sigma_{i_f} \biguplus \Sigma_{i_u} \biguplus \Sigma_{i_c}$. And for any two different components G_i and G_j, we have $(\Sigma_i \backslash \Sigma_{i_c}) \cap (\Sigma_j \backslash \Sigma_{j_c}) = \emptyset$. In other words, the only shared events between different components are unobservable communication events. Actually here we implicitly have two assumption about communication events, which can be relaxed, as it will presented in section 3.5.

Assumption 1 *Any communication event is unobservable.*

Assumption 2 *Any communication event is correct.*

The first assumption indicates that our case is the most difficult one for distributed systems in terms of observability. In other words, our analysis can become much easier if the communication events are observable. While the second one is for the sake of simplicity and understandability, which is considered as a constraint here but can be relaxed in a straightforward way, which will be discussed in section 3.4.

The global model of the entire system is implicitly defined as the synchronized FSM of all component models based on their shared events, here communication events, denoted by $G = \|_{i=1}^{n} G_i$. The synchronized FSM on any non-empty set $\{G_{i_1}, ...G_{i_m}\}$ is called a subsystem of the system G, denoted by G_S, where $G_{i_k}, k \in \{1, ...m\}$ could be any component in G. Figure 3.1 depicts a distributed system composed of three components G_1 (top), G_2 (middle) and G_3 (bottom), where the events Oi denote observable events, the events Fi denote unobservable fault events, the events Ui denote unobservable normal events and the events Ci denote the unobservable communication events.

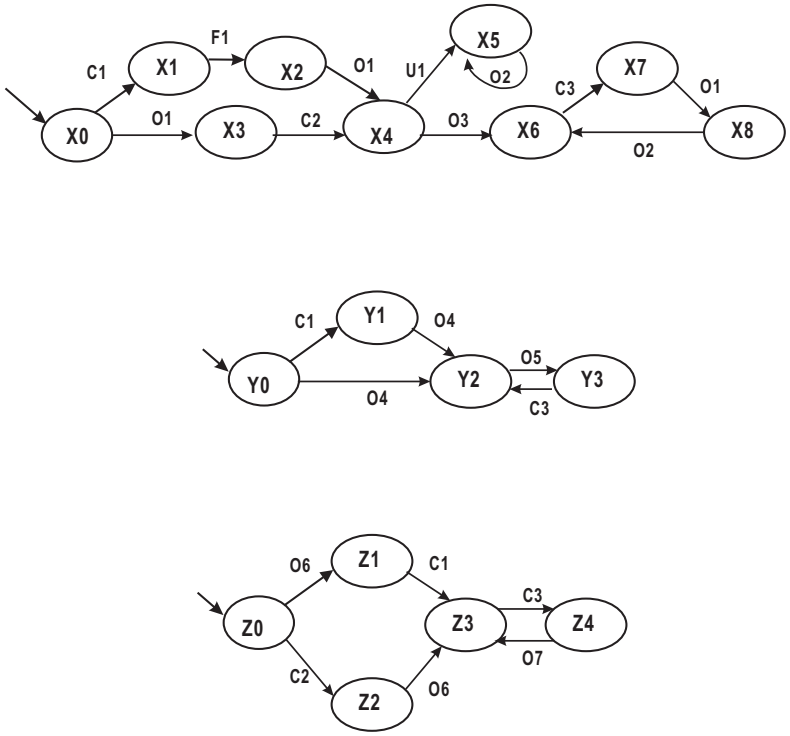

Figure 3.1: A system with three components G_1 (top), G_2 (middle) and G_3 (bottom).

In the distributed framework, we have a similar assumption to that described in [61], which holds in the whole thesis.

Assumption 3 *Each component projection of the global language is both live and observable live.*

This assumption means that for each global trajectory, its projection on each component is live without unobservable cycle.

3.2 Local twin plant and local twin checker

The basic idea of local twin plant, similar to that of twin plant described in [44] but for one component, is to build a FSM that compares every pair of local trajectories with the same observations to search for pairs of local trajectories with the same enough local observations but only one of them contains the fault. We first show how to construct the local pre-diagnoser from a given component model, which in turn serves to compute its corresponding local twin plant. The only difference between local pre-diagnoser and pre-diagnoser defined in definition 6 is that for local pre-diagnoser, we retain the information about communication events besides that about observable events, which will serve to check global consistency. While this is not the case for pre-diagnoser since in a centralized way, it is supposed to have a monolithic model for the whole system and thus the global consistency is implicitly guaranteed.

Definition 14 ***(Local pre-diagnoser)*** *The local pre-diagnoser of the component G_i is the FSM $D_i = (Q_{D_i}, \Sigma_{D_i}, \delta_{D_i}, q^0_{D_i})$ where*

- $Q_{D_i} \subseteq Q_i \times 2^{\Sigma_{i_f}}$ *is the set of states*
- $\Sigma_{D_i} = \Sigma_{i_o} \cup \Sigma_{i_c}$ *is the set of events*
- $\delta_{D_i} \subseteq Q_{D_i} \times \Sigma_{D_i} \times Q_{D_i}$ *is the set of transitions*
- $q^0_{D_i} = (q^0_i, \emptyset)$ *is the initial state*

The transitions of δ_{D_i} are those $((q, q_f), e, (q\prime, q_f\prime))$ with (q, q_f) reachable from the initial state $q^0_{D_i}$ and satisfying the following condition:

- there is a transition sequence $p = (q \xrightarrow{uo_1} q_1 ... \xrightarrow{uo_m} q_m \xrightarrow{e} q\prime)$ in G_i with $uo_k \in \Sigma_{i_u} \cup \Sigma_{i_f}, \forall k \in \{1, ..., m\}$, $e \in \Sigma_{i_o} \cup \Sigma_{i_c}$ and $q_f\prime = q_f \cup (\{uo_1, ...uo_m\} \cap \Sigma_{i_f})$.

Without loss of generality, we give the definition of local pre-diagnoser for the set of faults in the component G_i, which is perfectly suitable to deal with single fault when we run our algorithm each time for one fault. A local pre-diagnoser shows all possible faults after any local sequence of observable events and communication events. Figure 3.2 presents the local pre-diagnoser of the component G_1. The corresponding local twin plant is obtained by synchronizing the local pre-diagnoser with itself based on the set of observable events to obtain all pairs of local trajectories with the same observations. The two identical pre-diagnosers are denoted by D^l_i (left instance) and

D_i^r (right instance). Since this synchronization is based on the set of observable events and with assumption 1 that communication events are unobservable, the non-synchronized communication events are distinguished between the two instances by the prefix L and R: in D_i^l (D_i^r), each communication event $c \in \Sigma_{i_c}$ from D_i is renamed by $L : c$ ($R : c$) and all their observable events do not change their name.

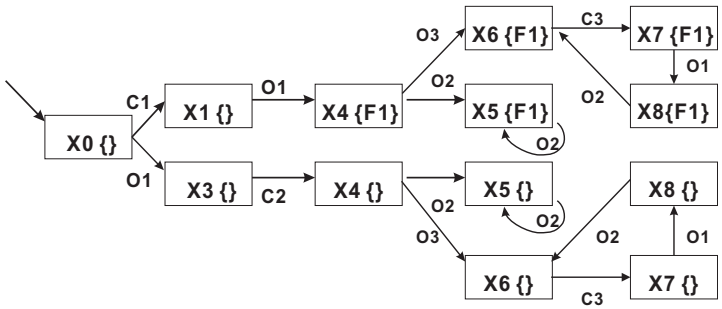

Figure 3.2: The local pre-diagnoser D_1 of component G_1 (see figure 3.1).

Definition 15 ***(Local Twin Plant)*** *The local twin plant of the component G_i is the FSM $T_i = D_i^l \| D_i^r$.*

From definition 15, we can see that the construction of local twin plant is different from that of twin plant defined in definition 8. The latter is constructed by the product of pre-diagnoser with itself based on the set of observable events. Different from pre-diagnoser, a local pre-diagnoser contains the communication events, which are not the synchronized events during local twin plant construction, where the only synchronized events are the observable events. While in pre-diagnoser there are only observable events. As said in section 2.3.2, the operation of synchronization reduces to the operation of product when the two FSMs have the same set of events and all events are the synchronized events. Thus the twin plant defined in definition 8 can also be expressed by the synchronization of the pre-diagnoser with itself based on its whole set of events.

Each state of a local twin plant T_i is a pair of local pre-diagnoser states that provide two possible diagnoses with the same local observations. Given a local twin plant state $((q^l, q_f^l)(q^r, q_f^r))$, if the fault $f \in q_f^l \cup q_f^r$ but $f \notin q_f^l \cap q_f^r$, which means that the occurrence of f is not certain up to this state with the same local observations, then this local twin plant state is called an ambiguous state with respect to f. An ambiguous state cycle is a cycle containing only ambiguous states. Figure 3.3 depicts a part of local twin plant T_1 of the component G_1, where each state label (top) is composed of a state label of D_i^l (middle) and that of D_i^r (bottom). The gray nodes represent

ambiguous states with respect to $F1$, which form ambiguous state cycles. In a local twin plant, if a path contains at least one ambiguous state cycle and from assumption 3, this cycle contains at least one observable event for this component, then it is called a **local possible critical path**, **LPCP** for short. Actually a LPCP corresponds to a pair of local trajectories in the local pre-diagnoser that have the same enough local observations but only one of them contains the fault, which is called a **local critical pair** in the following. If a LPCP is globally consistent (which will be described in next sections), which means that its corresponding local critical pair can be extended to be a global critical pair after synchronizing with other components.

We know that the function of local twin plant is to obtain all LPCPs. From figure 3.3, we can see that a local twin plant constructed as described above (see [53]) normally has a large redundant part which is useless, e.g. all paths without ambiguous state cycles. Furthermore, from the way to construct local twin plant, it can be seen that in the local twin plant, any local critical pair in the local pre-diagnoser has more than one corresponding LPCPs. In other words, there could be several LPCPs that correspond to the same pair of local trajectories in the local pre-diagnoser. For example, figure 3.4 shows a part of figure 3.3. We can see that two LPCPs $(R{:}C1.O1.L{:}C2.O2.O2^*), (L{:}C1.O1.R{:}C2.O2.O2^*)$ correspond to the same pair of local trajectories in the local pre-diagnoser (see figure 3.2)$(C1.O1.O2.O2^*)$, $(O1.C2.O2.O2^*)$ and another two $(R{:}C1.O1.L{:}C2.O3.(R{:}C3.L{:}C3.O1.O2)^*)$, $(L{:}C1.O1.R{:}C2.O3.(L{:}C3.R{:}C3.O1.O2)^*)$ correspond to the same pair of trajectories in the local pre-diagnoser$(C1.O1.O3.(C3.O1.O2)^*)$, $(O1.C2.O3.(C3.O1.O2)^*)$. So to ameliorate the redundant calculation, we can improve the local twin plant construction by reducing the left instance D_i^l and the right instance D_i^r. We reduce D_i^l constructed as above by only retaining the paths with at least one fault state cycle, i.e., there exists at least one cycle in this path of the local pre-diagnoser that contains a state (q, q_f), where $f \in q_f$ and f is the considered fault. Then D_i^r is reduced by only retaining the paths with at least one cycle without fault state. Then the improved local twin plant is constructed by synchronizing the reduced D_i^l and the reduced D_i^r as defined in definition 15, which is called optimized local twin plant in the following. Figure 3.5 depicts the reduced D_1^l (left) and the reduced D_1^r (right), where the gray nodes represent fault state in the local pre-diagnoser.

Lemma 1 *The set of LPCPs in the local twin plant corresponds to the same set of local critical pairs in the local pre-diagnoser as the set of LPCPs in the improved local twin plant does.*

Proof :

As described before, the local twin plant is constructed by synchronizing the non-reduced left

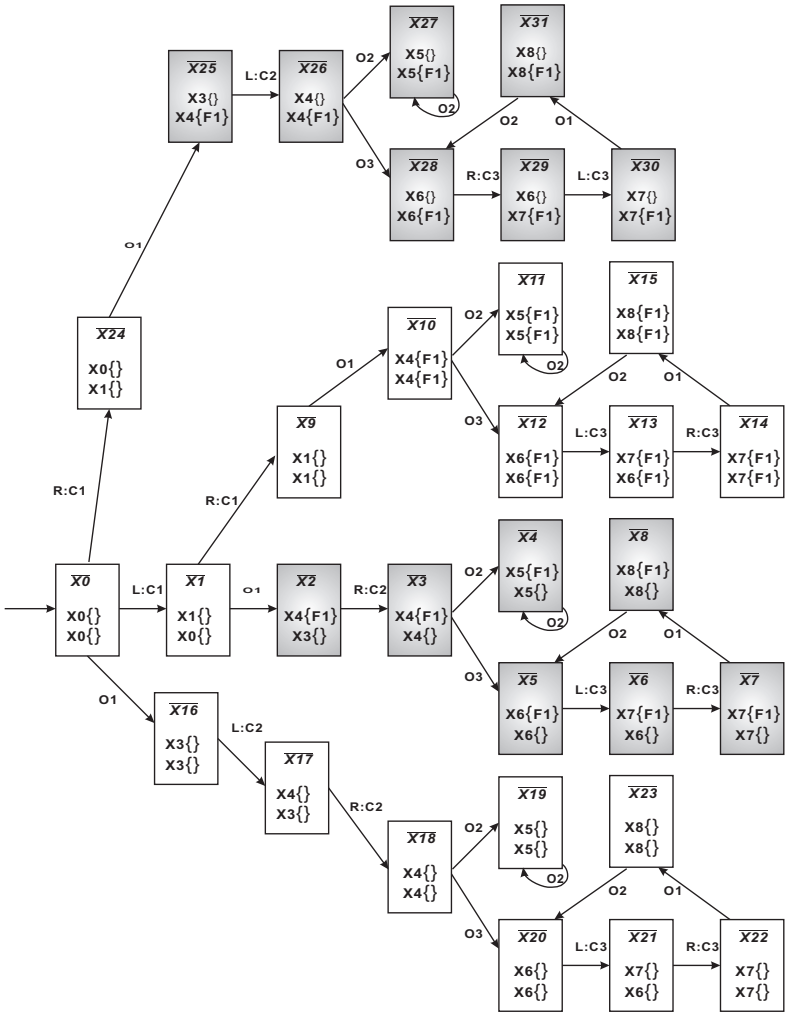

Figure 3.3: Part of local twin plant T_1 of component G_1 (see figure 3.1).

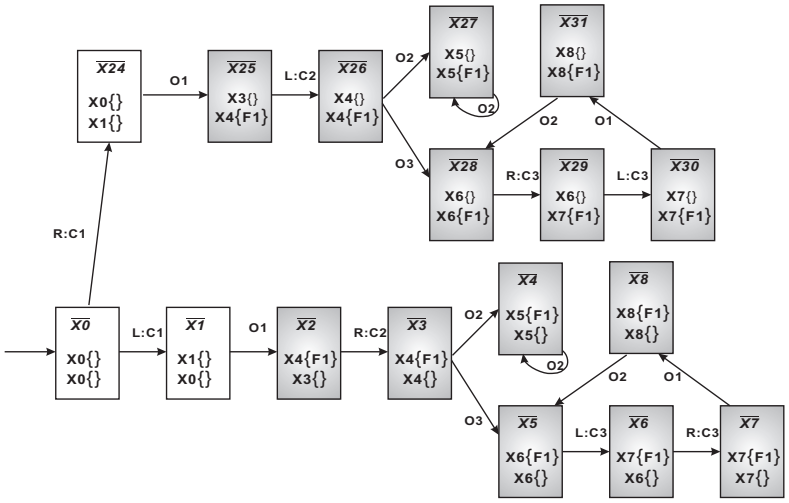

Figure 3.4: Part of local twin plant T_1 of component G_1 (part of figure 3.3).

instance D_i^l and the non-reduced right instance D_i^r while the optimized local twin plant is constructed by synchronizing the reduced left instance D_i^l and the reduced right instance D_i^r. For the sake of clarity, in the non-reduced D_i^l, we denote the set of paths with only fault state cycles by Λ_f^l, and denote the set of paths without fault state cycle by $\Lambda_{\neg f}^l$, and then denote the set of paths with both fault state cycles and cycles without fault state by Λ_{both}^l. In the same way, in the non-reduced D_i^r, we denote the set of paths with only fault state cycles by Λ_f^r, denote the set of paths without fault state cycle by $\Lambda_{\neg f}^r$, and denote the set of paths with both fault state cycles and cycles without fault state by Λ_{both}^r. Then the local twin plant is constructed by $(\Lambda_{\neg f}^l \cup \Lambda_f^l \cup \Lambda_{both}^l) \parallel (\Lambda_{\neg f}^r \cup \Lambda_f^r \cup \Lambda_{both}^r)$. On the other hand, the reduced D_i^l retains the paths with at least one fault state cycle and the reduced D_i^r retains the paths with at least one cycle without fault state. Then the optimized local twin plant is constructed by $(\Lambda_f^l \cup \Lambda_{both}^l) \parallel (\Lambda_{\neg f}^r \cup \Lambda_{both}^r)$. The local twin plant construction can also be expressed by the addition of the synchronized results of nine cases: 1) $(\Lambda_{\neg f}^l) \parallel_{\Sigma_{i_o}} (\Lambda_{\neg f}^r)$; 2)$(\Lambda_{\neg f}^l) \parallel_{\Sigma_{i_o}} (\Lambda_f^r)$; 3)$(\Lambda_{\neg f}^l) \parallel_{\Sigma_{i_o}} (\Lambda_{both}^r)$ 4)$(\Lambda_f^l) \parallel_{\Sigma_{i_o}} (\Lambda_{\neg f}^r)$; 5) $(\Lambda_f^l) \parallel_{\Sigma_{i_o}} (\Lambda_f^r)$; 6) $(\Lambda_f^l) \parallel_{\Sigma_{i_o}} (\Lambda_{both}^r)$; 7)$(\Lambda_{both}^l) \parallel_{\Sigma_{i_o}} (\Lambda_{\neg f}^r)$; 8) $(\Lambda_{both}^l) \parallel_{\Sigma_{i_o}} (\Lambda_f^r)$ and 9) $(\Lambda_{both}^l) \parallel_{\Sigma_{i_o}} (\Lambda_{both}^r)$. And in the same way, the optimized local twin plant construction can also be expressed by the addition of the synchronized results of above four cases, which are actually (case 4 + case 6 + case 7+ case 9). So compared to the optimized local twin plant, the local twin plant has five more synchronized results (case 1 + case 2 + case 3 + case 5 + case 8). Now consider case 4 and case 2, which are actually symmetrical. We can see that the part in left instance of case 2 is the same as the part in right instance of case 4 and the part in right instance of case 2 is the

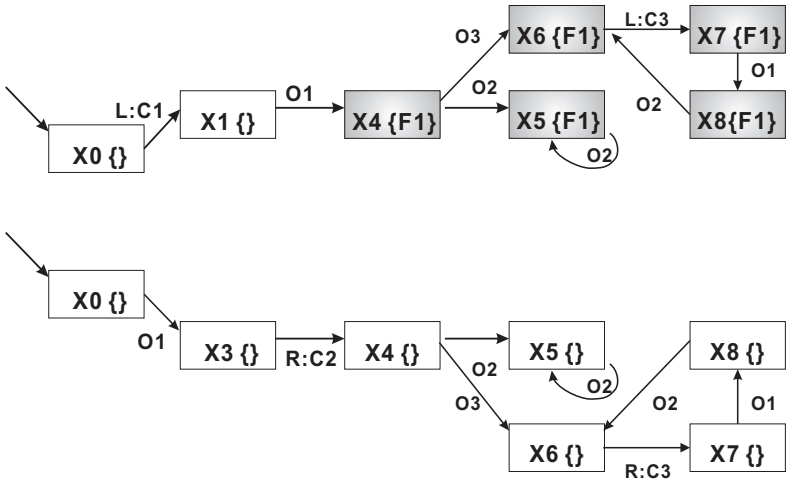

Figure 3.5: The reduced D_1^l(top) and the reduced D_i^r (bottom) (see figure 3.2).

same as the part in left instance of case 4. It is easy to prove that the synchronized result of case 2 corresponds to the same set of local trajectory pairs in the local pre-diagnoser as the synchronized result of case 4 does. In the same way, case 6 has the same result as case 8 and case 7 has the same result as case 3. Now the local twin plant has two more synchronized results (case 1 + case 5) than the optimized local twin plant. However, case 1 and case 5 can never get any LPCP. The reason is that in case 1, any path in $\Lambda^l_{\neg f}$ and in $\Lambda^r_{\neg f}$ has no fault state cycle, then the synchronized result has no ambiguous state cycle, which means that there is no LPCP. And in case 5, any path in Λ^l_f and in Λ^r_f has only fault state cycles, which means that there is no cycle without fault state. So their synchronization cannot obtain LPCP. Now we can say that the local twin plant corresponds to the same set of local trajectories in the local pre-diagnoser as the optimized local twin plant does, which proves lemma 1.

The part of local twin plant depicted in figure 3.3 is reduced to the part of optimized local twin plant shown in figure 3.7. So the state space of optimized local twin plant is much smaller than local twin plant but contains all local critical pairs. In our next sections, we calculate optimized local twin plant instead of local twin plant.

Next we define the local twin checker for a given component. First we operate delay closure on the component model, which is denoted by $G_{i\bullet} = \mathbb{C}_{\Sigma_{i_d}}(G_i)$, where Σ_{i_d} is the set of communication events and observable events of G_i. Then the local twin checker is obtained by synchronizing $G_{i\bullet}$ with itself based on the observable events. The idea is to obtain all pairs of local trajectories with the same observations. The two identical instances are denoted by $G^l_{i\bullet}$ (left instance) and

G_i^r• (right instance). Since this synchronization is based on the set of observable events and with assumption 1, the non-synchronized communication events are distinguished between the two instances by the prefix of L and R: in G_i^l• (G_i^r•), each communication event $c \in \Sigma_{i_c}$ from G_i• is renamed by $L : c$ ($R : c$). The difference between local twin plant and local twin checker is that local twin checker has no fault information while local twin plant provides fault information. Figure 3.6 depicts a part of the local twin checker of the component G_2 and a part of the local twin checker of the component G_3, where there is no fault information. Each path corresponds to a pair of local trajectories with the same observations.

Definition 16 ***(Local Twin Checker)*** *The local twin checker of the component G_i is the FSM $C_i = G_i^l \bullet \| G_i^r \bullet$.*

In the following, we denote the component where the fault f may occur by G_f. Note that the local twin plant is only constructed for the component G_f since the fault occurs only in this component. In the synchronized FSM of the local twin plant for the component G_f and a set of local twin checkers based on their communication events (left communication events synchronized with left ones and right communication events synchronized with right ones), $(\|_{i=1}^{m} C_{s_i}) \| T_f, s_i \in (\{1, ..., n\} \backslash \{f\}), \forall i \in \{1, ..., m\}$, any state is composed of one local twin plant state and a set of local twin checker states $q^t = (q_{T_f}^t, q_{C_{s_1}}^t, ..., q_{C_{s_m}}^t)$, where $q_{C_{s_i}}^t$ represents a state of the local twin checker C_{s_i} and $q_{T_f}^t$ is a state of the local twin plant T_f. If $q_{T_f}^t$ in q^t is an ambiguous state, then q^t is called an ambiguous state. Then we define global twin plant as follows.

Definition 17 ***(Global Twin Plant)*** *The global twin plant of a system with components $G_1, ..., G_n$ is the FSM $T = (\|_{i=1}^{(} n - 1) C_{s_i}) \| T_f, s_i \in (\{1, ..., n\} \backslash \{f\}), \forall i \in \{1, ..., n-1\}$.*

The global twin plant is defined by synchronizing the local twin checkers of all components except G_f and the local twin plant of G_f. Recall that in the centralized approach described in section 2.3.2.2, a path in the twin plant containing at least one ambiguous state cycle with at least one observable event is called a critical path. While in our approach, with the assumption 3, we have the following definition for global critical path.

Definition 18 ***(Global Critical Path)*** *Given the global twin plant, a path is called a global critical path if it contains at least one ambiguous state cycle ϕ such that $\forall i \in \{1, ..., n\}, \Sigma_\phi \cap \Sigma_{i_o} \neq \emptyset$, where Σ_ϕ denotes the set of events in the cycle ϕ.*

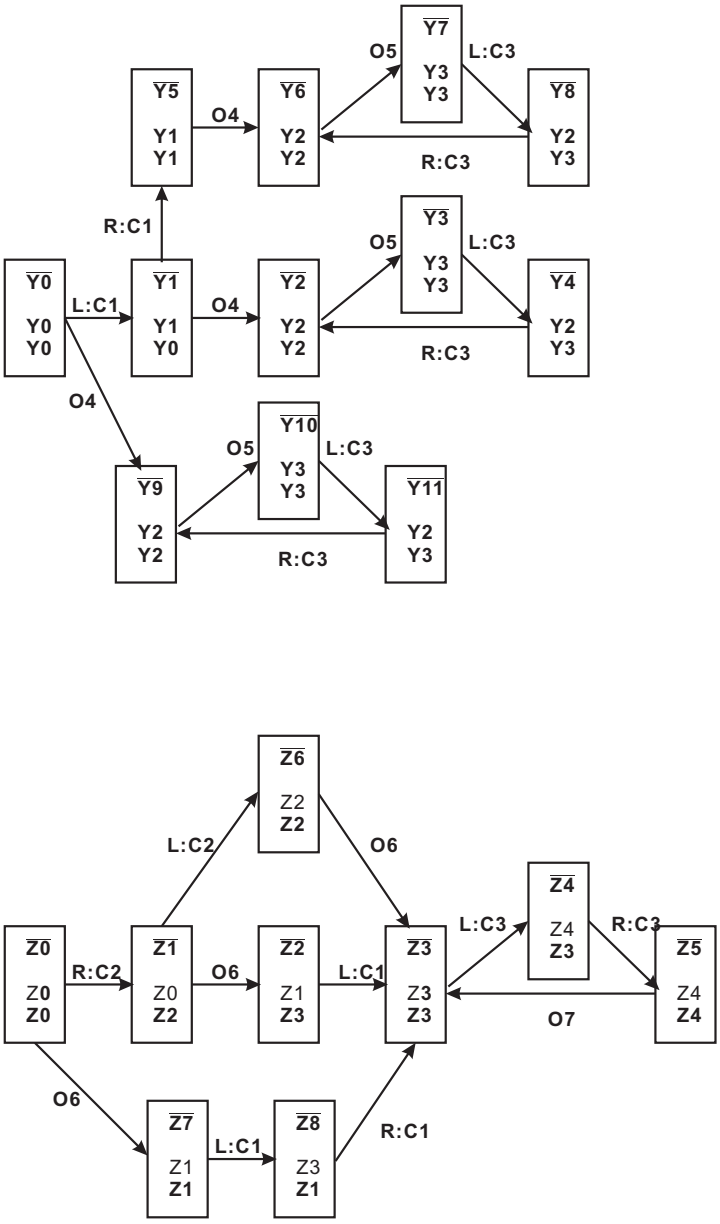

Figure 3.6: Part of local twin checker C_2 for G_2 and part of local twin checker C_3 for G_3.

In the global twin plant, if a path possesses an ambiguous state cycle containing at least one observable event for all components, then this path is called a global critical path, whose existence verifies non-diagnosability. Actually if a path in the global twin plant only possesses an ambiguous state cycle containing observable events for some but not all components, from the assumption 3, it can be deduced that its corresponding pair of trajectories must be blocked when synchronizing with the components that have no observable event in this ambiguous state cycle.

3.3 Distributed Framework

The non existence of global critical paths verifies diagnosability property of a system. The distributed idea is to decide diagnosability without necessarily building global objects, whose construction can be very computationally demanding for large and complex systems. In this section, we first define regional diagnosability for a subsystem in a distributed system and then show how to abstract necessary and sufficient diagnosability information and how to check regional diagnosability based on the abstracted version of local twin plant as well as those of local twin checkers with as small search space as possible.

3.3.1 Regional Diagnosability

For a distributed system, definition 5 can be geared to be suitable for a subsystem G_S containing a subset of components, which is called regional diagnosability. Let Σ_S denote the events set of the subsystem G_S, $P_S(p)$ denote the projection of the trajectory p to observable events in the subsystem G_S and s^f denote a trajectory ending with f. We formally define regional diagnosability as follows.

Definition 19 ***(Regional Diagnosability)*** *A fault f is regionally diagnosable in a system G with respect to a subsystem G_S, where $f \in \Sigma_S$, iff*

$$\exists k \in N, \forall s^f \in L(G), \forall t \in L(G) \backslash s^f, |P_S(t)| \geq k \Rightarrow$$
$$(\forall p \in L(G),\ P_S(p) = P_S(s^f.t) \Rightarrow f \in p).$$

If f is regionally diagnosable in G with respect to G_S, then G is called a f^{G_S}-diagnosable system and G_S is called a diagnosable subsystem with respect to f. In such a system, we are sure that f has effectively occurred in G_S when we observe enough events from G_S after the occurrence of f. Thus the observations from G_S are sufficient for diagnosis decision with respect to the fault

f, denoted by $Diag_f$. Given a f^{G_S}-diagnosable system, the diagnosis decision can be defined as follows:

- $Diag_f(obs_s) = 1$, when $\exists p \in L(G)$, $P_S(p) = obs_s$ and f occurs in p (as G is a f^{G_S}-diagnosable system, it is the case that $\forall p \in L(G)$ with $P_S(p) = obs_s$, then f occurs in p if obs_s is long enough);
- Otherwise, $Diag_f(obs_s) = 0$.

Here $Diag_f(obs_s)$ being 1 means the effective occurrence of f on the trajectory with obs_s as its observations, otherwise $Diag_f(obs_s)$ is 0. Note that with a diagnosable subsystem G_S, only observations from G_S are involved in diagnosis decision. Otherwise, we need all observations in the whole system to decide diagnosis.

Lemma 2 *If a system G is f^{G_S}-diagnosable, then it is $f^{G_{S\prime}}$-diagnosable, where $G_S \subseteq G_{S\prime}$.*

Proof :
Suppose that G is f^{G_S}-diagnosable and $G_S \subseteq G_{S\prime}$. Since G is f^{G_S}-diagnosable, from definition 19, with a finite and enough number of observations from G_S, we are sure that f has effectively occurred after the occurrence of f. Then from the fact that the observable events between components are disjoint and from assumption 3 that implies no loop of unobservable events in any component, the occurrence of f is also determinable after a finite and enough number of observations from the subsystem $G_{S\prime}$. It follows that G is $f^{G_{S\prime}}$-diagnosable. Actually without assumption 3, there may exist the case that the system is f^{G_S}-diagnosable but not $f^{G_{S\prime}}$-diagnosable. For example, if in the subsystem G_S, there exists a cycle with only unobservable events, it may still be a diagnosable subsystem, i.e., no existence of critical path. It is possible that $G_{S\prime}$ is not diagnosable subsystem if there exists a critical pairs in $G_{S\prime}$.

Lemma 2 means that the existence of a diagnosable subsystem verifies the diagnosability property of the whole system. Let G be a f^{G_S}-diagnosable system. If $\forall G'_S, G'_S \subset G_S$, G is not $f^{G'_S}$-diagnosable, then G_S is called a **minimal diagnosable subsystem** with respect to f, which is not necessarily unique.

3.3.2 Diagnosability Information Abstraction

We now present how to abstract diagnosability information from local twin plant and local twin checkers. As said before, G_f denotes the component where the fault f may occur and T_f denotes

the local twin plant of the component G_f. In a local twin plant, LPCP is a local path containing an ambiguous state cycle with at least one observable event of the component. For example, figure 3.7 contains two LPCPs of the local twin plant T_1(see figure 3.3) since both of them have ambiguous state cycles with at least one observable event of G_1. The relations between global critical paths and LPCPs can be concluded as follows.

1. For any global critical path in the global twin plant, its projection on the local twin plant T_f must be a LPCP.

2. If there is no LPCP in T_f, then there is no global critical path in the global twin plant.

3. If there is a LPCP in T_f, then it may or may not be extended as a global critical path in the global twin plant when synchronizing with other local twin checkers.

What we are interested in is the case 3, where the diagnosability information, i.e., LPCPs, originates only in the local twin plant T_f. So our goal is to determine if the LPCPs are going to develop as a global critical path with as small space as possible instead of computing the global twin plant. Since all local paths of local twin plant are synchronized via communication events with local twin checkers, they can only be blocked by communication events. With assumption 3, we can see that one way to check whether a local path in the local twin plant can survive after synchronizing with local twin checkers consists in checking the existence of observable events of all involved components in the corresponding cycles. So it suffices to consider all communication events and only those observable events in cycles.

In the local twin plant T_f, considering that the events set of a local twin plant is the set of communication events and observable events and in any component, each cycle contains at least one observable event of this component, an ambiguous state cycle could only be two types:

1. with both communication events and observable events;

2. with only observable events.

Then to keep all communication events and only observable events in ambiguous state cycles of all LPCPs, we first operate delay closure with respect to the set of communication events, which keeps all ambiguous state cycles in T_f of the first type. But this delay closure loses the observable events in this kind of cycles. For this, it is sufficient to add the existence information of observations in the component G_f. For the second type, operating delay closure with respect to the set of communication events on T_f loses those ambiguous state cycles with only observable events.

So we recuperate them by adding their corresponding ambiguous state cycle with observation information. We define abstracted local twin plant as follows, where we use obs_i to represent the existence of observable events of component G_i. The idea is to recover the lost ambiguous state cycles and the lost observable information in a qualitative way. While the question about how many observable events and which ones in a cycle does not affect diagnosability verification in the following steps.

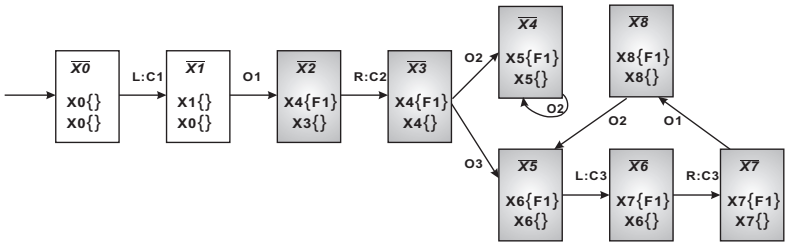

Figure 3.7: Two local possible critical paths (LPCPs) in the local twin plant T_1 (see figure 3.3).

Definition 20 ***(Abstracted Local Twin Plant-ALTP)*** *The abstracted local twin plant (ALTP) from a local twin plant T_i, denoted by T_i^a, is obtained by the following steps:*

1. *Delay Closure with respect to the set of communication events is operated on the local twin plant T_i, $T_i^a = \mathsf{C}_{\Sigma_{i_c}}(T_i)$.*

2. *For any cycle ϕ in T_i^a, $\phi = q_0' \xrightarrow{c_1} q_1' ... \xrightarrow{c_m} q_m'$ with $q_0' = q_m'$ and all states in this cycle are ambiguous states, suppose its corresponding cycle in T_i is: $q_0 \xrightarrow{e_1} q_1 ... \xrightarrow{e_n} q_n$ with $q_0 = q_n$, from assumption 3, we know that there exists at least one observable event in this cycle, then ϕ is modified as $\phi\prime = q_0' \xrightarrow{c_1} q_1' ... \xrightarrow{c_{m-1}} q_{m-1}' \xrightarrow{obs_i} q^F \xrightarrow{c_m} q_m'$ in T_i^a, where obs_i represents at least one observable event of component G_i, i.e., the existence of observable events of G_i, and q^F represents a local twin plant state that is ambiguous with respect to any fault in F, whose ambiguity is the same as any other local twin plant state in this cycle.*

3. *If there exists a local path in T_i: $q_0 \xrightarrow{e_1} q_1 ... \xrightarrow{e_n} q_n$, where q_0 is the initial state of T_i, $\exists j \in \{0, ..., n-1\}$, $q_j = q_n$, $\forall k \in \{j+1, ..., n\}, e_k \in \Sigma_{i_o}$ and $\forall q_p, p \in \{j, ..., n\}$, q_p is an ambiguous state, then suppose that the corresponding local path in T_i^a is $p = q_0' \xrightarrow{c_1} q_1' ... \xrightarrow{c_m} q_m'$, then it is modified as $p\prime = q_0' \xrightarrow{c_1} q_1' ... \xrightarrow{c_m} q_m' \xrightarrow{obs_i} q^F \xrightarrow{obs_i} q^F$, where obs_i represents at least one observable event of component G_i, and q^F represents an ambiguous*

local twin plant state with respect to the set of faults F, whose ambiguity is the same as q_k, $\forall k \in \{j, ..., n\}$.

In the above definition, clause(1) keeps all ambiguous state cycles in the local twin plant that with at least one communication event. Clause(2) gets back the information about the existence of observable events in the ambiguous state cycles with communication events. And clause(3) recuperates the lost ambiguous state cycles with only observable events by adding their corresponding cycle with observation information. So we can see that since ambiguous state cycles only originate in T_f, the ALTP of T_f is to retain the corresponding part of all ambiguous state cycles with the information about the existence of observable events as well as communication events, i.e., all original diagnosability information. Then we have the following lemma.

Lemma 3 *The ALTP T_f^a for the local twin plant T_f retains the corresponding part of all ambiguous state cycles with observation information of all LPCPs in T_f.*

In the similar way to ALTP, we define the abstracted local twin checker as follows.

Definition 21 ***(Abstracted Local Twin Checker-ALTC)*** *The abstracted local twin checker (ALTC) from a local twin checker C_i, denoted by C_i^a, is obtained by the following steps:*

1. *Delay Closure with respect to the set of communication events is operated on the local twin checker C_i, $C_i^a = \mathsf{C}_{\Sigma_{i_c}}(C_i)$.*

2. *For any cycle ϕ in C_i^a, $\phi = q_0' \xrightarrow{c_1} q_1' ... \xrightarrow{c_m} q_m'$ with $q_0' = q_m'$, suppose its corresponding cycle in C_i is: $q_0 \xrightarrow{e_1} q_1 ... \xrightarrow{e_n} q_n$ with $q_0 = q_n$, then ϕ is modified as $\phi\prime = q_0' \xrightarrow{c_1} q_1' ... \xrightarrow{c_{m-1}} q_{m-1}' \xrightarrow{obs_i} q^i \xrightarrow{c_m} q_m'$ in C_i^a, where obs_i represents at least one observable event of component G_i and q^i represents a local twin checker state for the component G_i.*

3. *If there exists a local path in C_i: $q_0 \xrightarrow{e_1} q_1 ... \xrightarrow{e_n} q_n$, where q_0 is the initial state of C_i, $\exists j \in \{0, ..., n-1\}$, $q_j = q_n$, $\forall k \in \{j+1, ..., n\}, e_k \in \Sigma_{i_o}$, suppose that the corresponding local path in C_i^a is $p = q_0' \xrightarrow{c_1} q_1' ... \xrightarrow{c_m} q_m'$, then it is modified as $p\prime = q_0' \xrightarrow{c_1} q_1' ... \xrightarrow{c_m} q_m' \xrightarrow{obs_i} q_m'$, where obs_i represents at least one observable event of component G_i.*

For all components except G_f, their ALTC preserves all communication events as well as all cycles with the existence of observable events. The idea is to check whether the ambiguous state cycles in T_f^a can survive after synchronizing with all other ALTCs. After synchronization, if these ambiguous state cycles do not disappear and contain at least one observable event for all involved

components, then they are considered to survive in the global twin plant. In T_f^a, the corresponding local path of a LPCP from T_f is still called a LPCP, which preserves all ambiguous state cycles. The ALTP thus keeps all necessary and sufficient diagnosability information but is practically much smaller than its corresponding local twin plant. Figure 3.8 illustrates part of ALTP T_1^a (top), part of ALTC C_2^a (middle) and part of ALTC C_3^a (bottom) of components G_1, G_2 and G_3, respectively. T_1^a is abstracted from figure 3.7 and C_2^a (resp. C_3^a) is abstracted from only one path in figure 3.6. Here Obs_1, Obs_2 and Obs_3 represent at least one observable event of component G_1, G_2 and G_3. And $q^{\{F1\}}$ represents an ambiguous local twin plant state with respect to $F1$. Only T_1^a contains diagnosability information, i.e., ambiguous state cycles formed by gray nodes with at least one observable event of G_1.

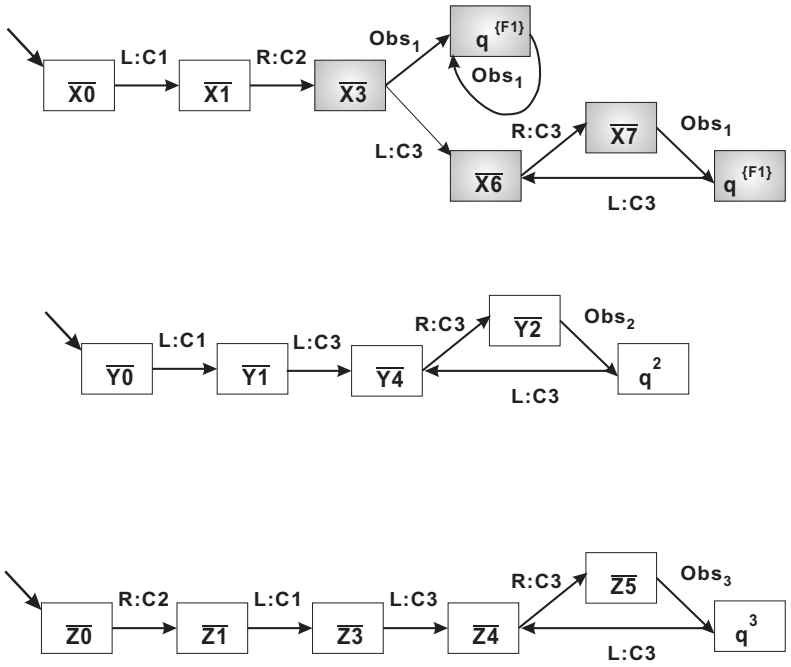

Figure 3.8: Part of ALTP T_1^a(top), part of ALTC C_2^a (middle) and part of ALTC C_3^a (bottom) of components G_1, G_2, G_3, respectively.

3.3.3 Distributed Verification

The reachability of LPCPs of T_f^a in the global twin plant can be determined by synchronizing T_f^a with other ALTCs. Now we first define a globally consistent LPCP.

Definition 22 ***(Globally Consistent LPCP)***

- *Given a LPCP ρ in T_f^a and a subsystem G_S containing the component G_f, if ρ does not disappear and contains an ambiguous state cycle with at least one observable event for all involved components of G_S in the synchronized FSM of T_f^a and the set of ALTCs corresponding to the set of components of G_S except G_f, then ρ is called a consistent LPCP of the subsystem G_S.*

- *If a local path ρ in T_f^a is a consistent LPCP of the system G, i.e., $G_S = G$, then ρ is called a* globally consistent LPCP.

Remark: if ρ is a consistent LPCP of the subsystem G_S and there is no communication event in G_S that is also contained in $G \backslash G_S$, then ρ is a globally consistent LPCP. The reason is that in this case all communication events in ρ are validated in terms of the interactions with its neighborhood. Thus it will still be a consistent LPCP when current subsystem is extended to the whole system. So ρ is globally consistent.

Lemma 4 *A LPCP in T_f^a is a globally consistent LPCP iff it corresponds to a global critical path.*

Proof :
($\Rightarrow$) Suppose a LPCP ρ in T_f^a is a globally consistent LPCP and that it does not correspond to a global critical path. Recall that a global critical path is a path in the global twin plant containing an ambiguous state cycle with at least one observable event for all components. It follows that there are two causes leading to non correspondence of ρ to a global critical path:

- all ambiguous state cycles in ρ disappear when synchronizing with other ALTCs, which means that it has no corresponding path with ambiguous state cycles in the global twin plant;

- ρ has a corresponding path in the global twin plant with ambiguous state cycles but none of them contains at least one observable event of all components, which means that these preserved ambiguous state cycles have observable events of only some components but not all.

From the definition of a globally consistent LPCP mentioned as above with assumption 3 implying that each cycle in any component of the system has at least one observable event of its component, both cases indicate that ρ is not a globally consistent LPCP, which contradicts the assumption that actually it is.

($\Leftarrow$) Now suppose that a LPCP ρ in T_f^a corresponds to a global critical path but it is not a globally consistent LPCP. Its non global consistency means that either all its ambiguous state cycles disappear when synchronizing with other ALTCs or its corresponding path in the global twin plant contains ambiguous state cycles but none of them contains observable events of all components. From the definition of a global critical path (see section 3.2), both cases imply that ρ has no corresponding global critical path. Thus the assumption that ρ corresponds to a global critical path is contradicted.

Lemma 4 implies the equality between globally consistent LPCPs and global critical paths, i.e., there is no globally consistent LPCP iff there is no global critical path. Then from theorem 2, we can obtain the following important result.

Theorem 4 *The fault f is diagnosable in a system G iff there is no globally consistent LPCP.*

3.3.4 Algorithm

Algorithm 1 Optimized Diagnosability Verification Algorithm for Distributed DES

```
1: INPUT:
   component models G_1, ..., G_n of the system G;
   the considered fault f that may occur in G_f
2: T ← ConstructALTP(G_f, f)
3: G_S ← G_f
4: while T ≠ ∅ and ConnectComp(G_S) ≠ ∅ do
5:    G ← Select(ConnectComp(G_S))
6:    T' ← ConstructALTC(G)
7:    T ← Sync(T, T')
8:    T ← Reduce(T)
9:    G_S ← Add(G_S, G)
10: end while
11: if T = ∅ then
12:    return G_S
13: else
14:    return T
15: end if
```

Now we describe our distributed algorithm to check diagnosability based on theorem 4, which is optimized in the sense that we reduce the search space as small as possible by distributing the analysis on the ALTP and relative ALTCs. The starting point is the construction of the ALTP of the component G_f. As said before, since f occurs only in the component G_f, we only need

to construct the ALTP of the component G_f. Then with this original ALTP, the core part is its incremental synchronization with the ALTCs of components communicating with current subsystem. Here the synchronized FSM of the ALTP T_i^a and the ALTC C_j^a is called the ALTP of the subsystem composed of G_i and G_j. As shown in the pseudo-code for this verification procedure, algorithm 1 performs as follows. Given the input as the set of component models, the fault f that may occur in the component G_f, we first construct the ALTP of G_f that contains all LPCPs (line 2). Current subsystem, denoted by G_S, is then assigned by G_f. When the reduced ALTP of G_S is not empty and there exists at least one component neighboring to G_S, i.e., a component has at least one communication event in common with the ones of G_S, which means that there exists at least one consistent LPCP of current subsystem whose global consistency should be further checked, then the algorithm repeatedly performs as follows:

1. Select one component neighboring to G_S and construct the ALTC of this selected component. (line 5-6)

2. The obtained ALTC is synchronized with the previous (reduced) ALTP based on their common communication events (left communication events and right communication events). Then the newly obtained ALTP is reduced by retaining only consistent LPCPs of the newly extended subsystem considering that the synchronization may possibly produce some paths that are not consistent LPCPs. This extended subsystem is obtained by adding this selected component. (line 7-9)

Note that each time when we reduce the ALTP to retain only LPCPs, we keep the same events set. In other words, a reduced ALTP has the same events set as its corresponding non-reduced ALTP. Only in this way, we can guarantee that the synchronized FSM of the reduced ALTP with another ALTC based on their shared communication events has the same result as the synchronized FSM of their corresponding local twin plant and local twin checker. If the reduced ALTP of current subsystem G_S is empty, which means that there is no consistent LPCP of G_S, thus we verify the non existence of global critical paths. Then the algorithm returns current subsystem as a diagnosable subsystem (line 11-12). Otherwise, if there is no component connected with G_S and the reduced ALTP of G_S is not empty, then there exists at least one globally consistent LPCP. From theorem 4, the system is not diagnosable. Thus the algorithm returns the final reduced ALTP that provides some useful information about global critical paths (line 13-14).

3.4 Results Discussion

When the algorithm returns a diagnosable subsystem G_S, if the number of involved components $|G_S| \leq 2$, we can directly prove that it is a minimal diagnosable subsystem. Otherwise, if $|G_S| > 2$, it is not necessarily a minimal diagnosable subsystem. When the system is diagnosable, we can enhance the possibility of the returned subsystem being a minimal diagnosable subsystem by adopting an appropriate component selection strategy. Let Σ_{S_c} be the set of communication events in the current subsystem. To choose next component for further exploitation, we prefer to select the one, suppose G_i, such that $|\Sigma_{S_c} \cap \Sigma_{i_c}|$, the number of communication events in G_i contained also in the current subsystem, is maximum compared to any other component to be selected. The idea here is to block LPCPs by the concerned communication events with as few components as possible if the system is diagnosable. In this way, more communication events of the selected component are involved in the current subsystem, i.e., more communication constraints imposed on LPCPs, more likely the LPCPs disappear after the synchronization.

Consider our example (see figure 3.8). Suppose that after T_1^a being built, we choose G_2 as the next component to decide diagnosability, the LPCPs in T_1^a are still consistent after synchronizing with the abstracted local twin checker C_2^a. Thus we select G_3 for next checking and all LPCPs disappear after synchronizing with C_3^a. Then our algorithm returns a diagnosable subsystem involving all three components. However, this is not a minimal diagnosable subsystem. If we adopt component selection strategy mentioned as above, after obtaining T_1^a, we select G_3 as the next component because it contains more communication events in common with the ones of G_1 (c1, c2, c3) and thus has more constraints compared to G_2, whose common communication events with G_1 is (c1, c3). When the LPCPs are synchronized with C_3^a, all of them disappear. Our algorithm thus returns a diagnosable subsystem composed of G_1, G_3, which is actually a minimal diagnosable subsystem.

We have described how a diagnosable subsystem can improve diagnosis algorithm in terms of observation reduction in section 3.3.1. Now we illustrate this with our example. Consider the diagnosable subsystem composed of G_1 and G_3. Since it is a diagnosable subsystem, then only the observations from G_1 and G_3 are sufficient for diagnosis decision, which means that we do not need observations from the component G_2. While without a diagnosable subsystem, to decide diagnosis, the observations from all three components are required. Table 3.1 shows the diagnosis decision with the observations required with the diagnosable subsystem composed of G_1 and G_3 and without it. In the table, $s1 \| s2$ denotes the synchronization of events sequence $s1$ and

Table 3.1: Diagnosis Decision with and without a Diagnosable Subsystem

Diagnosis Table	Observations	Diagnosis decision
With diagnosable subsystem	$O6.O1.O3.(O1.O2 \| O7)^*$	1
	$O1.(O3 \| O6)(O1.O2 \| O7)^*$	0
	...	...
Without diagnosable subsystem	$O6.(O1.O3 \| O4.O5).(O1.O2 \| O7 \| O5)^*$	1
	$((O1.(O3 \| O6)) \| O4.O5).(O1.O2 \| O7 \| O5)^*$	0
	...	...

sequence $s2$, where there is no synchronized event between these two sequences. For example, the result of $O1.O2 \| O7$ is the sequences set $\{\{O7.O1.O2\}, \{O1.O7.O2\}, \{O1.O2.O7\}\}$. With the diagnosable subsystem, with only observations from this subsystem, we can decide diagnosis. Otherwise, we need observations from all components.

For the sake of generalization, we give some definitions, such as local pre-diagnoser and local twin plant, to deal with set of faults. However, considering that the search space is exponential in the number of faults, it is better to check diagnosability by running our algorithm as many times as the number of faults, each time for one fault, which greatly reduces complexity. Obviously, our algorithm practically improves the efficiency of diagnosability problem solving. The twin plant method has polynomial space complexity in the number of system states, which however has exponential complexity in the number of components. In our approach, from the way to construct ALTP and ALTCs and to distribute diagnosability checking on them, in the worst case, the space complexity is polynomial in the number of a subset of system states, i.e., the states of communication transitions and observable transition in cycles. Even though we still have exponential complexity in the number of components, but normally the growth factor is greatly reduced, i.e., the state number of ALTP being much smaller than that of local twin plant. Furthermore, in practice, our algorithm often involves only a subset of components, both for the diagnosable cases and non-diagnosable cases.

3.5 Relaxation of assumptions

In our approach, we have the assumption 1 that any communication event is not observable, which is actually the more difficult case than that where the communication event is observable. Now we relax this assumption by dividing the set of communication events into two disjoint parts: $\Sigma_{i_c^o}$, observable communication events set and $\Sigma_{i_c^u}$, unobservable communication events set. In other

words, communication events could be observable or unobservable. Then our algorithm can be adapted as following.

- From the local component G_f, we construct its pre-local diagnoser by preserving the information about all observable events as well as all communication events, including observable ones and unobservable ones, and then append to each retained state fault information, which is the same as that described above.

- In this pre-local diagnoser, for each event $\sigma \in \Sigma_{i_c^u}$, we distinguish it between two instances by adding the prefix of L (for left instance, σ being $L : \sigma$) and R (for right instance, σ being $R : \sigma$). Then we obtain local twin plant by synchronizing these two instances based on the set of observable events and the set of observable communication events, i.e., Σ_{i_o} and $\Sigma_{i_c^o}$.

- We construct ALTP by keeping all communication events, including observable communication events, left unobservable communication events and right unobservable communication events, as well as the observable information in ambiguous cycles.

- For other connected components, we construct their local twin checker in the same way as local twin plant except without diagnosis information. Afterwards the corresponding ALTC is obtained by operating delay closure to keep all communication event and then by recuperating observable information for each cycle.

- Global consistency checking of LPCPs consists in synchronizing ALTP and connected ALTCs based on communication events (observable communication event with observable one, left unobservable communication event with left unobservable one and right unobservable communication with right unobservable one). Before each synchronization, we only keep those paths with ambiguous state cycles containing observable events for all involved components. If in the end, the final obtained FSM is not empty, which means that there does exist at least one globally consistent LPCP and thus non diagnosability is verified. Otherwise, if final FSM is empty, then non existence of globally consistent LPCP and thus diagnosability is proved.

For the sake of simplicity, we also have assumption 2 that a communication event is not a fault event, i.e. $\Sigma_{i_f} \cap \Sigma_{i_c} = \emptyset$. If we relax this assumption, which means that a fault event can also be an unobservable communication event. Actually our approach can also deal with the relaxed case. For example, without loss of generality, suppose that the considered fault event f is

a communication event contained in two components G_i and G_j. We check diagnosability in the following three ways.

- Construct the ALTP of G_i to obtain LPCPs and then check their global consistency with connected ALTCs, including that of G_j
- Construct the ALTP of G_j to obtain LPCPs and then check their global consistency with connected ALTCs, including that of G_i
- Construct the ALTPs of both G_i and G_j and then synchronize them to get LPCPs, and then check global consistency with connected ALTCs

Since the fault event is a shared event of G_i and G_j, from the global consistency checking procedure, it is easy to prove that the results of the above three methods are the same. So we can just choose only one component that contains the fault communication event and then perform the same algorithm as the case where the fault is not a communication event, i.e., the first or the second procedure described above.

As for assumption 3 that each component projection of the global language is observation live, which is quite constrained but can be easily relaxed to the liveliness of local observation language. In the relaxed case, we assume that both the language and the observable language of any component are live without considering their correspondence in global language. In other words, their observable liveness is not necessarily true in global language. In this case, we use the same algorithm with the only difference that during searching for LPCP and global consistency, we only care about those paths with ambiguous state cycles containing observable event only for the component G_f. So diagnosability verification procedure can be described as follows.

- From the local component G_f, we construct its pre-local diagnoser and then its local twin plant as well as ALTP.
- For other connected components, we construct their local twin checker in the same way as local twin plant except without diagnosis information. Then the corresponding ALTC is obtained by only operating delay closure to keep all communication event. Here we do not need to recuperate observable information in cycles because with the relaxed assumption, what we are interested in is only the observable events of G_f in ambiguous cycles, i.e., whether ambiguous cycles in local twin plant of G_f will be blocked during global consistency checking.

- During global consistency checking by synchronizing ALTP and connected ALTCs, before each synchronization, we only keep those paths with ambiguous state cycles containing observable events for G_f. If in the end, the final obtained FSM is not empty, non diagnosability is verified since there is no globally consistent LPCP. Otherwise, if final FSM is empty, we verify system diagnosability.

3.6 Related work

As said in chapter 2, the first definition of diagnosability for DES is introduced in [61]. The authors proposed a necessary and sufficient condition for testing diagnosability by constructing a deterministic diagnoser for the entire system. The main drawback is its exponential space complexity in the number of system states and as a consequence doubly exponential in the number of components in the system. Then the authors of [44] and [80] proposed new algorithms with polynomial complexity in the number of system states, which introduced the classical twin plant method. These approaches assume that the knowledge about the system is the monolithic model, which is not realistic for real complex systems. It is why very recently distributed approaches for diagnosability began to be investigated, relying on local objects.

In [53], the author introduces the diagnosability problem of a system in a distributed way. In this approach, the local twin plant is constructed for each component based on its local model and then the local twin plant for the component G_f, the component where the fault may occur, is incrementally synchronized with all connected local twin plants. The local twin plant of G_f is constructed by synchronizing the non-reduced left instance of pre-diagnoser and the non-reduced right instance. Furthermore, during the diagnosability verification, the set of local twin plants are neither abstracted nor reduced. So in the worst case, i.e., all components are connected directly or indirectly, the finally obtained FSM to search for global critical paths is actually the global twin plant. The search space can be reduced compared to centralized one only when there does exist at least one component that is not connected to the component G_f, neither directly nor indirectly.

In [64], to search for global critical paths in a distributed way, non-diagnosable states in each local twin plant are first decided by propagating diagnosability information. This is done by synchronizing relative local twin plants based on their connectivity with the local twin plant of the component G_f. And then reduced local twin plants are computed that only contain the parts relevant to solve the diagnosability problem. So diagnosability verification is performed by synchronizing the reduced but non-abstracted local twin plants.

Different from the existing distributed approaches mentioned above, we first optimize the local twin plant construction by reducing the left and right instances of the pre-diagnoser for the component G_f. And then we show how to construct ALTP and ALTC by abstracting necessary and sufficient diagnosability information from local twin plant and local twin checker. Then the search for global critical paths is performed by synchronizing the reduced ALTP with the connected ALTCs. In a system where there are a large number of observable events, our approach greatly reduces the search space since in our abstracted version, we only retain one observable event for each component in each cycle.

Another similar approach is described in [20], where modular diagnosability is defined and its verification is performed based on checking the global consistency of indeterminate cycles through synchronizing with only the communication events in other connected components. The major differences between our approach described in this chapter and theirs are as follows.

- The original diagnosability information is obtained by constructing deterministic local diagnoser in [20], which is modified from that described in [61], while we choose to get this by building local twin plant that is an optimized version of that presented in [53].

- Their modular diagnosability focuses only on traces where events from module or called component G_f, which is the component where the fault f originates. More precisely, modular diagnosability of f in the system implies that after f occurs in the component G_f, detection and isolation of that fault is only required along continuations that involve events from G_f. In other words, modular diagnosability requires that only observations from G_f is sufficient to determine whether f has occurred or not.While our proposed regional diagnosability is to search for a diagnosable subsystem, which exists if the fault is diagnosable in the entire system.

- We assume that observable behavior of each component is live. The idea is to guarantee that regional diagnosability of the fault implies its diagnosability in the whole system, which is not the case in [20]. Furthermore, we assume the common events between components,i.e., communication events, to be unobservable while they assume the common events to be observable, which leads to great difference during global consistency checking of original diagnosability. Obviously non observability of common events makes the verification more complex but more general when dealing with real systems.

Chapter 4

Distributed pattern diagnosability

The existing work for pattern diagnosability adopts centralized framework ([34], [43]), which means that the monolithic model for a distributed system is hypothesized beforehand. As said in the fault event case, this hypothesis is normally unrealistic when dealing with real complex systems. So we propose a distributed framework for pattern diagnosability. For the sake of simplicity, we first consider a pattern as one sequence of events. Then we will extend our approach for general patterns that may have multiple even infinite sequences of events.

In this section, we show how to generalize distributed diagnosability analysis from fault events to predefined patterns. The idea is to find an equivalent alternative to the centralized pattern diagnosability checking but being more efficient in order to improve the scalability of the problem. Our approach contributes to the pattern diagnosability problem in several aspects. First, we extend the pattern diagnosability problem from the centralized framework to the distributed one. Second, pattern recognition can be checked by incrementally constructing pattern recognizers, which may concern several components. Third, we construct pattern verifier and then abstract necessary and sufficient diagnosability information from it to search for partial critical paths, which are similar to LPCPs described in section 3.3.2, whose global consistency is then checked. Finally some key information about the reasons why the system is not diagnosable is provided by our algorithm when the system is not diagnosable, which can help the designer to improve the diagnosability level of the system by rearranging sensor placement, system reconfiguration, etc.

Actually there is another possible way to deal with pattern diagnosability for a distributed system, where a fault event is added to the end of pattern and the pattern is considered as one additional component in this system, i.e., the system is composed of its own components plus the pattern component. And then we can possibly apply the method in chapter 3 to check the

diagnosability of this added fault event. However, to fulfill all assumptions of chapter 3 and the assumptions about what is a pattern, it would imply that all events in the system should be considered as observable communicating events, which is too strict to be realistic. This is why we propose here a more efficient distributed method in this chapter.

4.1 Models and notations

As in the fault event case, a distributed system under our consideration is composed of a set of components $G_1, ...G_n$ that communicate with each other by their shared communication events. Each component is modeled by a FSM defined in the definition 13. Different from the fault event case (see section 3.1), the events set Σ_i of the component G_i is divided into three disjoint parts: Σ_{i_o}, the set of observable events in G_i; Σ_{i_u}, the set of unobservable events in G_i; and Σ_{i_c}, the set of unobservable communication events in G_i that are shared by at least one other component. The only shared events between different components are unobservable communication events. Figure 4.1 depicts a distributed system composed of three components G_1(top), G_2(bottom left) and G_3(bottom right), where the events Oi denote observable events, the events Ci denote unobservable communication events, the events Ui denote unobservable events.

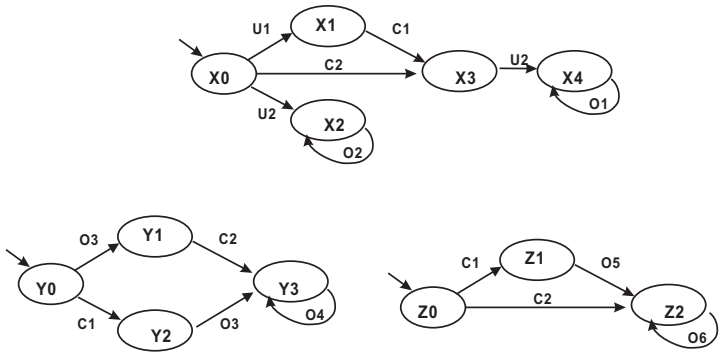

Figure 4.1: Distributed system with three components G_1(top), G_2(bottom left) and G_3(bottom right).

In a pattern $\Omega = (Q_\Omega, \Sigma_\Omega, \delta_\Omega, q^0_\Omega, F_\Omega)$, we call an event σ a significant event of Ω if $\exists(q, \sigma, q\prime) \in \delta_\Omega$ with $q \neq q\prime$. In other words, a significant event of a pattern is an event that changes at least one pattern state. We use Θ_Ω to denote the set of significant events of Ω and $\widehat{\varpi_q}$ to denote the set of events in Θ_Ω such that $\forall\sigma \in \widehat{\varpi_q}, \exists(q, \sigma, q\prime) \in \delta_\Omega$, with $q \neq q\prime$. Thus $\widehat{\varpi_q}$ is actually the set of significant events of Ω that change the state q.

Definition 23 *(Simple Pattern). A pattern Ω is called a simple pattern if* $\forall q_\Omega, q_\Omega \in Q_\Omega \wedge q_\Omega \notin F_\Omega$, *we have* $|\widehat{\varpi_{q_\Omega}}| = 1$.

In a simple pattern, for any state that is not a final state, there is one and only one event that changes it to another state. With simple patterns, the diagnosis problem can be generalized from detecting fault events to recognizing events sequence that can describe more general objectives, such as ordered occurrence of several important events, multiple occurrences of the same fault, etc [43]. Actually the fault event case is a special case of the pattern one.

Due to F_Ω being stable, we can merge all final states in one beforehand, which has no impact on our diagnosability analysis. In a simple pattern, the significant event that changes state q is denoted by ϖ^s_q. If q_f is the final state, $\varpi^s_{q_f} = \epsilon$. Figure 4.2 depicts a pattern that describes the ordered occurrence of two significant events $\Theta_\Omega = \{u1, o3\}$. For this pattern, we have $Q_\Omega = \{p0, p1, p2\}$, $\Sigma_\Omega = \{c1, c2, u1, u2, o1, o2, o3, o4, o5, o6\}$, $\Sigma_{\Omega_o} = \{o1, o2, o3, o4, o5, o6\}$, $\Sigma_{\Omega_u} = \{u1, u2\}$. We also have $\Theta_\Omega = \{u1, o3\}$, $\varpi^s_{p0} = u1$, $\varpi^s_{p1} = o3$, $q^0_\Omega = p0, F_\Omega = \{p2\}$. We can see that it has the same events set as the system depicted in figure 4.1.

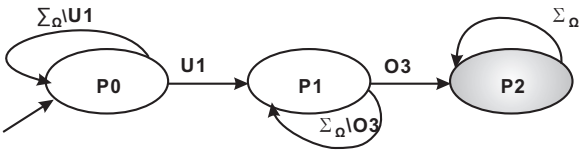

Figure 4.2: The predefined pattern Ω for the system depicted in 4.1.

Given a system $G = (Q, \Sigma, \delta, q^0)$ and a pattern $\Omega = (Q_\Omega, \Sigma_\Omega, \delta_\Omega, q^0_\Omega, F_\Omega)$, we assume $\Sigma = \Sigma_\Omega$, $\Sigma_o = \Sigma_{\Omega_o}$, $\Sigma_u = \Sigma_{\Omega_u}$ and $\forall \sigma \in \Theta_\Omega, \forall \sigma\prime \in \Sigma_c, \sigma\prime \neq \sigma$, where Σ_c is the set of communication events in G. In other words, any significant event of the pattern is not a communication event, which has meaning in the sense that communication events function only for exchanging messages between components of the system. And we assume that the pattern is not concerned with the distributed structure and communication architecture of the system.

4.2 Theoretical distributed framework for simple pattern diagnosability

Now we show how to distribute the pattern recognition and its diagnosability verification on subsystems without computing the global model and the global pattern verifier. We begin the pattern

recognition from the component containing the event $\omega^s_{q^0_\Omega}$, where q^0_Ω is the initial state of the pattern. In other words, this component, called the initial subsystem in the following, contains the significant event of Ω that changes the initial state of Ω. If the pattern cannot be completely recognized in this component, the subsystem will be extended by propagating only diagnosability relative part to the next selected component for the next recognition. If the next recognition is still not completed, then in the same way we continue to extend the subsystem until the recognition is completed. In this incremental way what we obtain is often a small subpart of the global model because the propagated subsystem part is often much smaller than the whole subsystem, which will be described in Section 4.2.2, and also because components concerned by the pattern are normally a subset of system components.

The idea of centralized pattern diagnosability approach is to check if there exists a critical path in the global pattern verifier. In the pattern case, we still adopt assumption 3, which means that in every component, each cycle has at least one observable event of this component. Then in the global pattern verifier, if a path has an ambiguous state cycle that contains at least one observable event for all components of the system, this path is called a global critical path in the following, whose existence verifies non-diagnosability. Our distributed approach is to avoid the calculation of the global pattern verifier by computing abstracted pattern verifier for the subsystem where the pattern recognition is completed to search for partial critical paths. Then we demonstrate how to decide whether a partial critical path, which is a path in the verifier containing an ambiguous state cycle with at least one observable event for all involved components of this subsystem, corresponds to a global critical path after global consistency checking. In this distributed way, the finally obtained FSM for diagnosability verification is normally a quite small subpart of the global pattern verifier.

As in the fault event case, let $P_S(p)$ denote the projection of the trajectory p to observable events in a subsystem G_S. We define regional pattern diagnosability as follows.

Definition 24 ***(Regional Pattern Diagnosability)*** *A pattern Ω is regionally diagnosable in a system G with respect to a subsystem G_S, iff*

$$\exists k \in N, \forall s \in L(G) \cap L_m(\Omega), \forall t \in L(G) \backslash s, |t| \geq k \Rightarrow$$
$$(\forall p \in L(G),\ P_S(p) = P_S(s.t) \Rightarrow p \in L_m(\Omega)).$$

If a pattern Ω is regionally diagnosable in G with respect to the subsystem G_S, then G_S is called a Ω-diagnosable subsystem. In such a system, we are sure about the occurrence of the pattern when enough events are observed from G_S after its occurrence. So similar to fault event case, the

observations from G_S are sufficient for diagnosis decision with respect to Ω, denoted by $Diag_\Omega$. So given a Ω-diagnosable subsystem G_S, the diagnosis decision can be defined as follows:

- $Diag_\Omega(obs_s) = 1$, when $\exists p \in L(G) \cap L_m(\Omega)$, $P_S(p) = obs_s$ (as G_S is a Ω-diagnosable subsystem, then we have that $\forall p \in L(G)$ with $P_S(p) = obs_s$, then $p \in L_m(\Omega)$ if obs_s is long enough);
- Otherwise, $Diag_\Omega(obs_s) = 0$.

Here $Diag_\Omega(obs_s)$ being 1 means the effective recognition of Ω on the trajectory with obs_s as its observations, otherwise $Diag_\Omega(obs_s)$ is 0. Note that with a Ω-diagnosable subsystem G_S, only observations from G_S are involved to decide diagnosis. Otherwise, we need all observations in the whole system to decide diagnosis. Then we have the following lemma, whose proof is similar to that of lemma 2.

Lemma 5 *In a system G, if the subsystem G_S is a Ω-diagnosable subsystem and if $G_S \subseteq G_{S\prime}$, then the subsystem $G_{S\prime}$ is also a Ω-diagnosable subsystem.*

The existence of a Ω-diagnosable subsystem verifies that the system is Ω-diagnosable. And a Ω-diagnosable subsystem is called a minimal Ω-diagnosable subsystem if it does not contain any other Ω-diagnosable subsystem.

4.2.1 Pattern recognizer

Pattern recognition in a subsystem is performed by constructing the corresponding pattern recognizer, which is defined as below.

Definition 25 *(Pattern Recognizer). Given a subsystem $G_S = (Q_S, \Sigma_S, \delta_S, q_S^0)$ and a pattern $\Omega = (Q_\Omega, \Sigma_\Omega, \delta_\Omega, q_\Omega^0, F_\Omega)$, then the pattern recognizer of G_S is $R_{G_S} = (Q_{R_{G_S}}, \Sigma_{R_{G_S}}, \delta_{R_{G_S}}, q^0_{R_{G_S}}, F_{R_{G_S}}) = G_S \times \Omega$, where the initial state is $q^0_{R_{G_S}} = (q_S^0, q_\Omega^0)$, $F_{R_{G_S}} = (Q_S \times F_\Omega) \cap Q_{R_{G_S}}$ is the set of final states of the pattern recognizer.*

Since Ω is a complete FSM, we have $L(\Omega) = \Sigma^*$ and thus we have $L(R_{G_S}) = L(G_S) \cap L(\Omega) = L(G_S)$. The pattern recognizer can show which part of the pattern can be recognized after any trajectory in the subsystem. Given the initial recognizer state $q^0_{R_{G_S}} = (q_S^0, q_\Omega^0)$, a state (q, q_Ω) is called a suspicious state of R_{G_S} if $q_\Omega^0 \neq q_\Omega$. Given the set of all suspicious states in R_{G_S}, $\omega = \{(q_1, q_\Omega^1)$,...$(q_n, q_\Omega^n)\}$, a state $(q_i, q_\Omega^i) \in \omega$ is called a Target Suspicious State (TSS) of R_{G_S} if

in Ω, an events sequence through which q_f, the final state of Ω, can be reached from q^i_Ω contains the minimal number of significant events of Ω compared to all q^k_Ω, where $k \in \{1, ...n\}$. In other words, for a simple pattern, given a recognizer state (q_i, q^i_Ω), it is TSS if the number of transitions with significant events (not those events with the source state as the same as the destination state) in the path from q^0_Ω to q^i_Ω in the pattern is maximal, where q^0_Ω is the initial state of pattern. TSS is thus defined to show which part of the pattern can be recognized in the current subsystem (from initial state to which state, the latter one is the pattern state contained in TSS). If $F_{R_{G_S}} \neq \emptyset$, then the pattern is completely recognized in G_S and R_{G_S} is called the complete recognizer, denoted by R_c. On the other hand, if $F_{R_{G_S}} = \emptyset$, then the pattern is not completely recognized in G_S and we should choose one next component to extend the current subsystem for building the next recognizer. As said before, the initial subsystem is the component that contains the event $\varpi^s_{q^0_\Omega}$. Suppose that $q^c \neq q_f$ is the pattern state contained in a TSS of the pattern recognizer of the current subsystem, then the next component to be selected for subsystem extension is the one containing the event $\omega^s_{q^c}$, which is the significant event that changes state q^c. Considering that any significant event is not a communication event and that Ω is a simple pattern, this selected component is always unique.

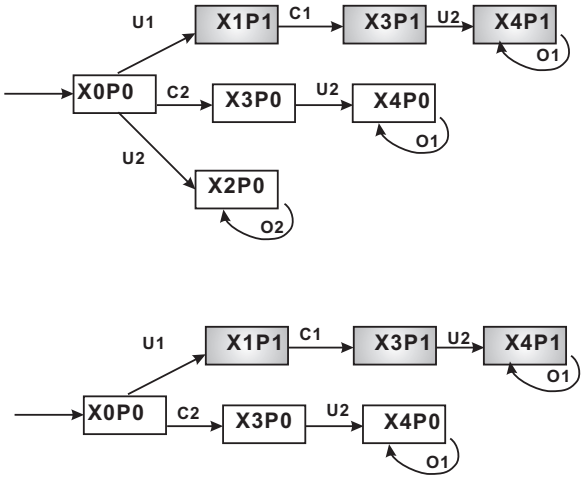

Figure 4.3: The pattern recognizer R_{G_S}(top) and its part $R^\Omega_{G_S}$(bottom)

For our example, the pattern recognizer of the initial subsystem, denoted by R_{G_S}, is depicted by the top part of figure 4.3. Actually the initial subsystem is the component G_1, which contains the event $\varpi^s_{q^0_\Omega} = \varpi^s_{p0} = u1$. In this recognizer, the states whose pattern state part is $p1$ are TSSs, which are represented by the gray nodes. In other words, in the initial subsystem, the recognized

pattern part is the part from the initial state $p0$ to the state $p1$. Then the next selected component should contain the event $\omega^s_{p1} = o3$, i.e., is G_2.

4.2.2 Diagnosability information propagation

A non complete recognizer can only recognize a part of the pattern. The recognition should be completed by incrementally extending the subsystem. Given a non complete recognizer R_{G_S}, we do the following reduction to retain only diagnosability relative part: retain the set of paths where there is at least one TSS, this set of paths denoted by χ_Ω; retain a path p, if $P_S(p) = P_S(p\prime)$ and $p\prime \in \chi_\Omega$, where $P_S(p)$ denotes the observations of p in the subsystem G_S. The reduced recognizer is denoted by $R^\Omega_{G_S}$. As said before, we have $L(R_{G_S}) = L(G_S)$ and we also have $L(R^\Omega_{G_S}) \subseteq L(R_{G_S})$, then we get $L(R^\Omega_{G_S}) \subseteq L(G_S)$. To recognize the next part of the pattern, we extend the current subsystem by synchronizing $R^\Omega_{G_S}$ with G_j, $R^\Omega_{G_S} \| G_j$, where G_j is the next selected component. Note that we keep the events set of $R^\Omega_{G_S}$ as the same as that of R_{G_S}. Only in this way, the result of this synchronization is the same as that of R_{G_S} with G_j. In a real complex system, $R^\Omega_{G_S}$ is normally a small subpart of R_{G_S}, $L(R^\Omega_{G_S}) \subset L(G_S)$, which means that the propagated part is actually quite limited. Thus the synchronized FSM of $R^\Omega_{G_S}$ and G_j contains smaller space compared to that obtained by synchronizing the whole subsystem recognizer R_{G_S} with the next component G_j. If the pattern can be recognized in the system, then the recognizer of the extended subsystem must achieve next recognition of the pattern.

There are two intentions of diagnosability information propagation. One is to continue the recognition process. Another is to keep all information about critical pairs to facilitate the diagnosability analysis because we retain not only the paths containing TSS but also the paths with the same observations as those with TSS.

For our example, the bottom part of figure 4.3 depicts the diagnosability relative part $R^\Omega_{G_S}$. There is one trajectory in R_{G_S} that is not in $R^\Omega_{G_S}$, which can never be recognized by the pattern when it is synchronized with other components and can never be the one with the same observations as those recognized by the pattern. In other words, this trajectory can never be involved in any global critical pair. Thus it is relative to neither the pattern recognition nor the pattern diagnosability verification. The top part of figure 4.4 presents a part of $R^\Omega_{G_S} \| G_2$. Then we construct the pattern recognizer of this extended subsystem, partly depicted by the bottom part of figure 4.4. It is actually the complete recognizer since it contains final states (corresponding to the final state $p2$ of Ω, in gray) and thus the pattern can be completely recognized in the current subsystem made

up of components G_1 and G_2.

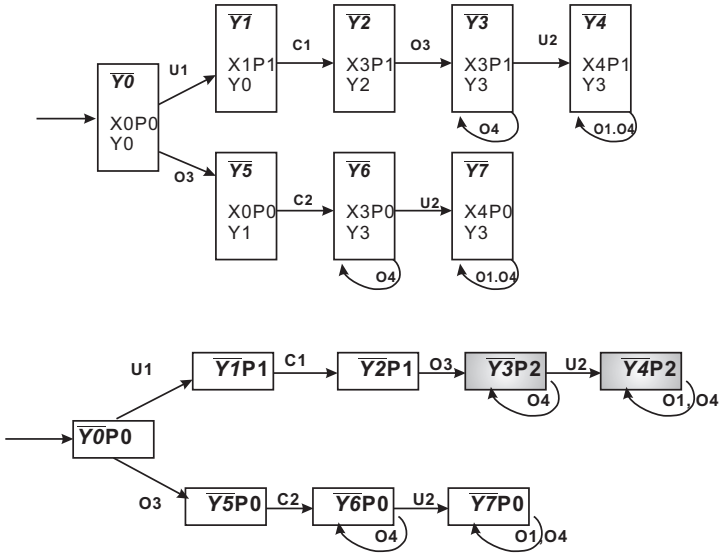

Figure 4.4: Part of the synchronization $R^{\Omega}_{G_S} \| G_2$(top) and part of the complete recognizer R_c (bottom).

4.2.3 Pattern verifier abstraction

Next we show how to construct the abstracted pattern verifier to search for partial critical paths and how to check their global consistency, which normally involves smaller search space than the global pattern verifier. The efficiency will be analyzed in section 5.2.2. Now we provide some lemmas that are the basis of proving the correctness of our distributed approach.

From non intersection of observable events set between components, we can get the following lemma.

Lemma 6 *The two trajectories of any global critical pair have the same local observable projection on each component.*

Proof :

As described before, in the distributed system, for any two different components G_i and G_j, we have $\Sigma_{i_o} \cap \Sigma_{j_o} = \emptyset$. Then from the definition of a global critical pair, i.e., a pair of global trajectories with the same enough observations but only one of them recognizes the pattern, it can be deduced that the observable projections of the two trajectories of any global critical pair on each component are the same, which proves the lemma.

Then from the way to construct the complete recognizer and from lemma 6, we get the next lemma.

Lemma 7 *The complete recognizer R_c contains the subpart in the corresponding subsystem of all global critical pairs.*

Proof :
Due to the fact that the only shared events between components are communication events and that any significant event of the pattern is not a communication event, then any significant event of the pattern is contained in only one component. Let G_i be the initial subsystem. In the pattern recognizer R_{G_i}, the paths not containing a TSS of R_{G_i} are never the subpart of those paths that can be recognized by the pattern because the significant event leading to the pattern state contained in a TSS cannot be in any other component. In other words, the set of paths in R_{G_i} with a TSS must contain the corresponding subpart of all paths that can be recognized by the pattern. From lemma 6, we get that the two trajectories of any global critical pair have the same observable projection on G_i. Thus since in the reduced recognizer $R^{\Omega}_{G_i}$, all the paths having the same observations as those containing a TSS are also retained, we know that $R^{\Omega}_{G_i}$ contains the subpart in the initial subsystem G_i of all global critical pairs. Then after synchronizing $R^{\Omega}_{G_i}$ with the next selected component, suppose G_j, again from lemma 6 and in the same way as above, we can deduce that the recognizer of the extended subsystem contains the subpart in the extended subsystem of all global critical pairs. We repeat the above steps until the complete recognizer is obtained. So now we can deduce that the complete recognizer must contain the subpart in the corresponding subsystem of all global critical pairs.

As in the fault event case, before pattern verifier construction, we refine the complete recognizer by the delay closure with respect to Σ_d, where Σ_d is the set of communication events and observable events, $R_r = \mathsf{C}_{\Sigma_d}(R_c)$. We obtain left instance of R_r, denoted by R^l_r, by prefixing the communication events with L and then by retaining only the paths with at least one cycle that contains both final states of the recognizer and at least one observable event for all involved components. Then we get the right instance of R_r, denoted by R^r_r, by prefixing the communication events with R and then by retaining the paths with at least one cycle that does not contain final states but contains at least one observable event for all involved components. The pattern verifier can be constructed by synchronizing the left instance with the right instance based on all observable events in R_r.

Definition 26 *(Pattern Verifier). Given the refined recognizer R_r, its corresponding pattern verifier is $V = R_r^l \| R_r^r$.*

In the pattern verifier, if a path ϱ has at least one ambiguous state cycle that contains at least one observable event for all involved components, then ϱ is called a partial critical path, which is with the same idea as LPCP in Section 3.3.2. The only difference between a partial critical path and a LPCP is that the ambiguous state cycles in the LPCP only contain observable events for one component and the ambiguous state cycles in the partial critical path should contain observable events for all involved components. The reason is that the LPCP is for the fault event case, which only occurs in one component while the partial critical path is for the pattern case, which normally relates to several components. Next we show how to abstract diagnosability information from the pattern verifier.

Definition 27 ***(Abstracted Pattern Verifier)*** *The abstracted pattern verifier, denoted by V^a, is obtained from its corresponding pattern verifier V by the following steps, where the involved components are $\{G_{s_1}, ...G_{s_m}\}$:*

1. *Delay Closure with respect to the set of communication events is operated on the pattern verifier V, $V^a = \mathsf{C}_{\Sigma_c}(V)$.*

2. *For any cycle ϕ in V^a, $\phi = q'_0 \xrightarrow{c_1} q'_1 ... \xrightarrow{c_m} q'_m$ with $q'_0 = q'_m$ and all states in this cycle are ambiguous states, suppose its corresponding cycle in V is: $q_0 \xrightarrow{e_1} q_1 ... \xrightarrow{e_n} q_n$ with $q_0 = q_n$. If $\forall G_{s_k}, k \in \{1, ...m\}, \Sigma_{s_{k_o}} \cap \{e_1, ..., e_n\} \neq \emptyset$, $\Sigma_{s_{k_o}}$ denoting the observable events set of the component G_{s_k}, which means that this cycle contains at least one observable event for all involved components, then ϕ is modified as $\phi\prime = q'_0 \xrightarrow{c_1} q'_1 ... \xrightarrow{c_{m-1}} q'_{m-1} \xrightarrow{obs_{s_1}} q^{\Omega} ... \xrightarrow{obs_{s_m}} q^{\Omega} \xrightarrow{c_m} q'_m$ in V^a, where obs_{s_i} represents the existence of observable events of component G_{s_i} and q^{Ω} represents a verifier state that is ambiguous with respect to the pattern Ω, whose ambiguity is the same as any other pattern verifier state in this cycle.*

3. *If there exists a local path in V: $q_0 \xrightarrow{e_1} q_1 ... \xrightarrow{e_n} q_n$, where q_0 is the initial state of V, $\exists j \in \{0, ..., n-1\}$, $q_j = q_n$, $e_k \in \Sigma_o, \forall k \in \{j+1, ..., n\}$, where Σ_o denotes the observable events set of the system, this means that all events in this cycle are observable events, if $\forall q_p, p \in \{j, ..., n\}$, q_p is an ambiguous state and if $\forall G_{s_k}, k \in \{1, ...m\}, \Sigma_{s_{k_o}} \cap \{e_{j+1}, ..., e_n\} \neq \emptyset$, which means that this is an ambiguous state cycle with at least one observable event for all involved components, suppose that the corresponding local path in V^a is $p = q'_0 \xrightarrow{c_1}$*

$q'_1 ... \xrightarrow{c_m} q'_m$, then it is modified as $p\prime = q'_0 \xrightarrow{c_1} q'_1 ... \xrightarrow{c_m} q'_m \xrightarrow{obs} q^{\Omega} \xrightarrow{obs_{s_1}} q^{\Omega} ... \xrightarrow{obs_{s_m}} q^{\Omega}$, where obs means that the event that changes the state from q'_m to q^{Ω} is an observable event but which one is not important, obs_{s_i} represents at least one observable event of component G_{s_i}, and q^{Ω} represents an ambiguous verifier state with respect to the pattern Ω, whose ambiguity is the same as q_p, $\forall p \in \{j, ..., n\}$.

From the construction of abstracted pattern verifier, we know that it retains all communication events as well as all ambiguous state cycles that contain at least one observable event for each involved component in the current subsystem. In other words, the abstracted pattern verifier retains the corresponding part of all partial critical paths in the pattern verifier, the proof is similar to that of lemma 3 in section 3.3.2.

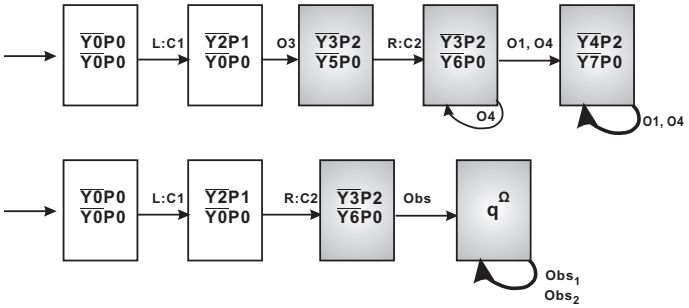

Figure 4.5: Part of the pattern verifier V(top) and part of abstracted pattern verifier V^a(bottom) .

For our example, the top part of figure 4.5 partly depicts the pattern verifier and the bottom part is its corresponding part of abstracted pattern verifier. The gray nodes represent ambiguous states with respect to the pattern Ω. This depicted part is actually a partial critical path since it contains an ambiguous state cycle with one observable event for both involved components G_1 and G_2. Note that in the abstracted pattern verifier, we do not retain those ambiguous state cycles with observable events for only some involved components but not all of them. For example, in figure 4.5, the abstracted one (bottom) does not preserve the ambiguous state cycle with only observable event $O4$ (top) but preserves the ambiguous state cycle with the observable events $O1$ and $O4$, which are represented by Obs_1 and Obs_2. The reason is that the ambiguous state cycle with only $O4$ will never be extended to an ambiguous state cycle containing observable events for all components, whose existence in the path is a necessary and sufficient condition for being a global critical path. So to search for global critical paths, in the abstracted pattern verifier, what we are interested in is those ambiguous state cycles with observable events for all involved components.

4.2.4 Global consistency checking

The existence of global critical paths implies the existence of partial critical paths in the abstracted pattern verifier but not each obtained partial critical path corresponds to a global critical path. The reason is that for now we do not take into account the communication of partial critical paths with the neighborhood of the current subsystem in the whole system. To solve this, we need to use the abstracted local twin checker of a component whose construction is described in section 3.2 and in section 3.3, which is to obtain all pairs of local trajectories with the same observations. And then the partial critical paths are synchronized with the abstracted local twin checker of the connected components to check their global consistency. Now we define the global consistency of partial critical path as follows.

Definition 28 *(Global consistency). Given a partial critical path, it is globally consistent if after synchronizing with abstracted local twin checkers of all connected components, it still contains an ambiguous state cycle with at least one observable event for all involved components.*

Algorithm 2 Global Consistency Checking for Abstracted Pattern Verifier

1: INPUT:
   component models $G_1, ..., G_n$ of the system G;
   the abstracted pattern verifier V^a;
   the current subsystem G_S, which is the subsystem corresponding to V^a
2: $V^a \leftarrow Reduce(V^a)$
3: **while** $V^a \neq \emptyset$ and $ConnectComp(G_S) \neq \emptyset$ **do**
4: $\quad G_j \leftarrow Select(ConnectComp(G_S))$
5: $\quad C_j^a \leftarrow ConstructALTC(G_j)$
6: $\quad V^a \leftarrow Sync(V^a, C_j^a)$
7: $\quad V^a \leftarrow Reduce(V^a)$
8: $\quad G_S \leftarrow Add(G_S, G_j)$
9: **end while**
10: **if** $V^a \neq \emptyset$ **then**
11: $\quad$ return V^a
12: **else**
13: $\quad$ return G_S
14: **end if**

Algorithm 2 provides the pseudo-code of global consistency checking procedure for the abstracted pattern verifier. With the set of component models, the abstracted pattern verifier V^a and the current subsystem G_S as input, the abstracted pattern verifier V^a is first reduced by only retaining all partial critical paths (line 2). When the reduced abstracted pattern verifier is not empty and

there exists at least one component G_j that is not involved in the current subsystem G_S but neighboring to G_S (line 3), which means that G_j contains at least one communication event contained also in G_S, then the algorithm repeatedly performs the following steps:

- Select one connected component G_j and construct its abstracted local twin checker C_j^a. (line 4-5)
- Synchronize the abstracted local twin checker C_j^a with the reduced abstracted pattern verifier V^a, where the synchronized events set is the set of common left and right communication events of G_S and G_j. This resulted FSM is still called an abstracted pattern verifier. (line 6)
- Reduce the newly obtained abstracted pattern verifier by only retaining all partial critical paths. Here the partial critical paths are those having an ambiguous state cycle containing at least one local observable event for all involved components including G_j and then the current subsystem G_S is updated by adding G_j. (line 7-8)

When there is no other connected component, any partial critical path obtained in the final FSM is globally consistent. If there is at least one such path, which means that the final FSM is not empty, then this FSM is returned to provide the non-diagnosability information (line 10-11). Otherwise, if there is no such path, which means that the current reduced abstracted pattern verifier is empty, then the current subsystem G_S is a diagnosable subsystem with respect to Ω, which is returned by our algorithm (line 12-13).

Lemma 8 *A partial critical path is globally consistent iff it corresponds to a global critical path.*

Proof :

($\Rightarrow$) Suppose that a partial critical path ρ is globally consistent and that ρ does not correspond to a global critical path. Since ρ has no corresponding global critical path, then either it has no corresponding path in the global pattern verifier or its corresponding path in the global pattern verifier has no ambiguous state cycle with observable events for all components. And then from the way to construct the global pattern verifier and from the way to check global consistency, it is easy to know that after global consistency checking, i.e., after synchronizing the abstracted pattern verifier with connected abstracted local twin checkers, ALTCs, any ambiguous state cycle in ρ either disappears or does not contain observable events for all involved components. It follows that ρ is not globally consistent, which contradicts the assumption.

($\Leftarrow$) Suppose now that a partial critical path ρ is not globally consistent and that it corresponds to a global critical path. From the non global consistency of ρ, it follows that any ambiguous state cycle in ρ either disappears or only contains observable events for some involved components but not all of them after global consistency checking, which means that at least one communication of ρ with its neighborhood is not valid. However, any global critical path has valid communication in the whole system because it is constructed from the global model and it contains an ambiguous state cycle with observable events for all components. This implies that ρ does not correspond to a global critical path, which contradicts the assumption.

Then from lemma 8 and theorem 3, we can directly obtain the following theorem to verify pattern diagnosability in a distributed way.

Theorem 5 *A pattern Ω is diagnosable in a system G iff there is no partial critical path that is globally consistent.*

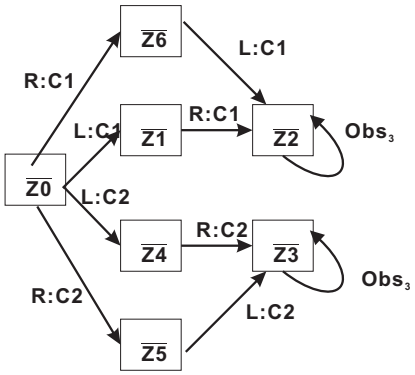

Figure 4.6: The abstracted local twin checker C_3^a for G_3.

For our example, after global consistency checking, the partial critical path depicted in figure 4.5 contains an ambiguous state cycle with only observable events for G_1 and G_2 but not for G_3 due to its synchronization with the abstracted local twin checker C_3^a of the component G_3, which means that it has no corresponding global critical path. Figure 4.6 shows the abstracted local twin checker C_3^a. In the same way, after checking all the rest part of the abstracted pattern verifier, we know that there is no partial critical path being globally consistent. Thus from theorem 5, the system is verified to be Ω-diagnosable and its diagnosable subsystem is actually the set of all three components.

Now let us change the component G_3 in the system depicted in figure 4.1. The changed component is shown in the top part of figure 4.7, which is denoted by G_3'. Then the bottom part

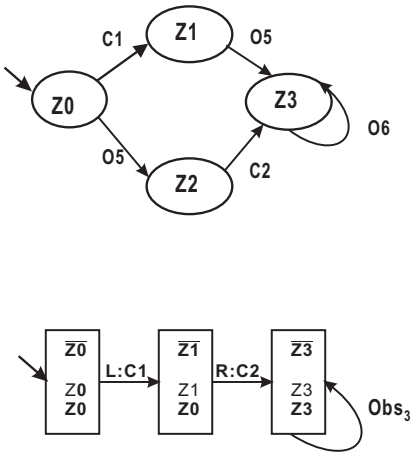

Figure 4.7: The component G'_3(top) and one path of abstracted local twin checker $C_3^a{}'$ for G'_3(bottom).

of figure 4.7 is one path of the abstracted local twin checker $C_3^a{}'$ for the component G'_3. After synchronizing the partial critical path depicted in figure 4.5 with this path of the abstracted local twin checker $C_3^a{}'$, the obtained path is shown in figure 4.8, which verifies that this partial critical path is actually globally consistent because the ambiguous state cycle does not disappear and it contains at least one observable event for all three components. So the pattern is not diagnosable in this new system, i.e., the system composed of G_1, G_2 in figure 4.1 and G'_3 in figure 4.7.

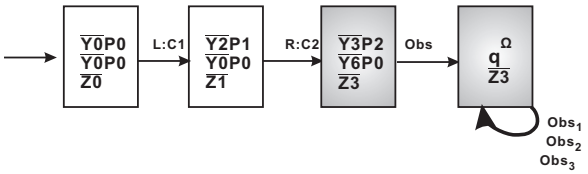

Figure 4.8: One globally consistent partial critical path.

4.2.5 Algorithm

This section presents the algorithm of simple pattern diagnosability verification in a distributed way. Note that in this algorithm, we do not need to calculate the global pattern verifier.

Algorithm 3 gives the pseudo-code of our proposed distributed simple pattern diagnosability verification procedure. With the input as the set of component models and the pattern to be diagnosed, after the initialization of the parameters, when the current recognizer is not the complete one and the next selected component with respect to the current pattern recognizer is not empty, which means that for the moment the pattern is not yet completely recognized in the cur-

Algorithm 3 Diagnosability Algorithm of Simple Pattern for Distributed Systems

INPUT:
component models $G_1, ..., G_n$ of the system G, denoted by $G = \{G_1, ...G_n\}$;
the pattern Ω to be diagnosed in G
Initializations:
$q^c \leftarrow q^0_\Omega$(currently recognized pattern state, initially the initial state of Ω);
$R \leftarrow \emptyset$ (the current recognizer, initially empty);
$G_S \leftarrow \emptyset$ (the current subsystem, initially empty)
while R is not the complete recognizer and $NextCom(G, G_S, \Omega, q^c) \neq \emptyset$ **do**
    $R \leftarrow REDUCE(R)$
    $G_i \leftarrow NextCom(G, G_S, \Omega, q^c)$
    $G_i \leftarrow Sync(G_i, R)$, where the synchronized events set is the set of common communication events of the current subsystem and the component G_i
    $R \leftarrow ConstructPR(G_i, \Omega)$
    $G_S \leftarrow Add(G_S, G_i)$
    $q^c \leftarrow q\prime$, where $q\prime$ is the pattern state in a TSS of R
end while
if R is not the complete recognizer **then**
    return "Ω cannot be recognized in G."
else
    $R \leftarrow Refine(R)$
    $V \leftarrow ConstructPV(R)$
    $V^a \leftarrow ConstructAPV(V, G_S)$
    $CheckGlobalConsistency(G, V^a, G_S)$, see algorithm 2
end if

rent subsystem and there exists the next component that we need to exploit for next recognition, the algorithm repeatedly performs the following steps.

1. The current recognizer is reduced as described in Section 4.2.2, doing nothing for the current recognizer being empty (for the first time), and then the next component G_i is selected for extending the subsystem, where G_i contains the event $\varpi^s_{q^c}$ and not involved in the current subsystem G_S. This strategy of next component selection is described in Section 4.2.1. (line 4-5)

2. The reduced recognizer is synchronized with the selected component based on the set of common communication events of the current subsystem and the selected component and then the pattern recognizer of this synchronized FSM is constructed again. (line 6-7)

3. The current subsystem G_S is now updated by adding the selected component and then the currently recognized pattern state q^c is updated by assigning $q\prime$, which is the pattern state in a TSS of the current recognizer. (line 8-9)

When the current recognizer is not the complete one and there does not exist the next component, since the pattern events set is the same as the events set of the system, there is only one reason leading to this situation. The reason is that the component containing $\varpi^s_{q^c}$ is in the current subsystem. In other words, the occurrence order of significant events in the current subsystem is not the same as that in the pattern. So the pattern cannot be recognized in the system and our algorithm returns information about non-recognition of the pattern. (line 11-12)

Otherwise, if the complete recognizer is obtained, we check pattern diagnosability first by refining this recognizer through the delay closure as described in Section 4.2.3 and then by constructing the corresponding pattern verifier as well as abstracted pattern verifier for global consistency checking (line 14-17). There are several causes that can stop this algorithm.

- The pattern cannot be recognized in the system, then the algorithm returns non-recognition information.

- The pattern can be recognized but is not diagnosable, then the returned FSM of function $CheckGlobalConsistency(G, V^a, G_S)$, which is described in algorithm 2, contains the information about the undistinguishable behaviors that cause the non-diagnosability of the pattern.

- The pattern can be recognized and is diagnosable, then $CheckGlobalConsistency$ (G, V^a, G_S) returns a diagnosable subsystem with respect to the pattern.

4.3 Extension to general patterns

Now we investigate how to extend pattern diagnosability in a distributed way from the simple pattern case to the general pattern case, which is more complicated considering that a general pattern could be composed of multiple or even infinite number of simple patterns.

As in the simple pattern case, we still use $\widehat{\varpi_q}$ to denote the set of significant events of Ω that change the state q. The difference is that in a simple pattern, $|\widehat{\varpi_q}|$, the number of such significant events is only one if q is not a final state while in a general pattern, $|\widehat{\varpi_q}|$ could be multiple if q is not a final state. Before pattern recognition, for the sake of simplicity, we first merge final states in the following way:

$$\forall \rho, \text{where } \rho \text{ is a path in } \Omega, \text{such that } q \xrightarrow{\sigma_1} q_1 ... \xrightarrow{\sigma_n} q_n \xrightarrow{\sigma\prime} q\prime,$$
$$\text{where } \{q_1, ..., q_n, q\prime\} \subseteq F_\Omega, q \notin F_\Omega \text{ and } \{\sigma_1, ...\sigma_n, \sigma\prime\} \subseteq \Theta_\Omega,$$
$$\text{it is modified as a path } \rho\prime = q \xrightarrow{\sigma_1} q_1 \xrightarrow{\Sigma_\Omega} q_1.$$

This means that in Ω, the set of final states Γ_q with the same last preceding state q that is not a final state and the same last transition σ_1 can be merged into one final state. Since the set of final states of Ω is stable, then this operation will not have an impact on the correctness of diagnosability algorithm except to make it more simpler. Figure 4.9 shows one example of a general pattern, denoted by $\Omega\prime$. The difference of $\Omega\prime$ from the simple pattern depicted in figure 4.2 is that the significant events of $\widehat{\varpi_{p1}}$ are $O3$ and $U2$. In $\Omega\prime$, the final state will be achieved when the occurrence of $O3$ is after the occurrence of $U1$ and there is no occurrence of $U2$ between $U1$ and $O3$.

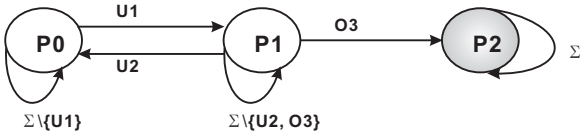

Figure 4.9: One example of a general pattern $\Omega\prime$.

For a subsystem G_S, we use the definition 25 to construct its pattern recognizer R_{G_S}. Then for the reduction of pattern recognizer by retaining only diagnosability relative part $R^{\Omega}_{G_S}$, the use of Target Suspicious State (TSS) defined in section 4.2.1 is not appropriate considering that a general pattern may contain multiple simple patterns and thus multiple final states even after the mergence described as above. So different from TSS , we define recognition relative paths and diagnosability relative paths as follows.

Definition 29 *(Recognition relative paths and Diagnosability relative paths).*

- *Given an infinite path ρ in the pattern recognizer of the subsystem G_S, if $\exists q_r = (q, q_\Omega) \in \rho$, such that either q_Ω is a final state of the pattern or $\exists \sigma \in \widehat{\varpi_{q_\Omega}}$ such that $\sigma \in \Sigma \backslash \Sigma_S$, where Σ_S is the events set of G_S and Σ is the events set of the entire system G, then ρ is a recognition relative path. And σ is called a next recognizable event with respect to the subsystem G_S. The set of next recognizable events with respect to G_S is denoted by Λ_{G_S}.*

- *Given an infinite path ρ in the pattern recognizer, if it is a recognition relative path or it has the same observations as any recognition relative path of this pattern recognizer, then it is a diagnosability relative path.*

Intuitively, a recognition relative path of the pattern recognizer contains either at least one final state of the recognizer or at least one state such that it is the source state of a significant event in the pattern that changes it and that is contained outside of the current subsystem. Only such kind of paths can possibly recognize the pattern after synchronizing with other components that are not in the current subsystem. And such a state is similar to TSS in the simple pattern case with the only difference that TSS pattern state is unique while this kind of states in the general pattern case can be multiple. The set of recognition relative paths contains the corresponding subpart of all global trajectories that recognize the considered pattern. So similar to the simple pattern case, the pattern recognizer is reduced to $R^{\Omega}_{G_S}$ by only retaining all diagnosability relative paths in R_{G_S}. Then it is easy to prove the following lemma since the set of diagnosability relative paths includes not only recognition relative paths but also all the paths with the same observations as any recognition relative path.

Lemma 9 *The reduced pattern recognizer $R^{\Omega}_{G_S}$ contains the corresponding subpart in the subsystem G_S of all global critical pairs.*

Consider the system depicted in figure 4.1 and the general pattern $\Omega\prime$ depicted in figure 4.9. The top part of figure 4.10 shows the pattern recognizer for the component G_1 and the bottom part is its diagnosability relative paths. In the pattern recognizer, from definition 29, we know that there is only one next recognizable event, $O3$. The path $(X0P0 \xrightarrow{U1} X1P1 \xrightarrow{C1} X3P1 \xrightarrow{U2} X4P0 \xrightarrow{O1} X4P0)$ is the recognition relative path, which has the same observations as the path $(X0P0 \xrightarrow{C2} X3P0 \xrightarrow{U2} X4P0 \xrightarrow{O1} X4P0)$. So we get all diagnosability relative paths shown in the bottom part of figure 4.10.

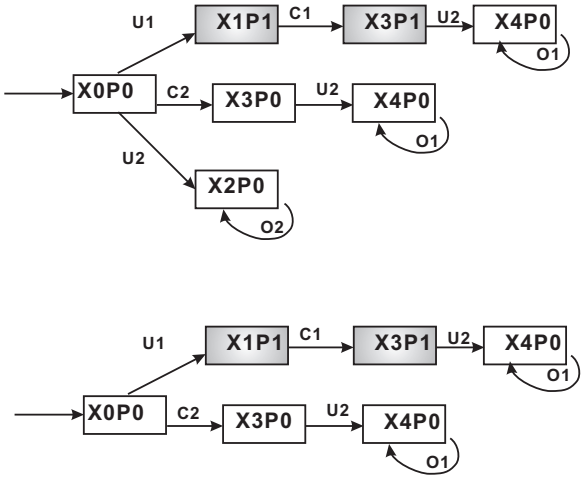

Figure 4.10: The pattern recognizer R_{G_S} for the initial subsystem, i.e., the component G_1(top) and its set of diagnosability relative paths $R^{\Omega}_{G_S}$ (bottom).

In the simple pattern case, we define the complete recognizer as a pattern recognizer that contains at least one final state. The reason is that since there is only one final state in a simple pattern, then if there is a final state in a pattern recognizer, this means that the simple pattern is completely recognized in the current subsystem and we do not need to exploit another component for next pattern recognition. However, in the general pattern case, since there could be multiple simple patterns as well as multiple final states, the complete recognizer is defined as a pattern recognizer with at least one final state and with no next recognizable event with respect to the current subsystem. In other words, a pattern recognizer R_{G_S} is the complete recognizer if it satisfies two conditions:

- in R_{G_S}, $\exists q \in Q_{R_{G_S}}$ such that $q \in F_{R_{G_S}}$, where $F_{R_{G_S}}$ is the set of final states of the pattern recognizer R_{G_S};
- $\Lambda_{G_S} = \emptyset$.

So the complete recognizer means that the pattern can be recognized in the current subsystem and there is no other component that should be further exploited for next pattern recognition. If the current pattern recognizer R_{G_S} is not the complete recognizer, then there are three cases:

1. there is no final state in R_{G_S} and the set of next recognizable events is not empty $\Lambda_{G_S} \neq \emptyset$;
2. there exists at least one final state in R_{G_S} and the set of next recognizable events is not empty $\Lambda_{G_S} \neq \emptyset$;

3. there is no final state in R_{G_S} and the set of next recognizable events is empty $\Lambda_{G_S} = \emptyset$;

In case 1 and case 2, where case 1 means that the pattern is not recognized in the current subsystem and there exists at least one next recognizable event and case 2 means that the pattern is recognized in the current subsystem and there exists at least one next recognizable event for next recognition, we select a component G_j that contains at least one next recognizable event with respect to G_S. In other words, we have $\Sigma_j \cap \Lambda_{G_S} \neq \emptyset$. Note that the number of events in Λ_{G_S} is not necessarily only one since the pattern under consideration is a general pattern, so G_j is not unique in general but the order of selection is however not influential for pattern recognition. Case 3 means that the pattern is not recognized in the current subsystem and there is no next recognizable event. In other words, case 3 implies that the pattern cannot be recognized in the whole system.

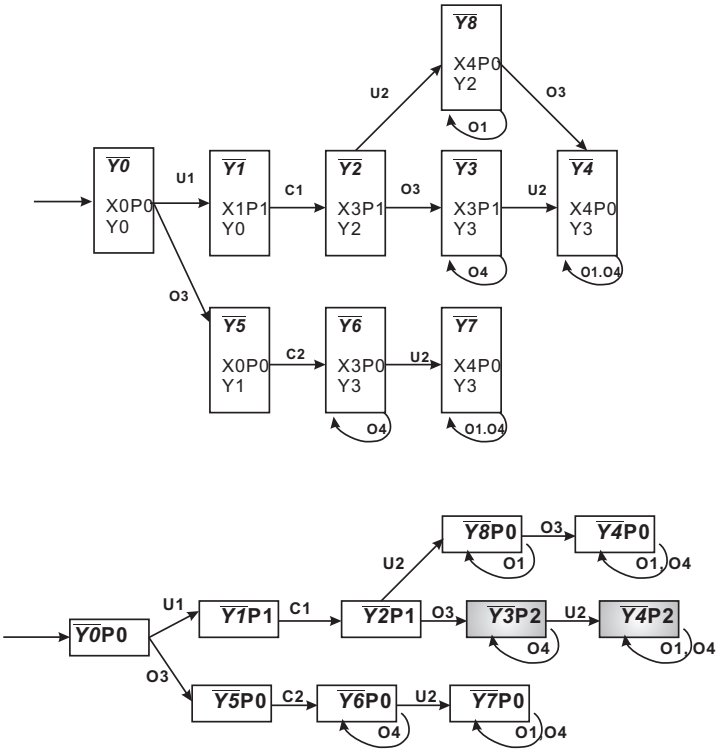

Figure 4.11: Part of the extended subsystem $R^{\Omega}_{G_S} \| G_2$ (top) and part of pattern recognizer for this extended subsystem (bottom).

A part of the extended subsystem is shown in the top part of figure 4.11, that is obtained by synchronizing the diagnosability relative paths in the pattern recognizer for the component G_1 with the component G_2, where G_2 contains the next recognizable event $O3$. Then the bottom part in the figure is a part of pattern recognizer for the extended subsystem. Then this pattern

recognizer is actually the complete recognizer since there is no next recognizable event and there does exist final states in it. Next we refine the complete recognizer through the delay closure with respect to the set of observable events and communication events and then construct the pattern verifier to get the abstracted pattern verifier. The top part of figure 4.12 depicts one partial critical path in the abstracted pattern verifier, which can be verified to be globally consistent when it is synchronized with the abstracted local twin checker for the component G_3 shown in figure 4.6. This globally consistent partial critical path is depicted in the bottom part of figure 4.12. So the general pattern $\Omega\prime$ shown in figure 4.9 is not diagnosable in the system depicted in figure 4.1 while the simple pattern Ω shown in figure 4.2 is diagnosable in the same system.

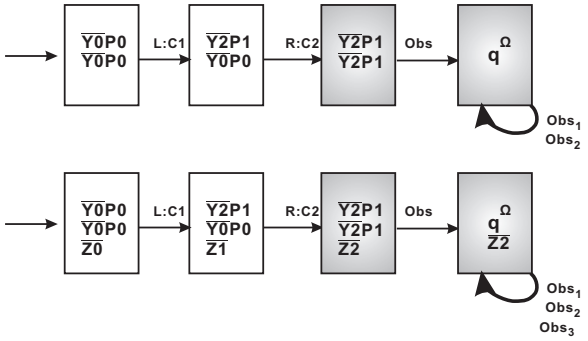

Figure 4.12: One partial critical path in the abstracted pattern verifier (top) and the corresponding globally consistent partial critical path (bottom).

The algorithm 4 describes the distributed diagnosability verification for general patterns. With the input as the set of component models and the pattern under consideration, the parameters of the algorithm are initialized. Then when there exists at least one next recognizable event with respect to the current subsystem (line 3), which means that there are other components that should be further exploited for next pattern recognition, then the following steps are repeatedly performed.

1. The current pattern recognizer is first reduced by only retaining diagnosability relative paths, doing nothing for the current recognizer being empty (for the first time), and then one component containing at least one next recognizable event is selected (line 4-5).

2. The reduced recognizer is then synchronized with the selected component based on the set of common communication events of the current subsystem and the selected component and then the pattern recognizer for this synchronized FSM is again constructed (line 6-7).

3. The current subsystem is now updated by adding the selected component and then the set

Algorithm 4 Diagnosability Algorithm of General patterns for Distributed System

1: INPUT:
component models $G_1, ..., G_n$ of the system G, denoted by $G = \{G_1, ...G_n\}$;
the pattern Ω to be diagnosed in G
2: Initializations:
$G_S \leftarrow \emptyset$ (the current subsystem, initially empty);
$R \leftarrow \emptyset$ (the current pattern recognizer, initially empty);
$\Lambda_{G_S} \leftarrow \widehat{\varpi_{q_\Omega^0}}$ (the set of next recognizable events with respect to the current subsystem, initially the set of significant events of Ω that change its initial state q_Ω^0);
3: **while** $\Lambda_{G_S} \neq \emptyset$ **do**
4: $R \leftarrow REDUCE(R)$
5: $G_i \leftarrow SelectCom(\Lambda_{G_S}, G)$
6: $G_i \leftarrow Sync(G_i, R)$, where the synchronized events set is the set of common communication events of the current subsystem G_S and G_i
7: $R \leftarrow ConstructPR(G_i, \Omega)$
8: $G_S \leftarrow Add(G_S, G_i)$
9: $\Lambda_{G_S} \leftarrow CollectNRE(R, G_S, G, \Omega)$
10: **end while**
11: **if** R is not the complete recognizer **then**
12: return "Ω cannot be recognized in G."
13: **else**
14: $R \leftarrow Refine(R)$
15: $V \leftarrow ConstructPV(R)$
16: $V^a \leftarrow ConstructAPV(V, G_S)$
17: $CheckGlobalConsistency(G, V^a, G_S)$
18: **end if**

of next recognizable events with respect to the current subsystem is updated as described in definition 29 (line 8-9).

When there is no next recognizable event and the current pattern recognizer is not the complete one, which means that there is no final state in this recognizer, it can be deduced that the pattern can never be recognized in the system. In this case, our algorithm returns the information about non recognizability of the pattern (line 11-12). Otherwise, i.e., the current pattern recognizer is the complete one, we first refine the complete recognizer and then construct the pattern verifier to get the abstracted pattern verifier before checking its global consistency. All these steps (line 14-17) are the same as in the distributed diagnosability algorithm for the simple pattern case, i.e., algorithm 3.

Chapter 5

Implementation and validation

In chapter 4, the correctness and efficiency of our proposed distributed pattern diagnosability algorithm have been theoretically proved. In this chapter, we show the implementation and validation from a practical point of view. Furthermore, to compare with the centralized approach for pattern diagnosability, we also implement the centralized algorithm ([43]). Our results emphasize that the search space of distributed algorithm is much smaller than that of centralized one in most cases.

5.1 Implementation

The implementation is coded in Java that is currently one of the most popular programming languages in use because of its reflexibility, scalability, simplicity, etc. From pattern diagnosability verification procedure, we can see that the software architecture is based on different types of FSMs, including subsystem, pattern, pattern recognizer, pattern verifier, local twin checker, etc. The implementation is based on the classes of FSMs. Considering that if the number of faults is high we will face a significant increase in complexity, it is better to check the diagnosability individually for each fault.

5.1.1 Flowchart of procedures

Figure 5.1 shows the flowchart of distributed pattern diagnosability verification procedure (see its formal algorithm 4). We recall its major steps as follows.

1. The pattern recognizer for a subsystem is obtained by the product of the subsystem model and the pattern. If there exists at least one next recognizable event outside of the current subsystem, then we reduce the pattern recognizer by retaining only diagnosability relative

paths and select one component containing such a next recognizable event. Then the reduced pattern recognizer is synchronized with the selected component to extend the current subsystem. Here next recognizable events and diagnosability relative paths are defined by definition 29 in section 4.3.

2. We repeat the above step until there is no next recognizable event. Then if the current pattern recognizer is not the complete one, i.e., there is no final state in the current recognizer, this means that the pattern cannot be recognized in the system and thus the non recognizability is returned. Otherwise, if the current pattern recognizer is the complete one, then the pattern verifier is constructed as described by definition 26 in section 4.2.3. To improve the efficiency, the abstracted pattern verifier is calculated from the pattern verifier as described by definition 27 in section 4.2.3.

3. In the current abstracted pattern verifier, if there exists at least one consistent ambiguous cycle, where a consistent ambiguous cycle is a cycle containing only ambiguous states with at least one observable event for all involved components in the current subsystem, then the abstracted pattern verifier is reduced to retain only those paths with consistent ambiguous cycles. Otherwise, if there does not exist consistent ambiguous cycle in the current abstracted pattern verifier, then the current subsystem is returned, which is a diagnosable subsystem with respect to the pattern.

4. If there does not exist a component neighboring to the current subsystem, then the current reduced abstract pattern verifier is returned to provide some information about the reasons why the pattern is not diagnosable. Otherwise, if there exists at least one connected component,i.e., a component containing at least one communication event that is also contained in the current subsystem, then one such component is selected before constructing its local twin checker. And the abstracted local twin checker is obtained from the local twin checker as described by definition 21 in section 3.3.2. Then the abstracted pattern verifier is updated by synchronizing the current reduced abstracted pattern verifier with the abstracted local twin checker. In the same way, the algorithm repeats step 3 and step 4 until either a diagnosable subsystem is returned when the pattern is diagnosable in the system or the current reduced abstracted pattern verifier is returned when the pattern is not diagnosable in the system.

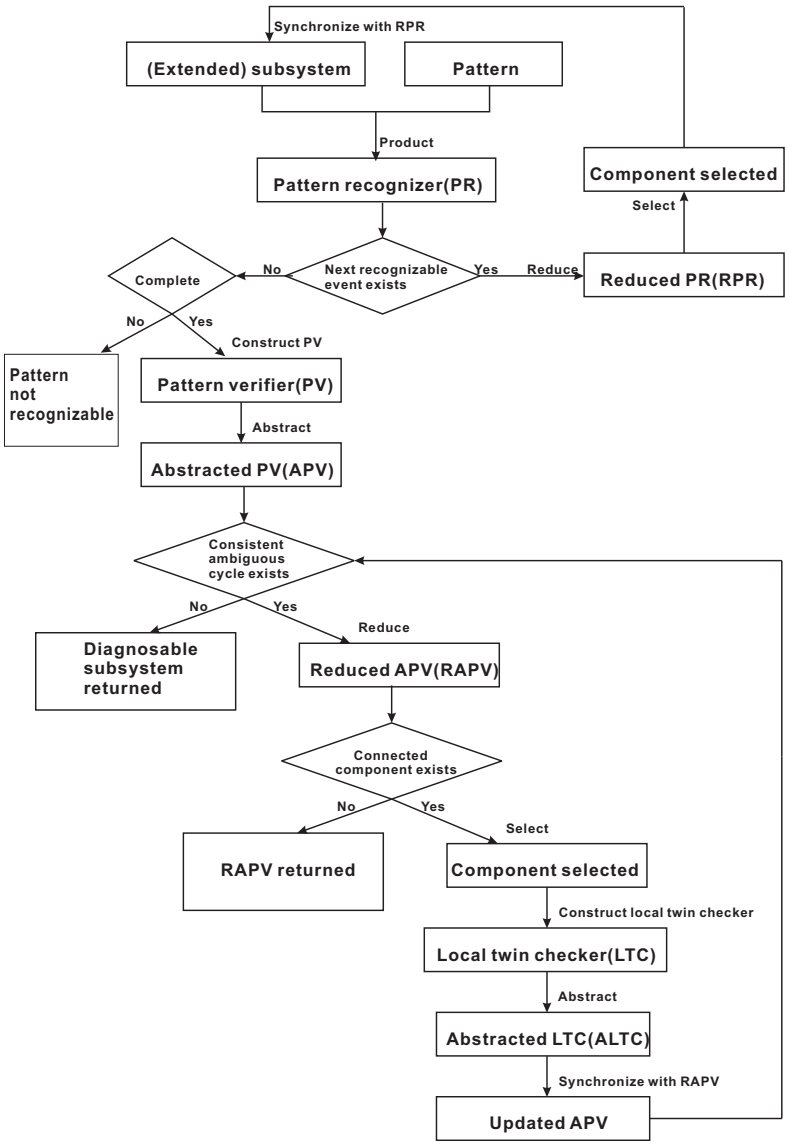

Figure 5.1: Flowchart of distributed pattern diagnosability checking procedure.

To compare with the centralized approach, we also implement the centralized algorithm. To be more clear, we present here its formal algorithm and the flowchart of its procedure.

Algorithm 5 Centralized Pattern Diagnosability Algorithm

1: INPUT:
   component models $G_1, ..., G_n$ of the system G, denoted by $G = \{G_1, ..., G_n\}$;
   the considered pattern Ω
2: Initialization:
   global model $G_S \leftarrow \emptyset$
3: **while** $|G_S| \neq |G|$ **do**
4: $G_i \leftarrow SelectComp(G_S, G_1, ..., G_n)$
5: $G_S \leftarrow Sync(G_S, G_i)$
6: **end while**
7: $R \leftarrow ConstructGPR(G_S, \Omega)$
8: **if** there does not exist final state in R **then**
9: return "pattern is not recognizable in the system"
10: **else**
11: $V \leftarrow ConstructGPV(R)$
12: **if** there exists at least one global critical path in V **then**
13: $V \leftarrow Reduce(V)$
14: return V
15: **else**
16: return "pattern is diagnosable in the system"
17: **end if**
18: **end if**

Algorithm 5 presents the centralized pattern diagnosability checking procedure and its flowchart is depicted in figure 5.2. We describe its major steps as follows.

1. The global model of the system is obtained by synchronizing all components (line 3-6 in algorithm).

2. When there is no component outside of the current subsystem, which means that this subsystem is actually the global model, then the global pattern recognizer is constructed through the product of the pattern and the global model (line 7).

3. If there is no final state in the global pattern recognizer, which means that the pattern cannot be recognized in the system, then the non recognizability information is returned (line 8-9). Otherwise, if there does exist at least one final state, the global pattern verifier is constructed first by operating the delay closure on the pattern recognizer with respect to the set of observable events and then by synchronizing the obtained FSM with itself based on the set of observable events (line 10-11).

4. In the global pattern verifier, if there exists at least one consistent ambiguous state cycle, i.e., a cycle containing ambiguous states with at least one observable event for all components, whose corresponding path is actually a global critical path, then the global pattern verifier is reduced to retain only all global critical paths, which are returned by the algorithm (line 12-14). Otherwise, if there is no consistent ambiguous state cycle, the pattern is diagnosable in the system and thus the diagnosability information is returned (line 15-18).

5.1.2 Comparison

Now consider the different major steps in the centralized approach and in the distributed approach from figure 5.2 and figure 5.1 to understand the efficiency of the distributed algorithm. In total, the differences lie in two major aspects: pattern recognizability analysis and pattern diagnosability verification.

- For pattern recognizability analysis, in the centralized approach, the global model is constructed before the global pattern recognizer construction. While in the distributed approach, we incrementally extend subsystem by synchronizing the diagnosability relative paths of the current subsystem with the next selected component, i.e., the component containing at least one next recognizable event. In this way, we can significantly save the state space because in real complex systems, the part of diagnosability relative paths in the subsystem is normally much smaller than the whole subsystem. Furthermore, figure 5.1 shows that the components that do not contain any next recognizable event in every step of subsystem extension are not involved in pattern recognizability analysis.

- For pattern diagnosability checking, in the centralized approach, the global pattern verifier is constructed by synchronizing the refined global pattern recognizer with itself, i.e., the synchronization of two identical instances. While in the distributed approach, the pattern verifier is obtained by synchronizing the reduced left instance of the refined complete pattern recognizer with its reduced right instance, where the reduced left instance contains only the paths with at least one consistent ambiguous state cycle that contains at least one observable event for all involved components and the reduced right instance contains only the paths with at least one cycle without ambiguous state but with at least one observable event for all involved components. This difference greatly improves the efficiency. Furthermore, to check the global consistency of partial critical paths, before synchronization, we

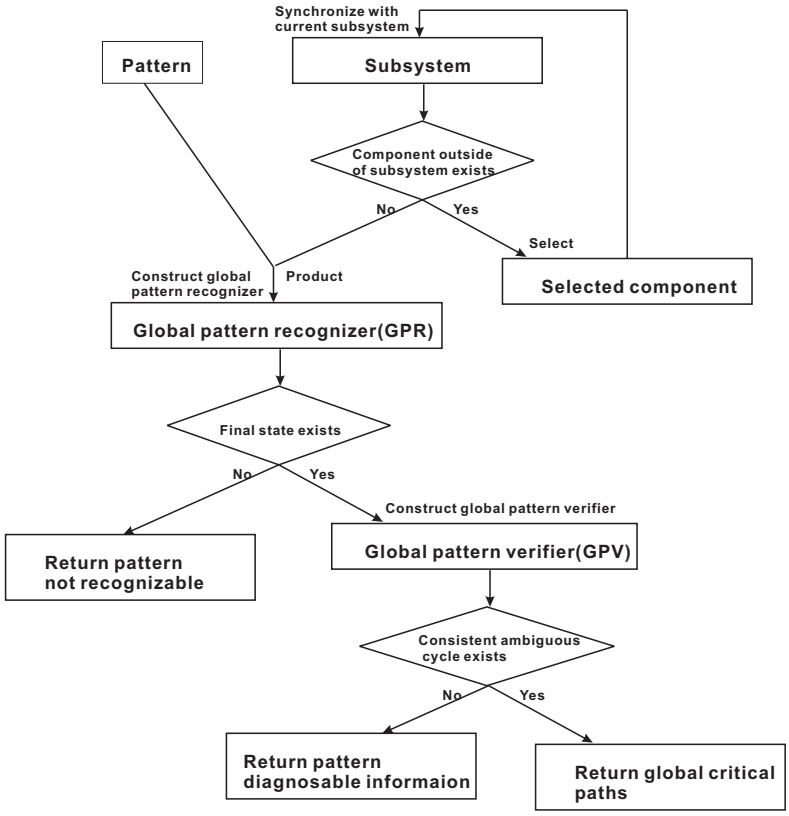

Figure 5.2: Flowchart of centralized pattern diagnosability checking procedure.

abstract necessary and sufficient information from pattern verifier and local twin checkers, i.e., abstracted pattern verifier and abstracted local twin checkers. The synchronization of abstracted ones is simpler than that of non abstracted ones.

We can see that abstraction and reduction play an important role to save search space compared to the centralized method even to normal distributed method without abstraction and reduction. While the abstraction and reduction mainly consist in searching ambiguous state cycles containing observations for all involved components. In our implementation, we separate cycles into two types: elementary cycles that contain no other cycle and embedded cycles that contain at least one other cycle. More precisely, the data structure of elementary cycle is a set of transitions, where there is only one pair of transitions such that the source state of one transition is the same as the destination state of the other transition. And searching elementary cycles is linear in the number of states and transitions of the corresponding FSM through depth-first search. For embedded cycles, all elementary cycles that share at least one state constitute an embedded cycle, where we can check observations for all concerned components. So searching ambiguous state cycles, no matter the cycle being elementary or embedded, is linear in the number of states and transitions in the concerned FSM in each step during diagnosability checking. We will see the search space reduction of the distributed approach compared to the centralized one in the next section through examples.

5.2 Validation

As said before, we have theoretically proved the correctness and efficiency of the distributed pattern diagnosability algorithm, i.e., with the same result, diagnosable or not diagnosable, as the centralized one while with smaller search space than the centralized one. In this section, we show the results of some examples to validate its properties.

5.2.1 Test case

The test case that we adopt is a simple example of an office system composed of three components including a file processer component, a scanner component and a photocopier component. This system is depicted in figure 5.3. The file processer component is to manipulate files, like creating file ($Create\_file$), deleting file ($Delete\_file$), modifying file ($Modify\_file$) and reading file ($Read\_file$). Furthermore, it can send the request to the scanner component to scan the

Table 5.1: Search space of pattern diagnosability checking for $G1$ and Ω_1

Centralized algorithm	Distributed algorithm
global pattern recognizer states number: 22 transitions number: 46	complete pattern recognizer states number: 17 transitions number: 33
global pattern verifier states number: 80 transitions number: 173	the pattern verifier after global checking states number: 24 transitions number: 41

file ($Scan\_request$). We suppose that the operation of $Delete\_file$ is an unobservable event, the operation of $Scan\_request$ is an unobservable communication event and all other events are observable events. The scanner component can execute two tasks. One is that when receiving the scan request ($Scan\_request$) from the file processer, then the Scan task is performed. The other task is the manual scan. In this case, what it receives is not the $Scan\_request$ from the file processer but the order from the environment outside of the system, e.g. one person manually pressing the scan button. Then the scanner performs the Scan task and saves the scanned file by sending it to the appropriate email address ($Send\_email$). And the photocopier component only executes the Copy task according to the order from outside of the system. We suppose that $Scan$, $Send\_email$ and $Copy$ are observable events as well as $Wait\_process$, $Wait_scan$ and $Wait_copy$.

Then the pattern under investigation that defines the faulty behavior with respect to this office system is shown in figure 5.4, denoted by Ω_1. The faulty behavior predefined for this system is that the occurrence of Scan is after the occurrence of $Delete\_file$ and there is no operation of $Create\_file$ between these two events. In other words, the scanner cannot scan an empty file. The events set of this pattern is the same as that of the whole system.

Tables 5.1, 5.2 and figures 5.5, 5.6 show the search space of the distributed pattern diagnosability algorithm and that of the centralized one for two system examples.

- The system with only two components in figure 5.3, denoted by $G1$: the file processer and the scanner, and the pattern Ω_1 that is shown in figure 5.4.
- The system with three components in figure 5.3, denoted by $G2$: the file processer, the scanner and the photocopier, and the pattern Ω_1.

During the diagnosability checking procedure, in the distributed algorithm, we calculate the number of states and the number of transitions of the complete pattern recognizer, which is pre-

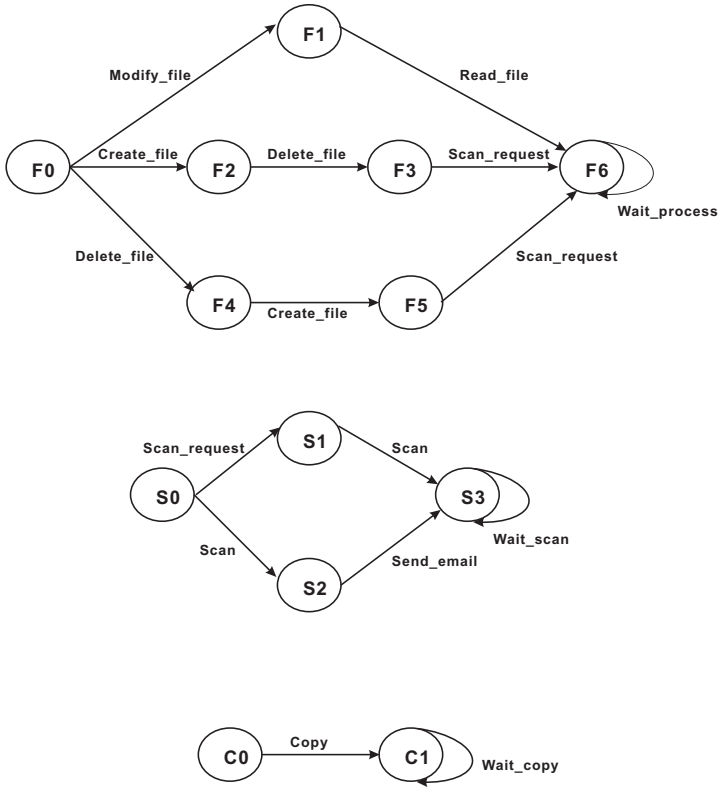

Figure 5.3: A distributed system composed of a file processer component (top), a scanner component (middle) and a photocopier component (bottom).

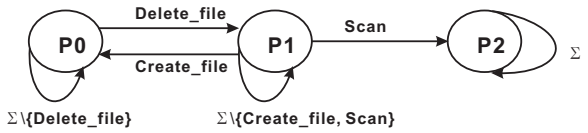

Figure 5.4: The pattern Ω_1 for the system depicted in 5.3.

Table 5.2: Search space of pattern diagnosability checking for $G2$ and Ω_1

Centralized algorithm	Distributed algorithm
global pattern recognizer states number: 44 transitions number: 136	complete pattern recognizer states number: 17 transitions number: 33
global pattern verifier states number: 198 transitions number: 809	the pattern verifier after global checking states number: 24 transitions number: 41

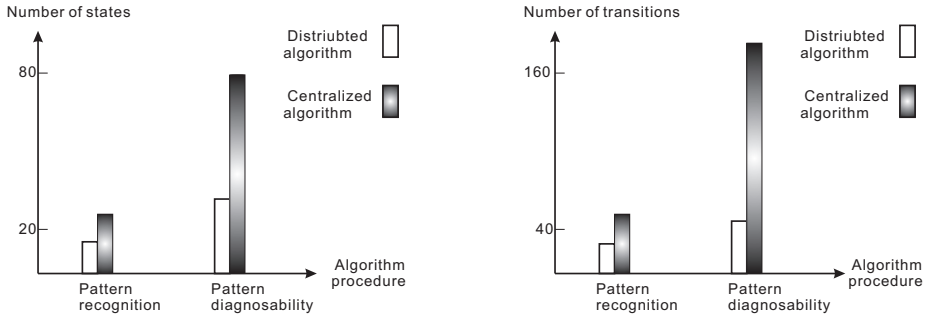

Figure 5.5: The search space of pattern recognition and pattern diagnosability for the system $G1$ and Ω_1.

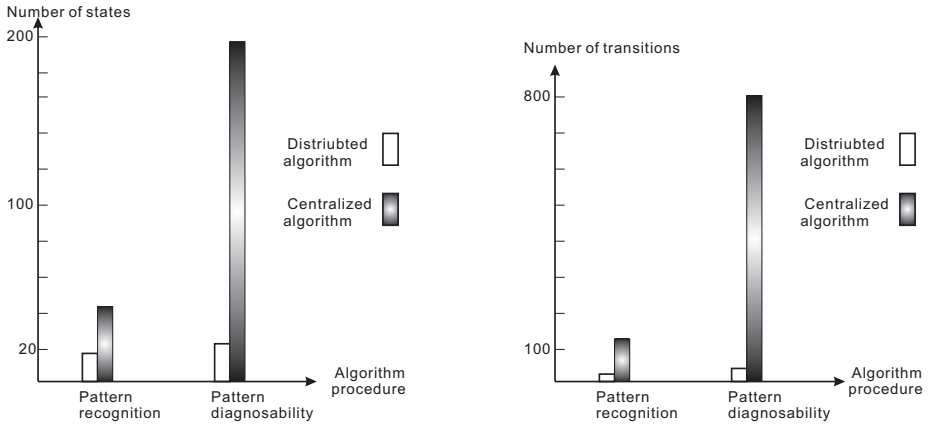

Figure 5.6: The search space of pattern recognition and pattern diagnosability for the system $G2$ and Ω_1 .

sented as pattern recognition in the figures. Correspondingly, in the centralized one, we calculate the number of states and the number of transitions of the global pattern recognizer, which is called pattern recognition in the figures. Furthermore, in the distributed one, we compute the number of states and the number of transitions of the finally obtained pattern verifier after the global consistency checking, which is called pattern diagnosability in the figures. Then in the centralized one, we compute the number of states and the number of transitions of the global pattern verifier that is called pattern diagnosability in the figures. In this way, we can clearly show the efficiency improvement of the distributed algorithm compared to the centralized one, i.e., search space is greatly reduced. We see that the search space is even much more reduced for the system $G2$. The reason is that for $G2$, in the distributed framework, the pattern recognition can be completed in the subsystem of two components: the file processer and the scanner. Then since the component of the photocopier has no communication event, the original diagnosability information obtained from the pattern verifier does not need global consistency checking considering that the system is com-

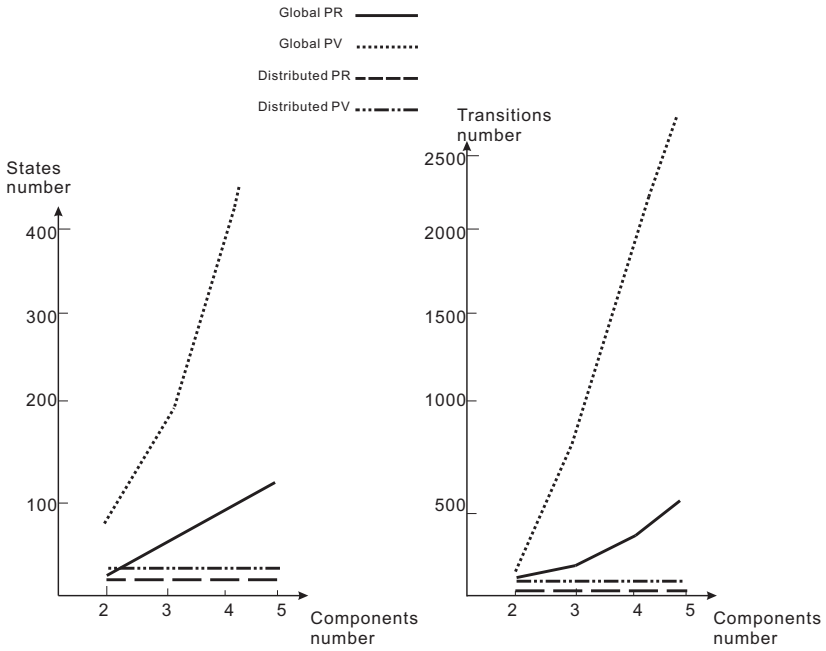

Figure 5.7: The search space growth when adding simple independent components.

posed of three components and the photocopier component is independent without any connection with other components. In other words, neither pattern recognition nor pattern diagnosability checking requires any information of the photocopier component. While in the centralized framework, for $G2$, we first synchronize the three components to get the global model and the global pattern recognizer before the global pattern verifier construction. For these two systems, both the centralized algorithm and the distributed one return some non diagnosability information, which means that the pattern is not diagnosable in $G1$ and $G2$. Now we can say that if the system has more independent components, i.e., components not connected to the subsystem where the pattern is completely recognized, the distributed algorithm can hugely reduce the search space. Figure 5.7 shows the growth of states number and of transitions number in the global pattern recognizer, the global pattern verifier, the distributed complete pattern recognizer and the distributed pattern verifier when the system $G1$ is extended by adding more independent components, where the system with components number 3 is actually $G2$ mentioned above. We can see that since the added components are independent, then for distributed approach, the search space of complete pattern recognizer (distributed PR) and of the distributed pattern verifier (distributed PV) never increase while for centralized approach, the search spaces of the global pattern recognizer (global PR) and of the global pattern verifier (global PV) dramatically increase (here we only consider the simplest

Table 5.3: Search space of pattern diagnosability checking for $G3$ and Ω_2

Centralized algorithm	Distributed algorithm
global pattern recognizer states number: 14 transitions number: 35	complete pattern recognizer states number: 8 transitions number: 14
global pattern verifier states number: 48 transitions number: 179	the pattern verifier after global checking states number: 8 transitions number: 18

added components, each one with two events and two states like the photocopier component in figure 5.3), which is consistent with the complexity analysis described in section 5.1.2.

Next we show the results of some other system examples. Consider the following systems and patterns.

- The system composed of three components that are depicted in figure 4.1, here denoted by $G3$.
- The system composed of three components, two of them are G_1, G_2 in figure 4.1 and another one is G_3' in figure 4.7, denoted by $G4$.
- The pattern depicted in figure 4.2, here denoted by Ω_2
- The pattern depicted in figure 4.9, denoted by Ω_3.

Then tables 5.3, 5.4, 5.5, and 5.6 show the search space of the centralized pattern diagnosability algorithm and that of the distributed one for the systems $G3, G4$ with respect to the pattern Ω_2 and the systems $G3, G4$ with respect to Ω_3 respectively. From these tables, we can see that for all these cases, the search space of the distributed approach is significantly reduced both for pattern recognition, i.e., global pattern recognizer vs. complete pattern recognizer, and for pattern diagnosability, i.e., global pattern verifier vs. the final pattern verifier after global consistency checking. And the pattern Ω_2 is diagnosable in the system $G3$ because there is no globally consistent critical path and the algorithm returns a diagnosable subsystem (G_1, G_2, G_3), which is the entire system. Then the pattern Ω_2 is not diagnosable in $G4$ and Ω_3 is not diagnosable both in $G3$ and $G4$ because in these three cases, there exists a set of globally consistent critical paths.

To show the efficiency of our method, we also compare the search space of the distributed method through abstraction and reduction and that of the distributed one without abstraction and reduction for three simple distributed systems, which are similar to the test case example except

Table 5.4: Search space of pattern diagnosability checking for $G4$ and Ω_2

Centralized algorithm	Distributed algorithm
global pattern recognizer states number: 20 transitions number: 50	complete pattern recognizer states number: 8 transitions number: 14
global pattern verifier states number: 101 transitions number: 349	the pattern verifier after global checking states number: 12 transitions number: 28

Table 5.5: Search space of pattern diagnosability checking for $G3$ and Ω_3

Centralized algorithm	Distributed algorithm
global pattern recognizer states number: 14 transitions number: 35	complete pattern recognizer states number: 10 transitions number: 19
global pattern verifier states number: 61 transitions number: 224	the pattern verifier after global checking states number: 14 transitions number: 37

Table 5.6: Search space of pattern diagnosability checking for $G4$ and Ω_3

Centralized algorithm	Distributed algorithm
global pattern recognizer states number: 20 transitions number: 50	complete pattern recognizer states number: 10 transitions number: 19
global pattern verifier states number: 114 transitions number: 378	the pattern verifier after global checking states number: 16 transitions number: 39

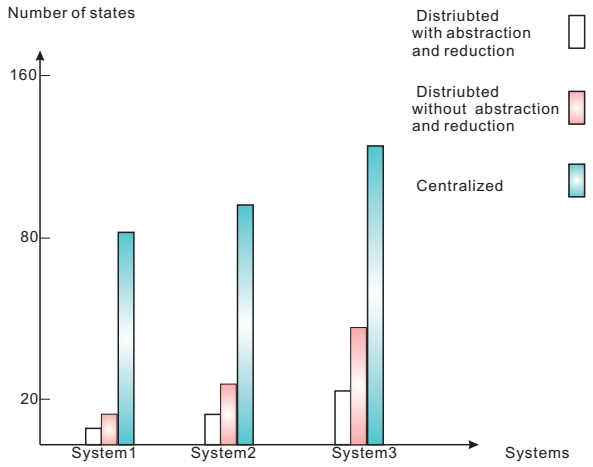

Figure 5.8: The state space of two distributed algorithms and the centralized one.

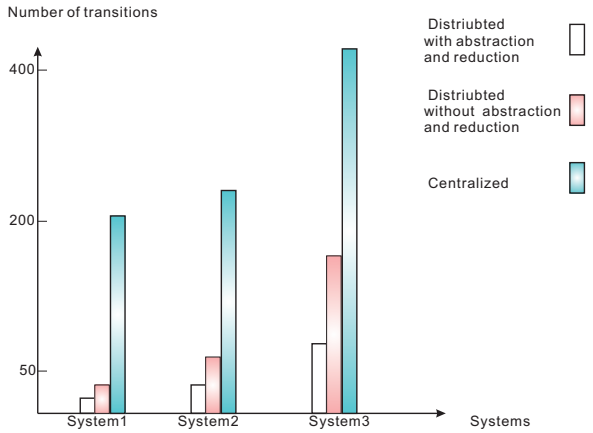

Figure 5.9: The transition space of two distributed algorithms and the centralized one.

that all components are connected with each other. In figure 5.8 and figure 5.9, the white columns represent the search space, the number of states and transitions, of our distributed algorithm with abstraction and reduction, the pink ones show the search space of the normal distributed algorithm, i.e., without abstraction and reduction, and the blue ones are the space of the centralized algorithm. We can see that the abstraction and reduction can further reduce search space during pattern diagnosability checking. The larger the system is, the more space that can be saved in our abstracted algorithm.

5.2.2 Results Discussion

In the centralized approach ([43]), all components are first synchronized to get the global model and then the pattern recognizer of the global model is constructed before calculating the non-optimized global pattern verifier as described in section 2.3.2.4. It is too expensive to apply the centralized approach to real complex distributed systems because the size of the state space of the global model risks an exponential growing with the number of system components. More precisely, in the centralized approach, for the global pattern recognizer, the maximum number of states is $(|Q_1| \times |Q_2| \times ... \times |Q_n| \times |Q_\Omega|)$, where $|Q_i|$ is the number of states in the component G_i and $|Q_\Omega|$ is the number of states in the pattern Ω, and the maximum number of transitions is $(|Q_1|^2 \times |Q_2|^2 \times ... \times |Q_n|^2 \times |Q_\Omega|^2 \times |\Sigma|)$, where $|\Sigma|$ is the number of events in the whole system. Then for the global pattern verifier, the maximum number of states is $(|Q_1|^2 \times |Q_2|^2 \times ... \times |Q_n|^2 \times |Q_\Omega|^2)$ and the maximum number of transitions is $(|Q_1|^4 \times |Q_2|^4 \times ... \times |Q_n|^4 \times |Q_\Omega|^4 \times |\Sigma|)$. Thus in the worst case, the complexity is $O(|Q_i|^{4n} \times |Q_\Omega|^4 \times |\Sigma|)$, where $|Q_i|$ is the number of states in the component whose states set has the maximum number compared to all other components in the system. Now consider our distributed method that avoids calculating global objects in the following way.

1. First consider the pattern recognition. In the distributed approach, let G^k be the k^{th} obtained subsystem. We have $G^1 = G_{i_1}$, where G_{i_1} is a component containing a significant event that changes the pattern initial state, i.e., the initial subsystem, and $G^k = R^\Omega_{G^{k-1}} \| G_{i_k}$, where $1 < k \leq n$, G_{i_k} is the k^{th} selected component for extending subsystem, and $R^\Omega_{G^{k-1}}$ is the diagnosability relative part of the $(k-1)^{th}$ pattern recognizer. As analyzed in section 4.2.2, normally we have $L(R^\Omega_{G^{k-1}}) \subset L(G^{k-1})$ and $|Q_{R^\Omega_{G^{k-1}}}| < |Q_{G^{k-1}}| \times |Q_\Omega|$, where $|Q_{R^\Omega_{G^{k-1}}}|$ is the number of states in the diagnosability relative part of the $(k-1)^{th}$ pattern recognizer and $|Q_{G^{k-1}}|$ is the number of states in the $(k-1)^{th}$ obtained subsystem. While for the centralized approach, to recognize pattern, the global model is constructed by synchronizing all components $G_1 \| ... \| G_n$. In other words, one major difference between centralized approach and distributed approach is that for the latter, beginning from the initial subsystem, each time for subsystem extension, we only synchronize the diagnosability relative part, which is actually a subpart of the current subsystem, with the next selected component. While for the centralized one, we totally synchronize all components without reduction. Suppose that in the distributed approach, the subsystem corresponding to the complete recognizer is the m^{th} obtained subsystem, denoted by G^m, then it is easy to de-

duce that normally we have $|Q_{R_{G^m}}| = |Q_{G^m}| \times |Q_\Omega| \ll |Q_1| \times |Q_2| \times ... \times |Q_n| \times |Q_\Omega|$. In other words, the subsystem corresponding to the complete recognizer is generally only a small subpart of the global model.

2. To construct pattern verifier, we synchronize the reduced left instance with the reduced right instance of the complete recognizer, which keeps all necessary and sufficient diagnosability information but makes the search space considerably smaller compared to the classical way adopted by the centralized approach, where the non-reduced left instance and non-reduced right instance are synchronized. More precisely, let $|Q_{R_c}|$ denote the number of states in the complete recognizer, $|Q_{R_c^l}|$ and $|Q_{R_c^r}|$ denote the number of states in the reduced left instance and that in the reduced right instance of the complete recognizer respectively. Normally we have $|Q_{R_c^l}| < |Q_{R_c}| < |Q_{R_G}|$ and $|Q_{R_c^r}| < |Q_{R_c}| < |Q_{R_G}|$, where $|Q_{R_G}|$ denotes the number of states in the global pattern recognizer. Thus we can get that normally the number of states in our initial pattern verifier obtained from the complete recognizer is much smaller than that in the global pattern verifier: $|Q_{R_c^l}| \times |Q_{R_c^r}| \ll |Q_{R_G}| \times |Q_{R_G}|$.

3. If the pattern concerned components do not include all components, i.e., $\{G_{i_1}, ...G_{i_m}\} \subset \{G_1, ...G_n\}$, and there are other connected components, we first construct the abstracted pattern verifier and then retain only partial critical paths every time before synchronizing with the connected abstracted local twin checker. What we are interested in is the existence of the globally consistent partial critical paths. Let $|Q_{V^a}|$ denote the number of states in the abstracted and reduced pattern verifier and let $|Q_{C_i^a}|$ denote the number of states in the abstracted local twin checker for the component G_i. Then normally we have $|Q_{C_i^a}| < |Q_{C_i}|$ and $|Q_{V^a}| < |Q_V|$, where $|Q_{C_i}|$ and $|Q_V|$ denote the number of states in the non abstracted local twin checker and in the non abstracted pattern verifier. Even in the case where the pattern is not diagnosable and all the components are connected with each other, the space that we obtain in the end is much smaller than the space of the global pattern verifier.

In this way, what we finally obtain is a reasonably small and necessary portion of the global pattern verifier, where in some cases there is even no information about those components that are completely not connected to the subsystem corresponding to the complete recognizer. The results of the test cases mentioned above are consistent with our analysis here.

We have shown that normally the complexity of the distributed pattern diagnosability algorithm is much smaller than that of the centralized one. Then what will happen in the worst case and what kind of system can be in the worst case? From the procedure of the distributed algorithm,

we can deduce that any system satisfying all the following conditions is in the worst case and for such a system, the search space of the distributed algorithm cannot be reduced compared to that of the centralized one.

- From the initial subsystem, in each pattern recognizer except the complete recognizer, every path is a diagnosability relative path. Only in this case, to incrementally recognize the pattern, the current subsystem cannot be reduced to a smaller one before synchronizing with the next selected component.
- Then in the complete recognizer, every path contains at least one cycle with final states and with at least one observable event for all involved components and also contains at least one cycle without final states but with at least one observable event for all involved components. In this case, the left instance and the right instance of the complete recognizer cannot be reduced to be smaller ones, before being synchronized to obtain the pattern verifier.
- If the subsystem, denoted by G^m, that corresponds to the complete recognizer does not contain all components in the system, then the system should furthermore satisfy the following conditions.
    1. All components outside G^m should be connected to G^m.
    2. Each path of the pattern verifier constructed from the complete recognizer should be a partial critical path. And in each path, observable events are only contained in cycles and there is only one observable event for each involved component in each cycle. In this case, the abstracted pattern verifier has the same space as the pattern verifier.
    3. In the local twin checker of each component outside of G^m, for each path, observable events are only contained in cycles and there is only one observable event in each cycle. Thus the abstracted local twin checker has the same space as its corresponding local twin checker.
    4. During the global consistency checking, each time after synchronizing the set of partial critical paths with the connected abstracted local twin checker, in the new obtained pattern verifier, each path is a partial critical path for the extended subsystem. In this case, our reduced pattern verifier containing only partial critical paths has the same space as the non reduced pattern verifier.

Only for systems satisfying all above conditions, the search space obtained from the distributed algorithm is not smaller than that obtained from the centralized one. However, the real systems in

our daily life that satisfy all these conditions are really rare. So in practice, we can say that the system in the worst case could hardly exist. In other words, our distributed framework makes real sense in search space reduction for pattern diagnosability verification. Furthermore, if the system has more observable events, then the distributed algorithm can reduce more search space due to abstraction process. And if the system has more independent components, i.e., components not connected to the subsystem where the pattern is completely recognized, the distributed algorithm can hugely reduce the search space.

However, in our distributed algorithm, when the system is not diagnosable, the returned non diagnosability information is normally different from that of the centralized one. For example, the centralized algorithm returns all observable events in global critical paths while the distributed algorithm returns the communication events in global critical paths and the information about the existence of observable events for components in the consistent ambiguous state cycles instead of the precise observable events. Thus on the one hand, information returned in centralized case is more precise in terms of observable events while in distributed case, the precise information about observable events is abstracted. On the other hand, the information returned in distributed case is more precise in terms of communication events, which are hidden in centralized case before global pattern verifier is constructed. But what is important when a pattern is found not diagnosable is to try to make it diagnosable and this is achieved by increasing observations, i.e., by adding sensors. Thus it is more important to have the information about communication events than that about observable events if we suppose that unobservable communication events can become observable after adding sensors.

Chapter 6

Distributed diagnosability for DES with autonomous components

In the previous chapters, we have the implicit assumption that each observable event in any component is globally observed, which means that there is still some global knowledge available and thus at the price of privacy. We propose here a new distributed framework for checking diagnosability of DES with autonomous components in terms of observation, where any component can only observe its own observable events and thus keeps its internal structure private.

There are several objectives of this chapter. The first one is to describe how to model systems with autonomous components and then to define communication compatibility of trajectories, which is identical to reconstructibility in trace theory. The second one is to propose the new definition of joint diagnosability for the system with autonomous components. Then we discuss about the undecidability of joint diagnosability verification with the assumption that communication events are unobservable. Based on this, we provide a new algorithm to test a sufficient condition of joint diagnosability. Afterwards, we show the decidability of joint diagnosability when communication events are observable. The third one consists in the discussion about the efficiency improvement by adopting a reasonable heuristic to choose the next component for further exploitation in the algorithm when the fault is jointly diagnosable in the system.

6.1 Preliminaries

In this section, we first describe how to model a DES with autonomous components and then give some important concepts.

Similar to chapter 3, we consider a distributed DES composed of a set of autonomous components $\{G_1, G_2,..., G_n\}$ that communicate with each other by communication events. Each component can whereas only observe its own observable events and thus can keep its internal structure private. Such a system is modeled by a set of FSMs, each one representing the local model of one component.

Definition 30 *(Local model of an autonomous component). The local model of the autonomous component G_i is a FSM, denoted by $G_i = (Q_i, \Sigma_i, \delta_i, q_i^0)$, where*

- *Q_i is the set of states;*
- *Σ_i is the set of events;*
- *$\delta_i \subseteq Q_i \times \Sigma_i \times Q_i$ is the set of transitions;*
- *q_i^0 is the initial state.*

The set of events Σ_i is partitioned into four subsets: Σ_{i_o}, the set of locally observable events, that can be observed only by its own component G_i; Σ_{i_u}, the set of unobservable normal events; Σ_{i_f}, the set of unobservable fault events; and Σ_{i_c}, the set of unobservable communication events shared by at least one other component, which are the only shared events between components. This definition is similar to definition 13. The only difference is that in the local model of an autonomous component, each observable event is locally observed, which means that it can only be observed by its own component when it occurs. While in definition 13, each observable event in any component can be observed by all components, i.e., globally observed. Figure 6.1 depicts a system example with three autonomous components: G_1 (top), G_2 (middle) and G_3 (bottom), where the events Oi denote locally observable events, the events Fi denote unobservable fault events, the events Ui denote unobservable normal events and the events Ci denote unobservable communication events.

Similar to the system described in chapter 3, for a distributed system with autonomous components, the global model of the entire system is also implicitly defined as the synchronized FSM of all component models based on their shared events, i.e., communication events. However, the global model will not be calculated in this chapter considering that the global knowledge of the whole system will not be required during our joint diagnosability analysis (see details in the next sections). And we also adopt assumption 3 that the projection of global language on each local model is observable live, i.e., there is no unobservable cycle.

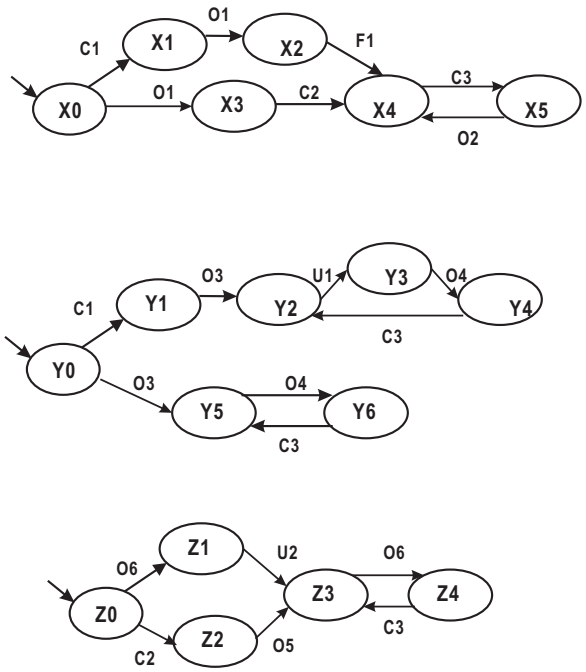

Figure 6.1: A system with three autonomous components: G_1 (top), G_2 (middle) and G_3 (bottom).

Next, we define the relative set with respect to a given component G_i, which contains all components neighboring to G_i directly or indirectly. In other words, any component in the relative set with respect to G_i either connects to G_i through common communication events or connects to G_i through some other components.

Definition 31 *(Relative set). Let G_i be an autonomous component in a system G, the G_i relative set, denoted by $\Re_{G_i}$, is the set of G_i relative components, where relative relation over the set of components is defined as follows:*

1. *For a component G_j, if G_j shares at least one event with G_i, G_j is a G_i relative component, $G_i \leftrightarrow G_j$;*

2. *The relative relation is the reflexive and transitive closure of the relation defined by point* 1*:*
   G_i is a G_i relative component, denoted by $G_i \leftrightarrow G_i$;
   given a component $G_j \neq G_i$, if $\exists G_m$ such that $G_m \neq G_i \wedge G_m \neq G_j$ and if $G_i \leftrightarrow G_m \wedge G_m \leftrightarrow G_j$, then $G_i \leftrightarrow G_j$.

Since the relation defined by point 1 of definition 31 is symmetric, with point 2, the relative relation is actually an equivalence relation.

Next we rephrase the definition of reconstructibility and quasi-reconstructibility introduced in [25] and [51] in our context. Here we denote $P_\Sigma(p)$ as the projection of trajectory p on the set of events Σ.

Definition 32 *(Reconstructibility). Given a trajectory in a subsystem G_S, after projecting on each subpart (component or subsystem) of G_S, the obtained set of trajectories is said to be reconstructible with respect to G_S.*

Definition 33 *(Quasi-reconstructibility). Given a set of trajectories in a set of components (subsystems), i.e., p_1 in G_{k_1},..., p_m in G_{k_m}, if $\forall(k_i, k_j), k_i \neq k_j$, we have $P_{\Sigma_c}(p_i) = P_{\Sigma_c}(p_j)$, where $\Sigma_c = \Sigma_{k_i} \cap \Sigma_{k_j}$, i.e., Σ_c is the set of common events between G_{k_i} and G_{k_j}, then we say that this set of trajectories $p_1, ..., p_m$ is quasi-reconstructible with respect to this corresponding set of components (subsystems), here G_{k_i} could be a component or a subsystem composed of several components.*

It has been proved that quasi-reconstructibility is a necessary but not sufficient condition of reconstructibility. In other words, a set of trajectories being reconstructible is also quasi-reconstructible, while the inverse is not necessarily true. However, if in definition 33, this set of trajectories involves only two components (subsystems), G_{k_i} and G_{k_j}, then quasi-reconstructibility is both sufficient and necessary condition of reconstructibility. Next for the sake of consistency with our framework and notations, we define in a recursive way the concept of communication compatibility, which is actually identical to reconstructibility.

Definition 34 *(Communication compatibility).*

- *Two trajectories p_1 and p_2 in different components (subsystems) are communication compatible if they are quasi-reconstructible with respect to these two components (subsystems).*
- *A set of trajectories $p_1, ..., p_n$ in different components (subsystems) are communication compatible if there exists a synchronized trajectory $p = \|_{i=1}^{n-1} p_i$ that is communication compatible with p_n.*

Lemma 10 *In a system G, given two subsystems G_S and G'_S, if $\Sigma_{S_c} \cap \Sigma_{S'_c} = \emptyset$, then $\forall(s, s\prime), s \in L(G_S), s\prime \in L(G'_S)$, s is communication compatible with $s\prime$.*

Lemma 10 means that if there is no common communication event between two subsystems, then any trajectory in one subsystem is communication compatible with any one in the other subsystem.

6.2 The distributed framework for joint diagnosability verification

Since classical diagnosability definition requires global observations, then it is not suitable for systems with autonomous components. Now we define joint diagnosability that only requires local observations without considering their global occurrence order. Then, its undecidable and decidable cases are discussed separately. With the unobservability of communication events, we prove the undecidability of joint diagnosability before giving an algorithm to test its sufficient condition. While when communication events are observable, joint diagnosability becomes decidable and thus we propose a simple algorithm to verify it.

6.2.1 Joint diagnosability

First we recall the classical diagnosability definition described in section 2.3.2. A fault f is diagnosable in a system G iff its occurrence is determinable when enough events are observed from the system after the occurrence of f. In other words, if f is diagnosable in G, then for each trajectory s^f that ends with f in G, for each t that is an extension of s^f with sufficient observable events, every trajectory p in G that is observation equivalent to $s^f.t$ should contain in it f. Here the observable events are assumed to be globally observed. The diagnosability checking consists in searching for critical pairs that witness non-diagnosability, i.e., pairs of trajectories p and $p\prime$ satisfying three conditions: 1) p contains f and $p\prime$ does not; 2) p has arbitrarily long observations after the occurrence of f; 3) $P(p) = P(p\prime)$. Unlike the case where the observable events are globally observed, autonomous components imply that no one has the global knowledge of the whole system and each component is autonomous in terms of observability. Definition 5 can be rephrased to be suitable for systems with autonomous components, which we called joint diagnosability, inspired from joint observability introduced in [68]. A fault f is jointly diagnosable in a system G iff for each trajectory s^f ending with the fault f, after any extension t with enough local observations of all components, we can be sure that f has effectively occurred. Let $P_i(p)$ denote the projection of the trajectory p to the set of locally observable events of the component G_i. We define the joint diagnosability as follows.

Definition 35 *(Joint diagnosability). A fault f is jointly diagnosable in a system G composed of a set of autonomous components $\{G_1, ...G_n\}$, iff*

$$\exists k \in N, \forall s^f \in L(G), \forall t \in L(G) \backslash s^f, (\forall i \in \{1, ..., n\}, |P_i(t)| \geq k) \Rightarrow$$
$$(\forall p \in L(G) \, (\forall i \in \{1, ..., n\}, P_i(p) = P_i(s^f.t)) \Rightarrow f \in p).$$

This definition means that for each trajectory s^f in G, for each t that is an extension of s^f with enough locally observable events in all components, every trajectory p in G that is local observation equivalent to $s^f.t$ for each component should contain in it f. In a system with autonomous components, we call a pair of trajectories p and $p\prime$ satisfying the following three conditions an **indeterminate pair**, which is similar to a critical pair in a system with global observations:

1. p contains f and $p\prime$ does not;
2. p has arbitrarily long local observations of all components after the occurrence of f;
3. $\forall i \in \{1, ..., n\}$, $P_i(p) = P_i(p\prime)$.

The main difference between a critical pair and an indeterminate pair is that for a critical pair, the two trajectories have the same sufficient global observations (the same global occurrence order), while the two trajectories of an indeterminate pair have the same sufficient local observations in each component without considering their global occurrence order.

Now we have the following fundamental theorem.

Theorem 6 *Given a system G with autonomous components, a fault f is jointly diagnosable in G iff there is no indeterminate pair in G.*

Proof :
($\Rightarrow$) Suppose that f is jointly diagnosable in a system G and there exists an indeterminate pair p and $p\prime$ with only p containing the fault f. Now let s^f denote the subpart of p that is ending with f and let t denote the rest part of p, i.e., $t = p \backslash s^f$. Since p and $p\prime$ are an indeterminate pair, from its definition, we have that p has arbitrarily long local observations for each component G_i after the occurrence of f, and that for each component G_i, p and $p\prime$ have the same local observations, i.e., $P_i(p) = P_i(p\prime)$. However, $p\prime$ does not contain f. This contradicts the definition of joint diagnosability, where any trajectory with the same enough local observations in each component as $s^f.t$ should also contain f. So f is not jointly diagnosable in G, which contradicts the assumption.
($\Leftarrow$) Now suppose that there is no indeterminate pair in G and f is not jointly diagnosable in G. From the non joint diagnosability of f, from definition 35, we know that for all $k \in N$, there exists at least one trajectory containing the fault f, denoted by $p = s^f.t$, where t has at least k observations in each component, such that there exists at least one another trajectory $p\prime$ in G without the occurrence of the fault but with the same local observations as p in each component. By choosing k greater than the maximum number of states of each component, which implies the presence of

observable cycles in p and $p\prime$ in each component, thus p and $p\prime$ can be prolonged arbitrarily long. It follows that p and $p\prime$ are an indeterminate pair since they satisfy three conditions of the definition of an indeterminate pair, which contradicts the assumption that there is no indeterminate pair.

Now we know that joint diagnosability verification consists in checking the existence of indeterminate pairs in the system. First recall that the basic idea of a local twin plant is to build a FSM that compares every pair of local trajectories to search for the pairs with the same enough local observations, but exactly one of them contains a fault, i.e., local critical pairs. For autonomous components, we still adopt local twin plant and local twin checker construction defined in section 3.2. In other words, we first construct the local pre-diagnoser for the component G_f. Then its reduced left instance is obtained by keeping the paths with at least one cycle containing fault state and then by renaming the communication events by adding the prefix L. Its reduced right instance is the one with paths containing at least one cycle without fault state and then its communication events renamed by adding the prefix R. The optimized local twin plant is constructed by synchronizing the reduced left instance and the reduced right instance based on its set of locally observable events.

Now consider our example. Figure 6.2 depicts the local pre-diagnoser D_1 for the component G_1 (top first), the reduced left instance of the local pre-diagnoser D_1^l (top second) with the reduced right instance of the local pre-diagnoser D_1^r (top third) and the local twin plant for the component G_1 (bottom). Now in a system with autonomous components, we define a path in the local twin plant that contains an ambiguous state cycle with at least one locally observable event as local critical path, which corresponds to a pair of local trajectories with the same sufficient local observations but exactly one of them contains the occurrence of the fault. In figure 6.2, the gray nodes represent ambiguous states with respect to $F1$, which form ambiguous state cycles. So the local twin plant depicted in the bottom contains an infinite number of local critical paths since they contain an infinite number of ambiguous state cycles with one locally observable event. Note that local critical paths contain original diagnosability information and can be obtained only in the local twin plant of the component G_f.

6.2.2 Diagnosability information propagation

The existence of a local critical path in the local twin plant of the component G_f does not imply that f is not jointly diagnosable because its corresponding pair of trajectories in the system is not necessarily an indeterminate pair even though they are indistinguishable in G_f. In other words,

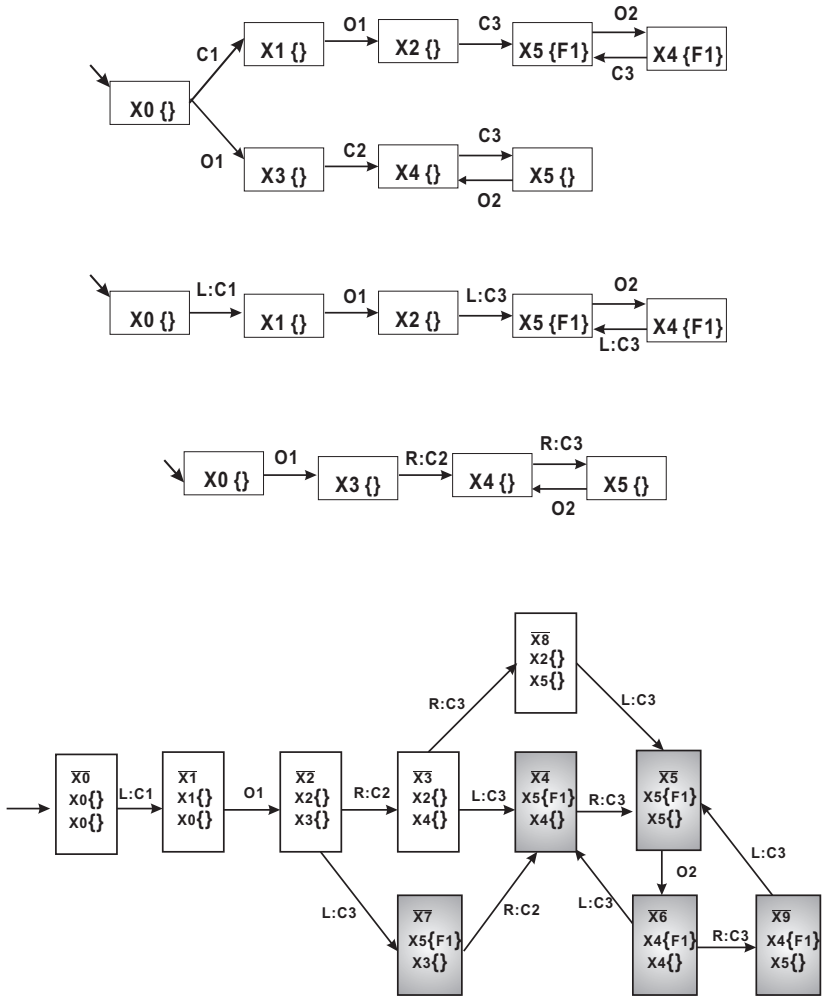

Figure 6.2: The local pre-diagnoser D_1 (top first), its left instance D_1^l (top second), its right instance D_1^r (top third) and the local twin plant (bottom) of component G_1.

there may exist another component, suppose G_i, whose cooperation can possibly distinguish this pair when the local observations of their subpart in G_i are different. However, since only the component G_f contains the fault information, the projection of any indeterminate pair on G_f must correspond to a local critical path in the local twin plant of G_f. Thus the joint diagnosability verification consists in checking the existence of local critical paths that correspond to indeterminate pairs.

Definition 34 refers to communication compatibility of trajectories. Now we define communication compatibility of paths in local twin plant or in local twin checkers. The difference is that each such path corresponds to a pair of local trajectories with the same local observations.

Definition 36 ***(Communication compatible path)*** *A set of paths in a set of local twin plant and local twin checkers are communication compatible if their corresponding set of left trajectories are communication compatible and so are the corresponding set of their right trajectories.*

For example, figure 6.3 presents a part of the local twin checkers C_2, C_3 of the components G_2, G_3, respectively. The path $(\overline{Y0} \xrightarrow{L:C1} \overline{Y1} \xrightarrow{O3} \overline{Y2} \xrightarrow{O4} \overline{Y3} \xrightarrow{L:C3} \overline{Y4} \xrightarrow{R:C3} \overline{Y2})$ of C_2 and the path $(\overline{Z0} \xrightarrow{O6} \overline{Z1} \xrightarrow{O6} \overline{Z2} \xrightarrow{L:C3} \overline{Z3} \xrightarrow{R:C3} \overline{Z4} \xrightarrow{O6} \overline{Z2})$ of C_3 depicted here are denoted by ϱ_2 and ϱ_3. The sequence of local communication events in the corresponding left trajectory of ϱ_2 in G_2 is $\{C1, C3^*\}$ and that in the left trajectory of ϱ_3 in G_3 is $\{C3^*\}$. Note that $C1$ is not contained in G_3, then from definition 33 and definition 34, these two trajectories are communication compatible. In the same way, the corresponding right trajectory of ϱ_2 in G_2 and that of ϱ_3 in G_3 are also communication compatible. From definition 33 and definition 36, ϱ_2 is communication compatible with ϱ_3.

Definition 37 ***(Compatibility of local critical path)*** *A local critical path is compatible in a subsystem G_S, where $G_f \in G_S$, if there does exist a path in the local twin checker of each component in this subsystem except G_f such that this set of paths, including this local critical path, are communication compatible.*

If ϱ_f is compatible in G_S, this set of corresponding local paths in the local twin plant and in the local twin checkers of all components in G_S that are communication compatible is called a **compatible path set** for ϱ_f in G_S. There may exist several compatible path sets for a local critical path in a given subsystem. Each path in a compatible path set has at least one cycle. If a local critical path is compatible in the whole system G, then it is said globally compatible. From definition 31 and lemma 10, it is easy to prove that a local critical path compatible in $\Re_{G_f}$ is

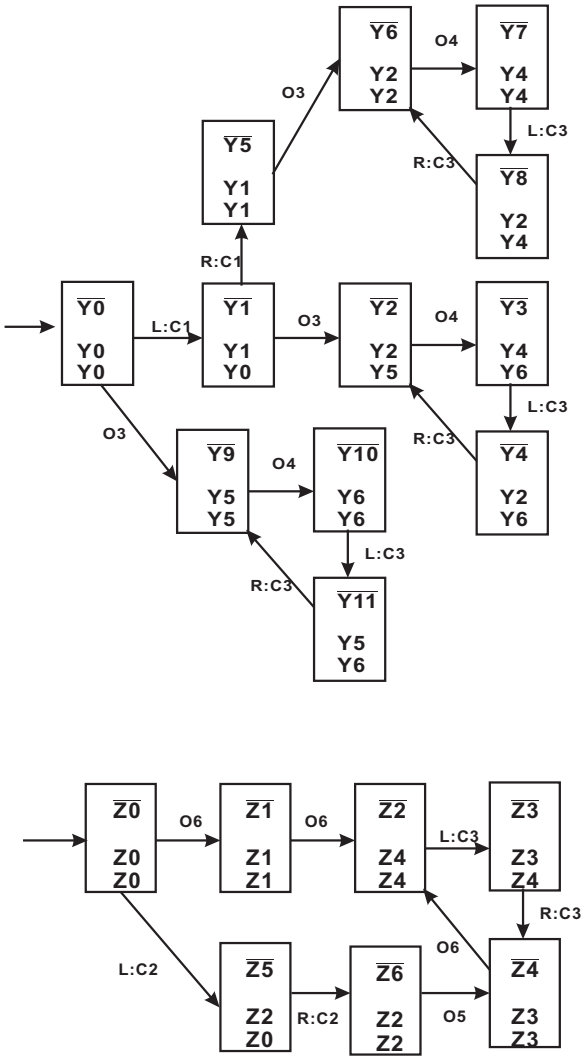

Figure 6.3: Part of local twin checker C_2 of G_2 (top) and part of local twin checker C_3 of G_3 (bottom).

globally compatible. From figure 6.2 and figure 6.3, the local critical path ($\overline{X0} \xrightarrow{L:C1} \overline{X1} \xrightarrow{O1} \overline{X2} \xrightarrow{R:C2} \overline{X3} \xrightarrow{L:C3} \overline{X4} \xrightarrow{R:C3} \overline{X5} \xrightarrow{O2} \overline{X6} \xrightarrow{L:C3} \overline{X4} \xrightarrow{R:C3} \overline{X5} \xrightarrow{O2} \overline{X6} \xrightarrow{R:C3} \overline{X9} \xrightarrow{L:C3} \overline{X5}$) in the local twin plant T_1, denoted by ϱ_f and presented in figure 6.2, is communication compatible with ϱ_2 but not with ϱ_3, where ϱ_2 and ϱ_3 are described above and shown in figure 6.3. Thus the set $\{\varrho_f, \varrho_2, \varrho_3\}$ is not a compatible path set for ϱ_f in the subsystem $\{G_1, G_2, G_3\}$. Actually here the path ϱ_f contains two elementary cycles that give birth to an infinite number of paths, due to the way these two cycles can be interleaved. In this example, the communication left (right) communication events for the two elementary cycles are the same: L:C3 (R:C3), thus we can conclude that there is no compatible path set for any of the infinite number local critical paths. Considering the case where there are embedded cycles whose elementary ones have not the same left (right) communication events, we cannot analyze communication compatibility by enumerating paths since there could be infinite number of paths with infinite number of different sequences of communication events.

Lemma 11 *In a system G with autonomous components, there exists a local critical path that is globally compatible iff there exists an indeterminate pair.*

Proof :

($\Rightarrow$) Suppose there exists a local critical path ϱ_f that is globally compatible. Since ϱ_f is globally compatible, from definition 37, there must exist at least one compatible path set for ϱ_f in the whole system. Due to communication compatibility of a compatible path set and ϱ_f being a local critical path, it can be deduced that this compatible path set, including ϱ_f, corresponds to a pair of trajectories in the whole system such that they have the same arbitrarily long local observations for each component but exactly one of them contains the fault f, which is actually an indeterminate pair. So there does exist an indeterminate pair

($\Leftarrow$) Now suppose that there exists an indeterminate pair, denoted by p and $p\prime$. The pair p and $p\prime$ being an indeterminate pair first implies that it corresponds to a local critical path in the local twin plant of G_f, denoted by ϱ_f, and then implies that $\forall i \in \{1, ...n\}$, we have $P_i(p) = P_i(p\prime)$, which forms a path in the local twin checkers of all other components except G_f. Furthermore, since p and $p\prime$ are global trajectories in the whole system, their corresponding paths in the local twin plant and in other local twin checkers must be reconstructible and thus communication compatible, which constitute a compatible path set for ϱ_f in the whole system. So from definition 37, ϱ_f is globally compatible.

Lemma 11 with its proof implies the equality between a local critical path that is globally compatible and an indeterminate pair. Then from theorem 6 and lemma 11, the major result of this chapter can be obtained as follows.

Theorem 7 *Given a system G with autonomous components, a fault f is jointly diagnosable in G iff there is no local critical path that is globally compatible.*

6.2.3 Undecidable case of joint diagnosability

From theorem 7, checking joint diagnosability is to check the existence of local critical path that is globally compatible. Next, we discuss about whether it is decidable or not. To be self-contained, first, we recall joint observability defined in [68].

Definition 38 ***(Joint observability)*** *Let L be a regular language over Σ that describes all normal behaviors and fault behaviors of a system and let K be another regular language describing the normal behaviors of the system, so we have $K \subseteq L$. Given $\Sigma_{i_o} \subseteq \Sigma, i = 1, ..., k$, then K is jointly observable with respect to L and $\Sigma_{i_1}, ..., \Sigma_{i_k}$, if the following condition holds:*

$$\forall \rho \in K, \rho\prime \in L - K, \exists i = 1, ...k, P_{\Sigma_{i_o}}(\rho) = P_{\Sigma_{i_o}}(\rho\prime)$$

Joint observability of a system means that there is no two system behaviors such that only one of them is fault behavior but they have the same observations to all k observers.

Then the undecidability of joint observability with at least two observers is proved by reducing Post's Correspondence Problem (PCP) to an observation problem. Now we briefly describe the outline of the proof with two observers [68].

- PCP: Given a finite alphabet Σ and two sets of words $v_1, v_2, ..., v_k$ and $z_1, z_2, ..., z_k$ over Σ, then a solution to PCP is a sequence of indices $(i_m)_{1 \leq m \leq n}$ with $n \geq 1$ and $1 \leq i_m \leq k$ for all m such that $v_{i_1} v_{i_2} ... v_{i_n} = z_{i_1} z_{i_2} ... z_{i_n}$.

- Now let $\Sigma\prime = \{a_1, ...a_k\}$ be a set of new letters, not in Σ. Then consider the language L over $\Sigma \cup \Sigma\prime$, defined by the regular expression: $good(v_1 a_1 + ... + v_k a_k)^+ + bad(z_1 a_1 + ... + z_k a_k)^+$, where Σ^+ denotes the set of all finite words over Σ except ϵ and all words in L that start with $good$ constitute the normal behaviors K.

- If there is a solution for the above PCP, i.e., there exist indices $i_1, ..i_n \in \{1...k\}, n \geq 1$, such that $v_{i_1}v_{i_2}...v_{i_n} = z_{i_1}z_{i_2}...z_{i_n}$, then the fault is not jointly observable. The reason is that in this case, we have a pair of words $\rho, \rho\prime$ such that $\rho = goodv_{i_1}a_{i_1}v_{i_2}a_{i_2}...v_{i_n}a_{i_n}$ and $\rho\prime = badz_{i_1}a_{i_1}z_{i_2}a_{i_2}...z_{i_n}a_{i_n}$. Thus both $\rho, \rho\prime \in L$ with $\rho \in K, \rho\prime \notin K$, and $\rho, \rho\prime$ have the same observations to both Σ and $\Sigma\prime$, which violates joint observability.

- On the other side, if the fault is not jointly observable, there is at least one pair of words violating joint observability, denoted by ρ and $\rho\prime$. Since only one of them is normal behavior, suppose ρ, then ρ must be of the form $goodv_{i_1}a_{i_1}v_{i_2}a_{i_2}...v_{i_n}a_{i_n}$ and $\rho\prime$ must be of the form $badz_{j_1}a_{j_1}z_{j_2}a_{j_2}...z_{j_m}a_{j_m}$. Furthermore, we know that ρ and $\rho\prime$ have the same observations both for Σ and $\Sigma\prime$. So we have $a_{i_1}a_{i_2}...a_{i_n} = a_{j_1}a_{j_2}...a_{j_m}$, which means that $m = n, i_1 = j_1, i_2 = j_2, ..., i_n = j_n$. And then we also get $v_{i_1}v_{i_2}...v_{i_n} = z_{i_1}z_{i_2}...z_{i_n}$, which means that there does exist a solution for the above PCP.

Now we show how to adapt the system used in the above proof to make it suitable in our framework. Consider a system composed of two components as follows, where communication events are unobservable.

- Component C_1: $(q_{10}, good, q_{11})$, where q_{10} is the initial state of C_1 and $good$ is an unobservable normal event; (q_{10}, bad, q_{12}), where bad denotes an unobservable fault event; $(q_{11}, v_1, q_{13}), ..., (q_{11}, v_k, q_{1k+2})$, where $v_i, i \in \{1...k\}$ denotes a sequence of observable events; $(q_{13}, c_1, q_{12k+3}), ..., (q_{1k+2}, c_k, q_{12k+3})$, where $c_i, i \in \{1...k\}$ is a communication event; (q_{12k+3}, c, q_{11}), where c is a communication event; $(q_{12}, z_1, q_{1k+3}), ..., (q_{12}, z_k, q_{12k+2}$), where $z_i, i \in \{1...k\}$ is a sequence of observable events; $(q_{1k+3}, c_1, q_{12k+4}), ..., (q_{12k+2}, c_k,$ $q_{12k+4})$; (q_{12k+4}, c, q_{12}).

- Component C_2: $(q_{20}, c_1, q_{21}), ...(q_{20}, c_k, q_{2k})$, where q_{20} is the initial state of C_2; $(q_{21}, a_1,$ $q_{2k+1}), ..., (q_{2k}, a_k, q_{2k+1})$, where $a_i, i \in \{1...k\}$ denotes an observable event; (q_{2k+1}, c, q_{20}).

The above system satisfies the system in the proof of [68] as well as all assumptions in our framework. From definition 35, we know that joint diagnosability is violated iff it exists two infinite trajectories, one with the fault and the other without the fault, and both give the same sufficient observations for each component (observer). So joint diagnosability checking boils down to checking the existence of an infinite sequence $i_1...i_n...$ such that $v_{i_1}...v_{i_n}.... = z_{i_1}...z_{i_n}...$, which is actually infinite PCP.

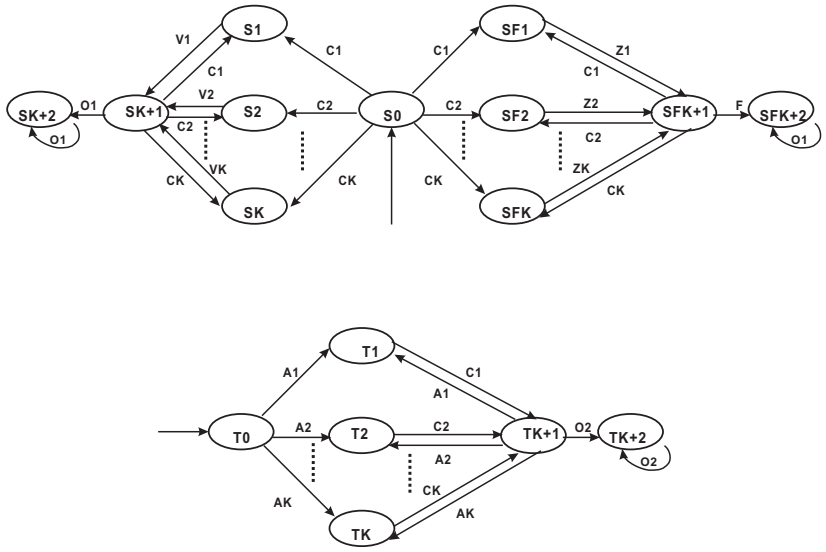

Figure 6.4: A system with two autonomous components G_1 (top) and G_2 (bottom).

For now we know that there are two major differences between joint diagnosability in our framework and joint observability in [68]. One is that the former assumes that local observers are attached to local components that are synchronized by common communication events. The other one is that joint diagnosability consists in separating infinite trajectories while joint observability is to separate finite ones. Then for joint diagnosability checking, if we adopt assumption 1, i.e., unobservability of communication events, then joint diagnosability checking boils down to infinite PCP, which is also proved to be undecidable [39].

To be more clear about undecidability of joint diagnosability with communication events being unobservable, for the sake of simplicity, we give a simple proof based on finite PCP [81].

Theorem 8 *Given a system with autonomous components where communication events are unobservable, then checking joint diagnosability is undecidable.*

Proof :

To prove this theorem, consider the example depicted in figure 6.4, where the system is composed of two components G_1 and G_2. In G_1, each one of $Vi, i \in \{1, ..., k\}$ and each one of $Zi, i \in \{1, ..., k\}$ denotes a sequence of observable events, $C1, ..., Ck$ are communication events, F denotes a fault event and $O1$ is an observable event. In G_2, each one of $Ai, i \in \{1, ..., k\}$ denotes an observable event, $C1, ..., Ck$ are communication events and $O2$ is an observable event. Then the observations in G_1 can be described as $Vi_1Vi_2...Vi_nO1^*$ without fault or $Zi_1Zi_2...Zi_nO1^*$ with

fault, where $\forall i_j, j \in \{1, ..., n\}, i_j \in \{1, ..., k\}$. In G_2, the observations are $Ai_1Ai_2...Ai_nO2^*$.
If we do not observe $O1$, we can be sure that the fault has not occurred. However, if we do observe $O1$, then the local observations are $w.O1^*$ for G_1 and $Ai_1Ai_2...Ai_nO2^*$ for G_2, where $w = Vi_1Vi_2...Vi_n$ when there is no fault or $w = Zi_1Zi_2...Zi_n$ when there is a fault. Clearly, if PCP has a solution, i.e., $\exists(i_m)_{1\leq m\leq n}$ such that $Vi_1Vi_2...Vi_n = Zi_1Zi_2...Zi_n$, we have two trajectories p and $p\prime$ such that the observations of p in G_1 are $Vi_1Vi_2...Vi_nO1^*$, which is a trajectory without fault, while the observations of $p\prime$ in G_1 are $Zi_1Zi_2...Zi_nO1^*$, which is a trajectory with a fault. And both p and $p\prime$ have the same observations for G_2, i.e., $Ai_1Ai_2...Ai_nO2^*$. Thus we get that p and $p\prime$ have the same observations for both G_1 and G_2, i.e., $Vi_1Vi_2...Vi_nO1^*$=$Zi_1Zi_2...Zi_n$ $O1^*$ for G_1 and $Ai_1Ai_2...Ai_nO2^*$ for G_2, then the fault is not jointly diagnosable.
On the other hand, if the fault is not jointly diagnosable, then we have at least one indeterminate pair, denoted by p and $p\prime$ such that the projection of p on G_1 is $Ci_1Vi_1Ci_2Vi_2...Ci_nVi_nO1^*$, on G_2 is $Ai_1Ci_1Ai_2Ci_2...Ai_nCi_nO2^*$ and that of $p\prime$ on G_1 is $Cj_1Zj_1Cj_2Zj_2...Cj_mZj_mFO1^*$ and on G_2 is $Aj_1Cj_1Aj_2Cj_2...Aj_mCj_mO2^*$. From the fact that p and $p\prime$ have the same observations for G_2, we get $Ai_1Ai_2...Ai_nO2^* = Aj_1Aj_2...Aj_mO2^*$ and thus we have $m = n$ and $i_1 = j_1, ..., i_n = j_n$. And then from the same observations of p and $p\prime$ on G_1, we get $Vi_1Vi_2...Vi_nO1^* = Zi_1Zi_2...Zi_nO1^*$, i.e., $Vi_1Vi_2...Vi_n = Zi_1Zi_2...Zi_n$, which means that there is a solution for PCP. Since PCP is undecidable, then checking joint diagnosability is also undecidable.

6.2.3.1 Algorithm to test a sufficient condition of joint diagnosability

From theorem 7, we know that joint diagnosability verification consists in checking the existence of globally compatible local critical paths, i.e., the existence of compatible path set in the whole system verifies non joint diagnosability. On the other hand, from theorem 8, we know that checking joint diagnosability with assumption of communication events being unobservable is undecidable. Next, we provide an algorithm to test a sufficient but not necessary condition of joint diagnosability. From definition 36 and definition 37, we know that to check the global compatibility of local critical paths, at least two points should be taken into account.

- The communication compatibility of left trajectories of paths in local twin plant and local twin checkers, shortly called left communication compatibility checking in the following.
- The communication compatibility of right trajectories of paths in local twin plant and local twin checkers, shortly called right communication compatibility checking in the following.

Algorithm 6 presents the procedure of our proposed algorithm to verify a sufficient condition of joint diagnosability. As shown in the pseudo-code, algorithm 6 performs as follows. Given the input as the set of component models, the fault f that may occur in the component G_f, we initialize the parameters as empty, i.e., G_S, the subsystem for the left communication compatibility checking, G'_S, the subsystem for the right communication compatibility checking. The abstracted local twin plant (ALTP) and abstracted local twin checker (ALTC) are reused here (see definition 20 and definition 21). The procedure of the algorithm can be separated by two parts: left communication compatibility checking(line 3-13) and right communication compatibility checking (15-26). We describe these two parts as follows.

- Left communication compatibility checking begins with the ALTP construction of G_f, the subsystem G_S being now G_f (line 3-4). When T_f^l is not empty and $DirectCC(G, G_S)$ is not empty (line 5), where any path of T_f^l corresponds to the set of paths in the ALTP and ALTC that are left communication compatible in the subsystem G_S, among which the path in the ALTP containing ambiguous state cycle, and $DirectCC(G, G_S)$ is the set of directly connected components to the subsystem G_S, the algorithm repeatedly performs the following steps to check left communication compatibility in an extended subsystem.

  1. Select one directed connected component G_i to the subsystem G_S, and then construct its ALTC, C_i (line 6-7).
  2. Synchronize T_f^l with C_i, based on the set of common left communication events of G_S and G_i, which is to check left communication compatibility in the subsystem composed of G_S and G_i (line 8). Since the set of synchronized events is the set of common left communication events, then the set of non-synchronized right communication events are distinguished by the prefix of component ID. For example, $(R{:}C2)$ in ALTC of G_2 is renamed as $(G_2{:}R{:}C2)$. The prefix of component ID is also useful in the next right communication compatibility checking.
  3. The subsystem is now updated by adding G_i and then in the newly synchronized FSM, we only retain paths with cycles containing observable events for all components in G_S (line 9-10).

  Then if T_f^l is empty, this means that there is no set of paths that are left communication compatible and thus non existence of local critical path that is globally compatible. So joint diagnosability information is returned. Otherwise, if T_f^l is not empty, then we proceed to

check right communication compatibility of the corresponding paths in T_f^l that are already verified to be left communication compatible in the whole system.

- Right communication compatibility checking begins with the function $AbstractRight(G_f, T_f^l)$, which is to perform delay closure with respect to the set of right communication event and the observable events of G_f, i.e., $AbstractRight(G_f, T_f^l) = \mathsf{C}_{\Sigma_d}(T_f^l)$, where $\Sigma_d = \{G_f{:}R{:}C_1,...,G_f{:}R{:}C_n, obs_f\}$, $(G_f{:}R{:}C_i)$ means any right communication event prefixed with G_f and obs_f means the observable events of G_f since in ALTP and ALTC we only have obs_i to represent observations from G_i. And then the subsystem G'_S is assigned as G_f (line 15-16). Then when T_f^r is not empty and $G_S \neq G'_S$ (line 17), where any path of T_f^r corresponds to the set of paths in the ALTP and ALTCs that are right communication compatible in the subsystem G'_S, we repeatedly perform the following steps to check their right communication compatibility in a subsystem that is extended from G'_S.

  1. Select a directed connected component to G'_S from G_S, where G_S is sufficient for this selection because any component outside G_S is not in the relative set of G_f (line 18).
  2. Perform the function $AbstractRight(G_i, T_f^l)$, which is the same as $AbstractRight(G_f, T_f^l)$ described as above except that the component is G_i instead of G_f, before synchronizing with T_f^r based on the set of common right communication events of the subsystem G'_S and G_i (line 19). Since the synchronized events are the common right communication events, before this synchronization, we rename the right communication events by removing the prefix of component ID, e.g. $(G_f{:}R{:}C1)$ renamed as $(R{:}C1)$.
  3. The subsystem G'_S is updated by adding G_i and then in the newly obtained FSM T_f^r after synchronization, we only retain paths with cycles containing observable events for all components in G'_S (line 20-21).

  If T_f^r is empty, then there is no set of paths corresponding to the set of left communication compatible paths retained in T_f^l that are right communication compatible and thus there is no globally compatible local critical path. In this case, the algorithm returns joint diagnosability information (23-24). Otherwise, if T_f^r is not empty, then we cannot determine whether the fault is jointly diagnosable or not. Then the algorithm returns the information about the indetermination of joint diagnosability (line 25-26). In other words, empty T_f^l or empty T_f^r is a sufficient condition but not necessary condition of joint diagnosability.

Lemma 12 *In algorithm 6, if T_f^l or T_f^r is empty, then the fault is jointly diagnosable, but the reverse is not true.*

Proof :
($\Rightarrow$) First suppose that T_f^l or T_f^r is empty and that the fault is not jointly diagnosable. From non joint diagnosability and from theorem 7, it follows that there exists at least one globally compatible local critical path, i.e., a compatible path set for a local critical path, this set denoted by κ. From the procedure of algorithm 6, we know that after left communication compatibility checking, since communication compatibility of the paths set κ implies both left communication compatibility and right communication compatibility (definition 36), then κ must correspond to a path both in T_f^l and in T_f^r. So the existence of globally compatible local critical paths implies that both T_f^l and T_f^r are not empty and thus the assumption is contradicted.
($\Leftarrow$) Now suppose that in system G, in the ALTP or ALTC of each component G_i, there exists two paths ρ_i^1 and ρ_i^2 such that after left communication compatibility checking, one path in T_f^l corresponds to the set of paths $\rho_1^1, ..., \rho_{n-1}^1, \rho_n^2$ in ALTP and ALTCs and another path in T_f^l corresponds to the set of paths $\rho_1^2, ..., \rho_{n-1}^2, \rho_n^1$. Then during right communication compatibility checking, after the function $AbstractRight$ for all components, we will keep the corresponding part of the paths ρ_i^1 and ρ_i^2 for all components. After right communication compatibility checking, if one path in T_f^r corresponds to the set of paths $\rho_1^1, ..., \rho_{n-1}^1, \rho_n^1$ in ALTP and ALTCs and another path in T_f^r corresponds to the set of paths $\rho_1^2, ..., \rho_{n-1}^2, \rho_n^2$, this means that both T_f^l and T_f^r are not empty but their retained paths do not necessarily correspond to the same set of paths in ALTP and ALTCs. Thus the situation becomes possible where there is no globally compatible local critical path, i.e., joint diagnosability, but both T_f^l and T_f^r are not empty. So the reverse is not necessarily true. Thus either T_f^l or T_f^r is empty is only sufficient but not necessary condition of joint diagnosability.

Now consider our example. The top first three parts of figure 6.5 are actually a part of ALTP of G_1 and that of ALTCs of G_2 and G_3 after renaming, i.e., each right communication event is prefixed with the component ID G_i. The bottom second part of this figure is a part of FSM obtained by synchronizing ALTP with ALTCs for connected components, i.e., G_2 and G_3. The gray nodes represent ambiguous states. This part represents infinite number of paths due to the two elementary ambiguous state cycles with observable events for all three components. So its corresponding paths in ALTP and ALTCs are left communication compatible. In this way, clearly, we can process embedded cycles that may produce infinite number of paths. Then this part will be retained to check right communication compatibility. Then the bottom part of figure 6.5 shows

Algorithm 6 Algorithm to Check Sufficient Condition of Joint Diagnosability

INPUT:
the system model $G = (G_1, ..., G_n)$;
f with G_f, the component where the fault f may occur
Initializations:
$G_S \leftarrow \emptyset$ (subsystem considered for left checking);
$G'_S \leftarrow \emptyset$ (subsystem considered for right checking)
$T_f^l \leftarrow ConstructALTP(G_f)$
$G_S \leftarrow G_f$
while $T_f^l \neq \emptyset$ and $DirectCC(G, G_S) \neq \emptyset$ **do**
  $G_i \leftarrow SelectDirectCC(G, G_S)$
  $C_i \leftarrow ConstructALTC(G_i)$
  $T_f^l \leftarrow Sync(T_f^l, C_i)$, where synchronized events are the set of common left communication events of current subsystem and the selected component G_i
  $G_S \leftarrow Add(G_S, G_i)$
  $T_f^l \leftarrow RetainConsisPaths(T_f^l)$
end while
if $T_f^l = \emptyset$ **then**
  return "f is jointly diagnosable in G."
else
  $T_f^r \leftarrow AbstractRight(G_f, T_f^l)$
  $G'_S \leftarrow G_f$
  while $T_f^r \neq \emptyset$ and $G_S \neq G'_S$ **do**
    $G_i \leftarrow SelectDirectCC(G_S, G'_S)$
    $T_f^r \leftarrow Sync(T_f^r, AbstractRight(G_i, T_f^l))$, where synchronized events are the set of common right communication of subsystem G'_S and the selected component G_i
    $G'_S \leftarrow Add(G'_S, G_i)$
    $T_f^r \leftarrow RetainConsisPaths(T_f^r)$
  end while
  if $T_f^r = \emptyset$ **then**
    return "f is jointly diagnosable in G."
  else
    return "Joint diagnosability cannot be determined."
  end if
end if

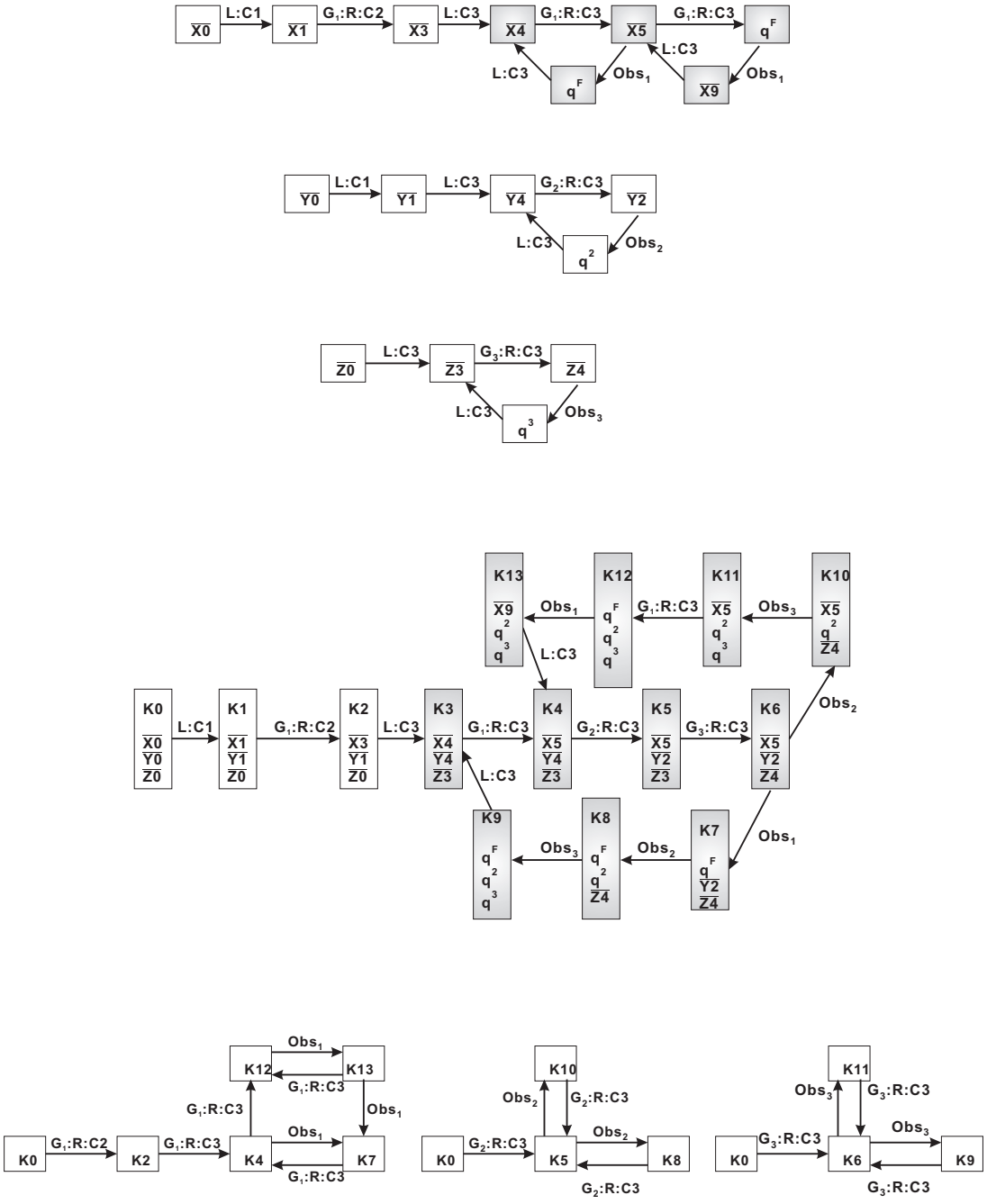

Figure 6.5: Part of ALTP for G_1 and of ALTCs for G_2 and G_3 after renaming (top first three parts), part of the synchronization based on common left communication events (bottom second) and the result of delay closure for three components G_1, G_2 and G_3 respectively (bottom).

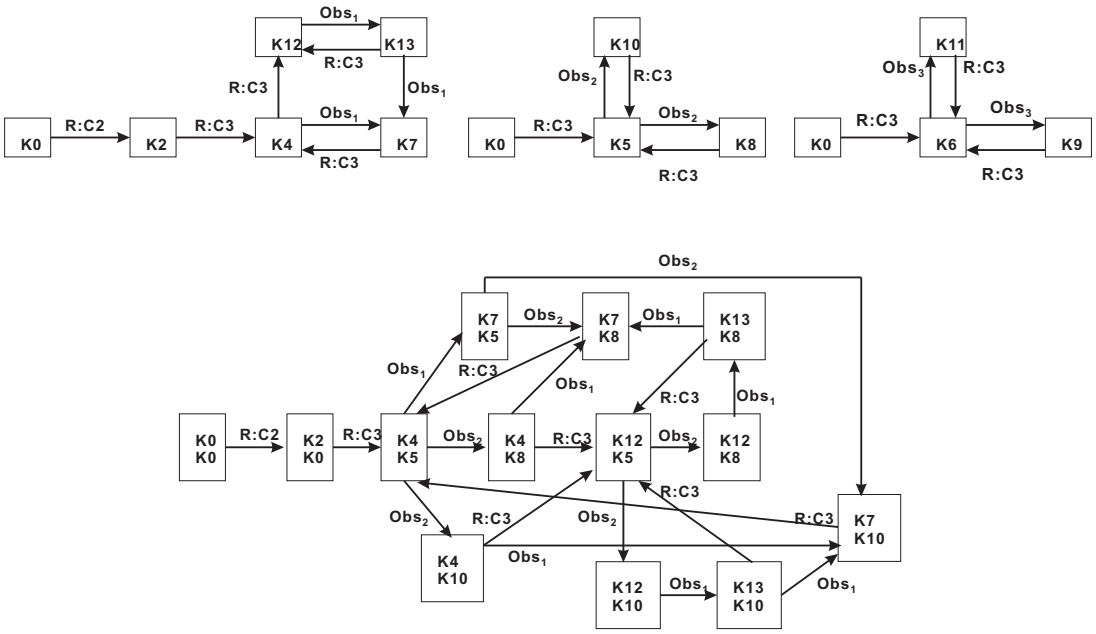

Figure 6.6: Result of delay closure for three components G_1, G_2 and G_3 respectively after renaming (top, see figure 6.5), part of synchronization of FSMs depicted in top left and top middle (bottom).

the result of performing delay closure with respect to right communication events and observable events for each component on the part depicted by the bottom second part. Then we rename the right communication events by removing the prefix of component ID, e.g. $(G_1{:}R{:}C3)$ renamed as $(R{:}C3)$. Afterwards we synchronize them based on the common right communication events. This synchronization check the right communication compatibility of the corresponding paths, where the existence of cycles with observable events for all involved components verifies their right communication compatibility. For this example, after synchronization, the bottom part of figure 6.6 shows a part of result after checking the right communication compatibility of the corresponding paths in G_1 and G_2. Then during the synchronization of this part with top right part of this figure (corresponding part in G_3), they will be blocked at the first event $(R{:}C2$ or $R{:}C3)$ since communication events $C2, C3$ are contained both in current subsystem composed of G_1 and G_2 and the component G_3. Thus we cannot get any path with cycles containing observable events for three components. In the same way, after checking all other parts of ALTP and ALTCs, there is no such path, which means there is no compatible path set for local critical path, i.e., there is no globally compatible local critical path. Thus the fault is jointly diagnosable in the system.

If f is jointly diagnosable in the system, we can improve the algorithm efficiency by searching for a subset of $\Re_{G_f}$ that is sufficient to verify joint diagnosability through an appropriate compo-

nent selection strategy. Let Σ_{S_c} be the set of communication events in the current subsystem. To choose the next component for further compatibility checking, we prefer to select the one, suppose G_i, such that $|\Sigma_{S_c} \cap \Sigma_{i_c}|$, the number of communication events in G_i contained also in the current subsystem, is maximum comparing to other components to be selected. This is a reasonable heuristic because more communication events of the selected component are involved in the current subsystem, more likely the compatible path sets for local critical paths in the current subsystem will be removed during compatibility checking for the extended subsystem. In this way, the involved components of the algorithm are as few as possible.

6.2.4 Decidable case of joint diagnosability

As shown before, undecidability of joint diagnosability is true when communication events are unobservable. If we assume their observability, then this problem becomes decidable. Next we provide a simple algorithm to check joint diagnosability with the assumption that any communication event is observable.

Algorithm 7 shows this verification procedure for joint diagnosability. Taking the system model and the faulty component as input, the parameter is initialized as empty, i.e., G_S, the current subsystem. Then the algorithm begins with the construction of ALTP of the faulty component G_f and the current subsystem being G_f (line 3-4). Here we should emphasize that when communication events are observable, then the local twin plant should be constructed by synchronizing two instances based on the set of observable events and the set of communication events due to the observability of communication events (please see definition 15 and definition 16). When both T_f and $DirectCC(G, G_S)$ are not empty (line 5), then the following steps are repeatedly performed.

- Select one component directly connected to current subsystem G_S and then construct its ALTC by first operating delay closure with respect to the set of communication events and observable events and then by synchronizing the two instances based on all events, i.e., the set of communication events and observable events. (line 6-7)

- Synchronize current ALTP T_f and the ALTC C_i based on their common communication events. (line 8)

- Then the current subsystem is updated by adding this selected component and we keep only the paths in the newly obtained FSM that contain ambiguous state cycles with observations for all involved components. (line 9-10)

During this procedure, if the ALTP for current subsystem happens to be empty, which means that there is no path that contains ambiguous state cycle with observations for all concerned components, thus there is no local critical path that is globally compatible. Otherwise, if in the end, the final FSM is not empty, then any path in it corresponding to a global compatible local critical path. The reason is that if the communication events are observable, then any path in ALTP and ALTCs corresponds to a pair of local trajectories with the same observations, including the same communication events. In other words, the separate checking for left and right communication compatibility of algorithm 6 becomes only one checking for communication compatibility in algorithm 7. While in algorithm 6, the checking into two separate phases is the reason why it is only sufficient but not necessary for joint diagnosability verification. So with the assumption of observability of communication events, the joint diagnosability checking becomes decidable, whose verification algorithm is provided here.

Algorithm 7 Algorithm for checking joint diagnosability with observability of communication events

INPUT:
the system model $G = (G_1, ..., G_n)$;
f with G_f, the component where the fault f may occur
Initializations:
$G_S \leftarrow \emptyset$ (subsystem considered for current checking)
$T_f \leftarrow ConstructALTP(G_f)$
$G_S \leftarrow G_f$
while $T_f \neq \emptyset$ and $DirectCC(G, G_S) \neq \emptyset$ **do**
  $G_i \leftarrow SelectDirectCC(G, G_S)$
  $C_i \leftarrow ConstructALTC(G_i)$
  $T_f \leftarrow Sync(T_f, C_i)$, where synchronized events are the set of common communication events of current subsystem G_S and the selected component G_i
  $G_S \leftarrow Add(G_S, G_i)$
  $T_f \leftarrow RetainConsisPaths(T_f)$
end while
if $T_f = \emptyset$ **then**
  return "f is jointly diagnosable in G"
else
  return T_f
end if

6.3 Comparison

From the previous sections, we know that joint diagnosability for systems with autonomous components is actually stronger than diagnosability for systems with global observations described in chapter 3.

Lemma 13 *Given two systems G composed of $G_1, ...G_n$ and $G\prime$ composed of $G'_1, ...G'_n$ such that for each $i \in \{1, ..., n\}$, component G_i and G'_i have the same structure except that any observable event in the component G_i can only be observed by G_i while any observable event in the component G'_i can be observed by all components of $G\prime$. In other words, G is the system with autonomous components and $G\prime$ is the one with global observations. Then we have the following result:*
if the fault f is jointly diagnosable in G, then it is diagnosable in $G\prime$.

Proof :
Suppose that the fault f is jointly diagnosable in G and that f is not diagnosable in $G\prime$. From the non diagnosability of f in $G\prime$ and $G\prime$ is a system with global observations, we know that there exists at least one global critical pair of trajectories $p1\prime$ and $p2\prime$ in $G\prime$, i.e., $p1\prime$ and $p2\prime$ satisfying three conditions: 1) only one of them contains f, suppose $p1\prime$; 2) $p1\prime$ has enough observations after the occurrence of f in all components; 3) $P(p1\prime) = P(p2\prime)$, the projection of $p1\prime$ to observable events set of $G\prime$ is the same as that of $p2\prime$, which means that they have the same observations from a global point of view. Now in the system with autonomous components G, let $p1$ and $p2$ denote the corresponding trajectories of $p1\prime$ and $p2\prime$. If we do not consider the difference that each observable event in $p1$ and $p2$ can only be observed by its own component while each observable event in $p1\prime$ and $p2\prime$ can be observed by all components, then we have $p1 = p1\prime$ and $p2 = p2\prime$. It follows that the fault f is contained in $p1$ but not in $p2$ and after the occurrence of f, $p1$ has enough local observations in each component. Furthermore, we also have $\forall k \in \{1, ..., n\}, P_k(p1) = P_k(p2)$. Clearly, $p1$ and $p2$ are an indeterminate pair and thus f is not jointly diagnosable in G, which contradicts the assumption that f is jointly diagnosable in G.

If the fault f is diagnosable in $G\prime$, it is not necessarily jointly diagnosable in G. Actually, if f is diagnosable in $G\prime$, this means that there is no critical pair in $G\prime$, however, this does not imply that there is no indeterminate pair in G. Suppose that there is an indeterminate pair $p1$ and $p2$ in G with $p1$ containing the fault. Then we have $\forall k \in \{1, ..., n\}, P_k(p1) = P_k(p2)$. Now

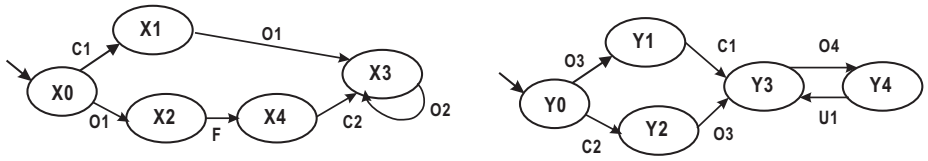

Figure 6.7: A simple system model with two components: G_1(left) and G_2(right).

in the system with global observations $G\prime$, let $p1\prime$ and $p2\prime$ denote the corresponding trajectories of $p1$ and $p2$. Then we can get $\forall k \in \{1, ..., n\}, P_k(p1\prime) = P_k(p2\prime)$, which does not mean that we can get $P(p1\prime) = P(p2\prime)$, which is however one condition of a critical pair. So there does not necessarily exist a critical pair in $G\prime$. In other words, if two trajectories in G have the same enough local observations in all components, their corresponding trajectories in $G\prime$ may have different observations from global point of view. Thus the existence of indeterminate pairs in G does not imply the existence of critical pairs in $G\prime$.

Now consider the system depicted in figure 6.1. It is a system with autonomous components. After joint diagnosability checking as described in previous sections, we verify that the fault is jointly diagnosable because there is no local critical path that is globally compatible, i.e., there does not exist indeterminate pairs. Now suppose that the system depicted in figure 6.1 is a system with global observations, i.e., any observable event in each component can be observed by all components. From lemma 13, we can deduce that the fault is diagnosable in this system. Now let us check the system diagnosability as described in chapter 3, after synchronizing the local possible critical paths with other local twin checkers, all local possible critical paths in the local twin plant of G_f are verified to be not globally consistent, which means that there is no critical pair and thus the fault is diagnosable in this system with global observations, which is consistent with lemma 13.

Figure 6.7 shows a very simple system with two components, G_1 (left) and G_2 (right). Suppose that it is a system with global observations. Then after checking diagnosability of the fault F through the method in chapter 3, we can get that F is diagnosable because the local critical path in the ALTP of component G_1 cannot survive when synchronizing with ALTC of G_2. In other words, there is no local critical path that is globally consistent. Intuitively, for any faulty trajectory, the occurrence of observable event $O1$ is before the occurrence of observable event $O3$. While for any normal trajectory, the occurrence of $O3$ is before that of $O1$. Now suppose that this system is composed of autonomous components. It is easy to find an indeterminate pair, such as the pair of trajectories $\rho = (O3, C1, O1, (O2, O4)^*)$ and $\rho\prime = (O1, F, C2, O3, (O2, O4)^*)$. Only $\rho\prime$ is

a faulty trajectory and both of them have the same sufficient observations for each component. Thus the fault is not jointly diagnosable. Here we can see that the global occurrence of observable events makes the fault F diagnosable. While since joint diagnosability does not require global occurrence of observations, so the fault is not jointly diagnosable.

6.4 Discussion

In this chapter, we suppose that a distributed system is composed of a set of components and they communicate with each other by communication events. Furthermore, in each component, the observable events can only be observed by its own component. Clearly, we do no need the monolithic model for the whole system and each component can make its own local decision based on the order of its own observable events. Thus the distributed nature of real systems is taken into account. When the communication events are assumed to be unobservable, then we prove the undecidability of joint diagnosability checking. But we still give an algorithm to test a sufficient condition of joint diagnosability. To check the non existence of indeterminate pairs in the system, we begin from local critical paths in the ALTP and then first check left communication compatibility and then check right communication compatibility. However, when the communication events are observable, checking joint diagnosability becomes decidable and then we propose a simple verification algorithm. If the fault is jointly diagnosable in the system, we can adopt heuristic strategy to select component, which may stop algorithm at first steps.

Some relative approaches are described in [56] and [71] (also see [30], [72], and [66]). In their approaches, several sites that observe a subset of observable events of the entire system are separated from the system and there is no common events between sites (the monolithic model is implicitly assumed). In addition, the authors define the notion of decentralized diagnosability, i.e., codiagnosability. A fault f is codiagnosable iff for each $s^f.t$ in a system, there exists at least one local site such that any trajectory indistinguishable from $s^f.t$ at this site contains f in it. Then its verification is based on the construction of one special structure, which is to directly show whether there exists at least one situation that violates the codiagnosability defined in the paper. Obviously, different from codiagnosability, our joint diagnosability means that for each $s^f.t$, every trajectory in the system that is local observation equivalent to $s^f.t$ for each component should contain in it f. Actually, codiagnosability requires more trajectories that should contain f since the number of the trajectories indistinguishable from $s^f.t$ at one site is normally larger than that of the trajectories indistinguishable from $s^f.t$ at each component. In other words, codi-

agnosability is stronger than joint diagnosability and from last section, the joint diagnosability is stronger than diagnosability. Furthermore, in their framework, to get the global decision, they do not need internal information of other sites. While in our case, we need to tell the communication information between components since the system under our investigation is a distributed system with autonomous components that connect with each other through communication events, i.e., common events.

Chapter 7

Conclusion

In this chapter, we recall main contributions of this thesis and then outline some directions for future work.

7.1 Thesis overview

Diagnosability analysis is crucial in system design stage, which determines whether a diagnosis algorithm can correctly and precisely make diagnosis decision given a sequence of observations issued from the system. Considering the centralized approaches are not realistic for large complex distributed systems since they require the monolithic model of the entire system. Thus we propose an abstracted approach for diagnosability analysis of distributed systems. We first define the notion of regional diagnosability for a subsystem and then the notion of diagnosable subsystem. The existence of a diagnosable subsystem verifies the diagnosability property of the whole system. Furthermore, we describe how to improve the diagnosis algorithm given a diagnosable subsystem, i.e., only the observations in the diagnosable subsystem is sufficient for diagnosis decision instead of the observations from the whole system. Then the search for a diagnosable subsystem is based on abstracted local twin plant for the component where the fault may occur and all connected abstracted local twin checkers to this abstracted local twin plant. In other words, the abstracted local twin plant contains only necessary and sufficient original diagnosability information. Then its global consistency is checked by synchronizing with connected abstracted local twin checkers that eliminate all observable information not in cycles and contain only the information about existence of observable events in each cycle for each component. We prove that these abstracted ones are sufficient to check the global consistency of original diagnosability information, i.e., the

set of local critical paths in the abstracted local twin plant. Thus the existence of local critical paths being globally consistent verifies non-diagnosability of the system.

Then we propose a distributed approach for pattern diagnosability taking into account the fact that all related investigations are centralized. We first show how to recognize the pattern by synchronizing the diagnosability relative paths with other pattern relative components to avoid global model construction. Then for pattern diagnosability checking, we adopt the abstracted method to further reduce the search space. For the sake of simplicity, we illustrate the distributed framework for simple pattern, i.e., only one sequence of significant events, and then extend simple case to general case, i.e., multiple even infinite sequences of significant events. Then we implement our algorithm to show how much state space is saved in our distributed approach compared to centralized one. The result shows that our final state space is only a quite small part of that obtained in the centralized approach.

Finally we propose a new framework for diagnosability analysis for systems with autonomous components, i.e., each component has only the access to its own observable events. We first define communication compatibility of trajectories in different components or subsystems. Then we define joint diagnosability that is stronger than classical diagnosability definition for system with normal components. In other words, to be jointly diagnosable, the observation requirement is more strict for the system with autonomous components than that with normal components to be diagnosable. Then we prove the undecidability of joint diagnosability when communication events are unobservable but still provide an algorithm to test its one sufficient condition. Then we propose another algorithm to test joint diagnosability with the assumption of observability of communication events, where the problem becomes decidable.

7.2 Future work

The directions of future work relative to this thesis are described as follows.

- When a diagnosable subsystem is returned by our approach, one future work is to investigate whether the observations in this subsystem can be reduced to make the system still diagnosable and if yes, how to reduce them ([10]). The reason is that the sensor placement is very expensive and the reduction of the number of sensors in the system can economize on its cost.

- Another direction of future work is to investigate the predictability property of distributed

systems. The property of predictability is quite meaningful since it concerns the ability of systems to predict the fault before its occurrence, which can thus be avoided ([42]). Of course, the predictability property is stronger than diagnosability property. In other words, a predictable system must be diagnosable while a diagnosable system is not necessarily predictable. In the literature, only centralized approach for predictability analysis is studied. So the investigation of distributed framework makes much sense for large complex systems that are normally distributed.

- To be more reliable, the repairability property of systems will also be studied in the future. The repairability is the ability of systems to be repaired automatically after perceiving the fault occurrence [23]. In other words, the system is repairable if it has at least one repair plan for each fault and with the diagnosis decision, it can automatically launch the corresponding repair plans.

- One very interesting and perspective direction is that for joint diagnosability checking, we can study to which level that we can relax the observability of communication events (make some of them unobservable) but still keep the problem decidable.

References

[1] A.Aghasaryan, E.Fabre, A.Benveniste, R.Boubour and C.Jard, "Fault detection and dignosis in distributed systems: An approach by partially stochastic petri nets", Journal of Discrete Event Dynamical Systems: Theory and Applications, 203-231, (1998)

[2] L. Ardissono, L. Console, A. Goy, G. Petrone, C. Picardi, M. Segnan and D. T. Dupré, "Enhancing web services with diagnostic capabilities", In Proceedings of the 3rd IEEE European Conference on Web Services, (2005).

[3] K. Autio and R.Reuter, "Structural abstraction in model-based diagnosis", In Proceedings of the 13th European Conference on Artificial Intelligence (ECAI'98), pages 269-273, Brighton, UK, (1998).

[4] P. Baroni, G. Lamperti, P. Pogliano and M. Zanella, "Diagnosis of large active systems", Artificial Intelligence 110(1):135-183, (1999).

[5] S. Bavishi and E. Chong, "Automated fault diagnosis using a discrete event systems framework", In proceedings of the 9th IEEE International Symposium on Intelligent Control, 213-218, (1994).

[6] M. Bayoudh, L. Travé-Massuyès, and X. Olive, "Hybrid systems diagnosability by abstracting faulty continuous dynamics", In Proceedings of the 17th International Workshop on Principles of Diagnosis (DX-06), Burgos, Spain, (2006).

[7] M. Bayoudh, L. Travé-Massuyès and X. Olive, "Coupling continuous and discrete event system techniques for hybrid system diagnosability analysis", In Proceedings of the 18th European Conference on Artificial Intelligence (ECAI-08), 219-223, Patras, Greece, (2008).

[8] A. Benveniste, E. Fabre, S. Haar and C. Jard, "Diagnosis of asynchronous discrete event systems, a net unfolding approach", IEEE Transactions on Automatic Control 48(5):714-727, (2003).

[9] G. Booch, J. Rumbaugh and I. Jacobson, Unified Modeling Language User Guide. Addison-Wesley Object Technology Series, (1998).

[10] L. Brandan Briones, A. Lazovik and P. Dague, "Optimizing the System Observability Level for Diagnosability", In Proceedings of the 3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, 815-830, (2008).

[11] L.K. Carvalho, J.C.Basilio and M.V.Moreira, "Robust diagnosability of discrete event systems subject to intermittent sensor failures", In Proceedings of the 10th International Workshop on Discrete Event Systems (WODES 2010), 94-99, (2010)

[12] C.G. Cassandras and S. Lafortune, Introduction To Discrete Event Systems, Second Edition. Springer, New York, (2008)

[13] L. Chittaro and R. Ranon, "Hierarchical model-based diagnosis based on structural abstraction", Advanced Engineering Informatics, 155(1-2):147-182, (2004).

[14] R. Cieslak, C. Desclaux, A. Fawaz and P. Varaiya, "Supervisory control of discrete-event processes with partial observations", IEEE Transactions on Automatic Control, 33(3):249-260, (1988).

[15] A. Cimatti, C. Pecheur and R. Cavada, "Formal verification of diagnosability via symbolic model checking", In Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03), 363-369, (2003).

[16] L. Console and O. Dressler, "Model-based diagnosis in the real world: lessons learned and challenges remaining", In Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99), (1999).

[17] L. Console, C. Picardi and M. Ribaudo, "Diagnosis and diagnosability analysis using PEPA", In Proceedings of the 14th European Conference on Artificial Intelligence (ECAI00), (2000)

[18] L. Console, C. Picardi and M. Ribaudo, "Process algebras for systems diagnosis", Artificial Intelligence 142(1):19-51, (2002).

[19] L. Console, C. Picardi and D. Theseider Dupré, "A framework for decentralized qualitative model-based diagnosis", In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07), 286-291, Hyderabad, India, (2007).

[20] O. Contant, S. Lafortune and D. Teneketzis, "Diagnosability of Discrete Event Systems with Modular Structure", Journal of Discrete Event Dynamic Systems 16: 9-37, (2006).

[21] O. Contant, S. Lafortune and D. Teneketzis, "Diagnosis of modular discrete event systems", In Proceedings of the 7th International Workshop on Discrete Event Systems (WODES-04), Reims, France, (2004).

[22] O. Contant, S. Lafortune and D. Teneketzis, "Failure diagnosis of discrete event systems: The case of intermittent faults", In Proceedings of the 41st IEEE Conference on Decision and Control (CDC-02), 4006-4011, (2002).

[23] M.-O. Cordier, Y. Pencolé, L. Travé-Massuyès and T. Vidal, "Self-healability = diagnosability + repairability", In Proceedings of the 18th International Workshop on Principles of Diagnosis (DX-07), 265-272, Nashville, TN, USA, (2007).

[24] M.-O. Cordier, L. Travé-Massuyès, and X. Pucel, "Comparing diagnosability in continuous and discrete-event systems", In Proceedings of the 17th International Workshop on Principles of Diagnosis (DX-06), 55-60, (2006).

[25] R. Cori and Y. Métivier, "Recognizable subsets of some partially abelian monoids", Theoretical Computer Science, 35:179–189, (1985)

[26] M. Daigle, X. Koutsoukos and G. Biswas, "An event-based approach to hybrid systems diagnosability", In Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), Blue Mountains, Australia, (2008).

[27] A. Darwiche and G. Provan, "Exploiting system structure in model-based diagnosis of discrete event systems", In Proceedings of the 7th International Workshop on the Principles of Diagnosis (DX-96), Val Morin, Canada, (1996).

[28] R. Debouk, S. Lafortune and D. Teneketzis, "Coordinated decentralized protocols for failure diagnosis of discrete event systems", Journal of Discrete Event Dynamical Systems: Theory and Application 10(1ÍC2):33-86, (2000).

[29] R. Debouk, R. Malik and B. Brandin, "A modular architecture for diagnosis of discrete event systems", In Proceedings of the 41st IEEE Conference on Decision and Control (CDC-02), 417ÍC422, Las Vegas, NV, USA, (2002).

[30] E. Fabre, A. Benveniste and C. Jard, "Distributed diagnosis for large discrete event dynamic systems", In Proceedings of the IFAC world congress, 237-256, (2002).

[31] E. Fabre, A. Benveniste, C. Jard, L. Ricker and M. Smith, "Distributed state reconstruction for discrete event systems. In proceedings of the 38th IEEE Control and Decision Conference (CDC-00), Sydney, (2000).

[32] P.Frank, "Fault diagnosis in dynamic systems using analytical and knowledge based redundancy", Automatica, vol.26, 459-474, (1990).

[33] S. Genc and S. Lafortune, "A distributed algorithm for on-line diagnosis of place bordered Petri nets", In proceedings of the 16th International Federation of Automatic Control World Congress, Prague, Czech Republic, (2005).

[34] S. Genc and S. Lafortune, "Diagnosis of Patterns in Partially-Observed Discrete-Event Systems", In Proceedings of the 45th IEEE Conference on Decision and Control, 422-427, San Diego, CA, USA, (2006).

[35] S.Gilmore and J.Hillston, "The PEPA workbench: A tool to support a Process Algebra based approach to performance modelling", Proceeding 7th International on Mod. Techniques and Tools for Computer Perf. Eval., volume 794 of LNCS, (1994)

[36] A. Grastien, J. R. Anbulagan and E. Kelareva, "Diagnosis of discrete-event systems using satisfiability algorithms", In proceedings of the 22th American National Conference on Artificial Intelligence (AAAI-07), (2007).

[37] A. Grastien, M.-O. Cordier and C. Largouët, "Extending decentralized discrete-event approach to diagnose reconfigurable systems", In proceedings of the 15th International Workshop on Principles of Diagnosis (DX-04), (2004).

[38] S. Haar, A. Benveniste, E. Fabre and C. Jard, "Partial order diagnosability of discrete event systems using petri nets unfoldings", In Proceedings of the 42nd IEEE Conference on Decision and Control (CDC-03), Hawaii, USA, (2003).

[39] V. Halava and T. Harju, "Undecidability of infinite post correspondence problem for instances of Size 9", Theoretical Informatics and Applications - ITA , vol. 40, no. 4, 551-557, (2006)

[40] W. Hamscher, L. Console and J. de Kleer, Readings in model-based diagnosis, Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, (1992).

[41] J. E. Hopcroft and J. D. Ullman, Introduction to Automata Theory, Languages and Computation. Addison-Wesley, (1979).

[42] T. Jéron, H. Marchand, S. Genc and S. Lafortune, "Predictability of sequence patterns in discrete event systems", In Proceedings of the 17th World Congress, (2008).

[43] T. Jéron, H. Marchand, S. Pinchinat, and M. O. Cordier, "Supervision patterns in discrete event systems diagnosis. Proceedings of the 8th International Workshop on Discrete Event Systems", 262-268, (2006).

[44] S. Jiang, Z. Huang, V. Chandra, and R. Kumar, "A polynomial time algorithm for testing diagnosability of discrete event systems", IEEE Transactions on Automatic Control, 46(8):1318-1321, (2001).

[45] J. de Kleer and B.C.Williams, "Diagnosing multiple faults", Artificial Intelligence 32(1):97ÍC130, (1987).

[46] M. Krysander, J. Aslund and M. Nyberg, "An efficient algorithm for finding minimal overconstrained subsystems for model-based diagnosis", IEEE Transactions on Systems, Man, and Cybernetics Part A, 38(1):197-206, (2008).

[47] G. Lamperti and M. Zanella, Diagnosis of active systems. Kluwer Academic Publishers, (2003).

[48] G.Lamperti and M.Zanella, "Flexible diagnosis of discrete-event systems by similarity-based reasoning techniques", Artificial Intelligence 170:232-297, (2006).

[49] S. lapp and G. Powers, "Computer-aided synthesis of fault trees", IEEE Trans. Reliability Engineering, vol.26, no. 1, pp. 2-13, April 1977.

[50] F. Lees, "Process computer alarm and disturbance analysis: Review of the state of the art", Computers and Chemical Engineering, vol. 7, no. 6, pp. 669-694, 1983.

[51] J. Messner, "Pattern matching in trace monoids", Lecture Notes in Computer Science, Volume 1200/1997, 571-582, (1997)

[52] Y. Pencolé, "Assistance for the design of a diagnosable component-based system", 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI05), 549-556, (2005).

[53] Y. Pencolé, "Diagnosability analysis of distributed discrete event systems", In Proceedings of the 16th European Conference on Articifial Intelligent (ECAI04), 43-47, (2004).

[54] Y. Pencolé and M.-O. Cordier, "A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks", Artificial Intelligence 164:121-170, (2005)

[55] X. Pucel, L. Travé-Massuyès and Y. Pencolé, "Another point of view on diagnosability", In Proceedings of the 19th International Workshop on Principles of Diagnosis (DX-08), Blue Mountains, Australia, (2008).

[56] W. Qiu and R. Kumar, "Decentralized failure diagnosis of discrete event systems", IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, (2006).

[57] R.Reiter, "A theory of diagnosis from first principles", Artificial Intelligence 32(1):57ÍC95, (1987).

[58] J.Rintanen and A.Grastien, "Diagnosability testing with satisfiability algorithms", In proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07), 532-537, (2007)

[59] L.Rozé and M.O.Cordier, "Diagnosing discrete event systems : extending the ąśdiagnoser approachąś to deal with telecommunication networks", Journal on Discrete-Event Dynamic Systems : Theory and Applications 12(1):43-81,(2002).

[60] M. Sampath, "A hybrid approach to failure diagnosis of industrial systems", In Proceedings of the 2001 American Control Conference, (2001).

[61] M.Sampath, R.Sengupta, S.Lafortune, K.Sinnamohideen and D.Teneketzis, "Diagnosability of discrete event system" IEEE Transactions on Automatic Control, pages 40(9):1555-1575, (1995)

[62] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen and D. Teneketzis, "Failure diagnosis using discrete event models", IEEE Transactions on Control Systems Technology 4(2):105-124, (1996).

[63] W.Scherer and C.White, "A survey of expert systems for equipment maintenance and diagnostics", Fault Detection and Reliability: Knowledge Based and Other Approaches, M.Singh, K.Hindi, G.Schmidt and S.Tzafestas, editors, Pergamon Press, (1987).

[64] A. Schumann and Y. Pencolé, "Scalable diagnosability checking of event-driven systems", 20th International Joint Conference on Artificial Intelligence (IJCAI-07), 575-580, (2007).

[65] P. Struss, "Fundamentals of model-based diagnosis of dynamic systems", In proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97), Nagoya, Japan, 480-485, (1997).

[66] R. Su and W.M. Wonham, "Global and local consistencies in distributed fault diagnosis for discrete-event systems", IEEE Transactions on Automatic Control 50(12):1923-1935, (2005).

[67] R. Su, W.M. Wonham, "Hierarchical distributed diagnosis under global consistency. In Proceedings of the 7th International Workshop on Discrete Event Systems (WODES-04), 157-162, Reims, France, (2004).

[68] S. Tripakis, "Undecidable Problems of Decentralized Observation and Control", In proceedings of the 40th IEEE Conference on Decision and Control, (2001).

[69] N.Viswanadham, "Control systems: Reliability", Systems and Control Encyclopedia: Theory, Technology, Applications, M.Singh, editor, Pergamon Press, (1997).

[70] N.Viswanadham and T.L.Johnson, "Fault detection and diagnosis of automated manufacturing systems", Proceeding 27th IEEE Conferencce on Decision and Control, 2301-2306 (1988).

[71] Y.Wang, T.Yoo and S.Lafortune, "Diagnosis of discrete event systems using decentralized architectures", Journal of Discrete Event Dynamic Systems, Volume 17, 233-263(31), (2007).

[72] Y. Wang, T. Yoo and S. Lafortune, "New results on decentralized diagnosis of discrete event systems", In Proceedings of the 42nd Annual Allerton Conference on Communication, Control, and Computing, (2004).

[73] A.Willsky, "A survey of design methods for failure detection in dynamic systems", Automatica, vol.12, 601-611, (1976).

[74] Y. Yan, Y. Pencolé, M.-O. Cordier and A. Grastien, "Monitoring web service networks in a model-based approach", In Proceedings of the 3rd European Conference on Web Services (ECOWS-05), (2005).

[75] L. Ye and P. Dague, "Diagnosability of patterns in distributed discrete event systems", , In proceedings of the 7th Symposium on Fault Detection, Supervision and Safety for Technical Processes (SAFEPROCESS-09), 1551-1556, Barcelone, Espagne, (2009).

[76] L. Ye, P. Dague and Y. Yan, "A distributed approach for pattern diagnosability", In Proceedings of the 20th International Workshop on Principles of Diagnosis (DX-09), 179-186, (2009).

[77] L. Ye, P. Dague and Y. Yan, "An incremental approach for pattern diagnosability in distributed discrete event systems", In proceedings of the 21st International Conference on Tools with Artificial Intelligence (ICTAI-09), 123-130, Newark, NJ, USA, (2009).

[78] L. Ye and P. Dague, "An optimized algorithm for diagnosability of component-based systems", In proceedings of the 10th International Workshop on Discrete Event Systems (WODES-10), 153-158, (2010).

[79] L. Ye and P. Dague, "Diagnosability analysis of discrete event systems with autonomous components", In proceedings of the 19th European Conference on Artificial Intelligence (ECAI-10), 105-110, (2010).

[80] T. Yoo and S. Lafortune, "Polynomial-time verification of diagnosability of partially observed discrete-event systems", IEEE Transactions on Automatic Control, 47(9):1491-1495, (2002).

[81] P. Gastin, private communication, July 2011