Towards Efficiently Diagnosing Large Scale Discrete-Event Systems

Anika Schumann

A thesis submitted for the degree of
Doctor of Philosophy at
The Australian National University

August 2007

Except where otherwise indicated, this thesis is my own original work.

Anika Schumann

Anika Schumann

August 31, 2007

Acknowledgements

I want to thank all people, who have supported this research work in Australia. My special thanks are due to my supervisors, to Dr Thiébaux for her valuable guidance throughout my thesis, to Dr Pencolé for the many interesting discussions about existing diagnosis approaches and new ideas and for the enumerative implementation of some diagnosis methods that also allowed me to compare my symbolic approaches with existing work, and to Dr Huang for pointing me to the interesting properties of jointrees and his numerous helpful comments on the presented work with jointrees. My thanks belongs also to Dr Grastien for his valuable feedback on my entire thesis.

I am also very thankful for the friendly working atmosphere at the Computer Science Laboratory of the Australian National University and for all new friends, with whom I share many nice memories.

Abstract

In this thesis we investigate the diagnosis of large discrete-event systems, where the task is to determine, on-line, all failures and states that explain a given sequence of observations. The main challenge here is to deal with the large number of possible explanations which either results in a very slow diagnosis or, in case they are compiled off-line, in huge space requirements for the diagnosis algorithms. We tackle this problem from two angles: On the one hand we present a broad spectrum of approaches differing in the amount of reasoning and compilation performed off-line and therefore in the way they resolve the tradeoff between the space occupied by the compiled information and the time taken to produce a diagnosis. This allows the use of our approach to applications with diverse time and space requirements. On the other hand we define a framework to assist a human system supervisor in reducing the number of possible diagnosis explanations by identifying the causes that make the system so poorly diagnosable and that require a respecification of the system behaviour. This asks for an extended handling of the *diagnosability* problem to not only verify whether accurate diagnostic reasoning can be performed on the system but also to provide all possible reasons of why this might not be the case.

To increase efficiency, we have defined a *symbolic* framework for our spectrum of diagnosis approaches based on binary decision diagrams. This allows for the compact representation of the compiled diagnosis information, and for its handling across many diagnosis explanations at once rather than for each explanation individually. In contrast, the efficiency of solving the extended diagnosability problem is increased by exploiting the system structure and organizing the system components into a special tree structure, known as a *jointree*.

Contents

Chapter 1

Introduction

1.1 Diagnosis - an Informal Introduction

Diagnosis is commonly regarded as the task of explaining an abnormal behaviour of a physical system. This is achieved by monitoring the system and interpreting its behaviour. It is then the responsibility of the system's supervisor to use this interpretation in order to choose appropriate actions that remove the system's abnormalities. This concept is illustrated in Figure 1.1.

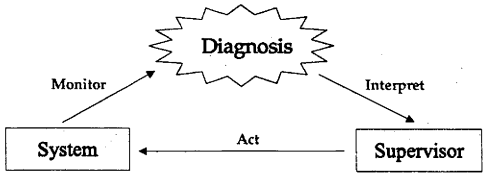


Figure 1.1: Diagnosis concept

1.1.1 Why to Diagnose

Fault detection and isolation are crucial for a wide range of applications. Several of the significant industrial disasters in the past, such as the major blackout of New York city, or the Appollo 13 incident could have well been prevented by a timely and accurate detection of a failed relay, or a burnt-out switch [Perrow, 1984]. Thus safety and reliability are two main factors motivating the diagnosis study. Moreover, avoiding undesirable effects of faults, improves the operational goals of industries, such as increased quality of performance, product integrity, and reduced cost of equipment maintenance and service.

However, given the complex and often non-apparent interactions and coupling between system components, manual fault detection is extremely difficult if not impossible. *Automated* diagnosis mechanisms are therefore needed to monitor large dynamic applications such as telecommunication networks, business processes, web services, spatial systems, software components, and power supply networks. This thesis presents several approaches that aim at increasing the efficiency of the on-line diagnosis to allow for a more timely identification of faults.

1.1.2 What to Diagnose

In order to perform diagnosis it is necessary to know, how to use the information obtained from the system to reason about the system's behaviour and its abnormalities. It is also important to determine which abnormalities are to be diagnosed and passed on to the supervisor. Diagnosis aims to disclose all possible faults. A *fault* is any abnormality of a system that requires some actions by the system's supervisor. For instance, in the context of telecommunication networks a cut cable is considered as fault while a wrongly dialled phone number is not.

Faults can be divided into *primary* and *secondary* faults. Primary faults, like a cut cable, are independent from other faults. Instead, secondary faults occur as consequence of another fault. For example, a blocked telephone might have its origin in a cut cable.

Moreover, faults are partitioned into *permanent* and *intermittent* faults. For instance, a cut cable is a permanent fault, since it cannot be repaired by the system. In contrast, a telephone that is blocked due to a cut cable might be repairable. Once the system recognises the problem it might use a reserve cable for the data transmission. Then the previously blocked telephone is functioning normally again.

One can also distinguish between *external* and *internal* faults. External faults (e.g. a cut cable) are the consequence of some event outside the system, while internal faults (e.g. a blocked telephone) have their origin within the system. Generally the system is able to detect the reason leading to an internal fault, while it cannot reason about external faults. For example, it is important to detect that a cable is cut, but not how the cable was cut.

The classification of external and internal faults relates to the distinction between a system and its environment. The system is composed of all those entities that are relevant to the diagnostic reasoning. However, a system cannot be viewed independently from its environment. For instance, the person that cuts or repairs

the cable is part of the system's environment, since the consequences of his actions have to be taken into account. However, he is not part of the system, since his actual action, that is *how* he cut the cable, is irrelevant for the diagnostic reasoning.

Finally, different types of fault information can be distinguished: *fault detection* which simply states whether the systems is faulty or not, *fault localisation* where the faulty components are determined, *fault identification* where the exact faults that have occurred are computed, and *fault propagation* where also the consequences of faults and their dependencies are considered. Clearly, the richness of the diagnosis result increases from fault detection to fault propagation. In the same way, the time and space requirements of the diagnosis methods that compute these fault types increase.

Summarising, diagnosis can be defined as the aim of detecting, localising, identifying, or propagating all primary and secondary, permanent and intermittent, external and internal faults of a system, that have any importance to the system's supervisor. These relevant faults have to be specified beforehand by a human supervisor.

1.1.3 How to Diagnose

Firstly, diagnosing faults of a system requires some understanding of the system's internal structure and its interdependencies. It is necessary to know the consequences of the faults. If a fault occurs the system usually changes its behaviour, since it is no longer able to perform all its designed functionality. Secondly, in order to reason about state changes, the system needs to be monitored or observed. This is done by equipping it with a set of sensors. The kind of sensors required depends on the faults to be diagnosed. For example, a sensor placed at a cable might provide the information, whether the cable is cut or not. By observing this sensor it is now possible to detect the fault 'cut cable'. In general, the diagnostic task can be described as follows: Given a set or a sequence of observations (sensor readings), what are the faults that have occurred? This task can be performed *on-line* or *off-line*. In on-line diagnosis, the system is assumed to be in working operation and the fault information is continuously updated with the events observed. In off-line diagnosis, the system is not in working operation and can be thought of as being in a testbed. Here the fault information is computed once and for all based on the complete sequence of events observed.

A system is said to be *diagnosable* if the occurrence of every fault can be detected with certainty after a finite number of subsequent observations. In practice, industrial systems are not diagnosable due to sensor costs and technological feasibility. However, many systems are equipped with some kind of indicators such as alarms or warning lights. By observing these indicators it is possible to determine all faults that might have occurred without knowing whether they have indeed occurred.

1.2 Approaches to Diagnosis

Due to its dramatic importance in many application domains, automated diagnosis of large scale systems has received constant and considerable attention from researchers in the fields of Artificial Intelligence and Control. The existing diagnosing methods can be classified as fault-tree based systems, expert systems and other knowledge-based, and model-based ones.

1.2.1 Fault-tree Based Methods

A fault tree [Viswanadham & Johnson, 1988] is a graphical representation of the cause-effect relationship of faults in the system. Based on an observation that indicates an abnormality of a system, the fault tree is used to reason backwards, until the root cause of the fault is found. This method is typically applied to alarm analysis in complex system. The main task here is to identify and localise the source of a fault based on simple observations like alarms. However, a fault in one component often causes a faulty behaviour in another component. This leads to a number of different alarms being emitted which makes it difficult to identify the initial fault. Moreover, the complex problem of constructing the fault tree limits its applicability in practice.

1.2.2 Expert Systems Methods

For systems with subtle and complicated interactions a diagnosis based on expert systems [Scherer & White, 1987] is often best suited. These systems are traditionally rule based. The rules are retrieved from the heuristic knowledge of an expert who relates observations to the faults that produce it. Many of these systems are based on structures and techniques relating to fault trees [Poucet *et al.*, 1987]. The drawbacks of expert system approaches are: the effort to capture the expertise

required for the diagnosis, the difficulty of validating the systems and their domain dependence.

1.2.3 Model-based Methods

It is now widely recognised that building large-scale systems is best achieved by means of model-based representation. That is, rather than relying on procedural expert knowledge, the system exploits explicit models of individual components, which it combines to automate the reasoning about system wide interactions.

In the classical theory of model-based diagnosis [de Kleer & Williams, 1987; Reiter, 1987], a diagnosis problem consists of a system description, a set of system components, and observations of the system. The system description specifies the general rules that must be followed in order for the system to function normally. Here, the system is considered to be *static*. This means that the observations are given at a single time point. A *diagnosis* is defined to be a minimal set Δ of system components such that the following two conditions are consistent with the events observed:

- all components in Δ are faulty and
- all other components are normal.

Thus it is possible that more than one diagnosis exist where none of the diagnoses is a subset of another one.

More recently, diagnostic tasks have been successfully developed for some classes of dynamic systems [Struss, 1997], which allow spontaneous state changes, either triggered by events in the environment, or resulting from the system's internal dynamics. A good deal of these research efforts has been devoted to model-based diagnosis of systems modelled as discrete-event systems (DES). DES [Cassandras & Lafortune, 1999] are dynamic systems with a discrete state space. Its behaviour is governed by the occurrence of physical events that cause abrupt changes to the state of the system. The majority of large complex systems can be modelled as DES at some level of abstraction. This discrete-change abstraction is simpler than a continuous-change one and it is still quite powerful, since for diagnostic purposes many continuous systems can be modelled as discrete using qualitative reasoning techniques [Iwasaki, 1997].

Qualitative reasoning methods enable a program to reason about the behaviour of physical systems without the kind of precise quantitative information needed

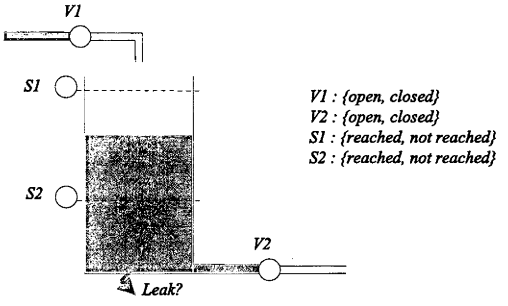


Figure 1.2: Abstract representation of a tank

by conventional analysis techniques. Figure 1.2 illustrates this concept using the example of a tank. The tank is equipped with two valves V_1 and V_2 that can be *open* or *closed* and with two sensors S_1 and S_2 that provide the information whether the water has *reached* the specified level or not. The tank contains an arbitrary amount of water and the flow speed of water passing through the open valve V_2 varies with the quantity of water in the tank. Thus this system is continuous. However, for diagnostic purposes it might only be relevant to know, whether the water level is between the two sensors or not, and whether the valves are open or not. In that case the system can be modelled as discrete-event system. The state space consists of the combination of the valve and sensor settings: $V_1 \times V_2 \times S_1 \times S_2$. Now it is no longer possible to retrieve the exact amount of water in the tank, but from a diagnostic point of view this is irrelevant. For instance, given a closed valve V_2, a leak can be diagnosed when the sensor S_2 returns *reached* followed by *not reached.*

1.3 Frameworks for Diagnosing DES

In the last few years a number of approaches have been developed for the diagnosis of DES. As stated in the previous section, these approaches are not only applicable to systems that are typically discrete like communication networks or computer systems, but also to systems that are traditionally continuous. This section first describes the different ways DES can be represented to perform diagnosis tasks. All these diagnosis approaches assume completeness. Then we give an overview of the research carried out, and we provide several classifications of diagnosis algorithms. The main challenge when dealing with large scale systems is the handling of the space/time tradeoff of diagnosis methods. The section therefore closes by presenting state of the art research aiming at solving this problem.

1.3.1 Representation Formalisms for DES

Finite State Machines

The first framework for diagnosing DES was defined in [Sampath *et al.*, 1995]. Here, the system can consists of several distinct physical components which might share certain events. A component can be thought of as the smallest replaceable unit of a system. The components are modelled as finite state machines (FSMs). The states of the FSM correspond to the component's internal state and the transitions refer to its events. Each transition represents the change of states caused by the occurrence of a single event. Events are divided into sets of *observable* and *unobservable* events. Anything that can be *observed* by the component using its sensors is modelled as *observable* event. All other events are considered *unobservable.* A subset of the latter are the *fault* events. Recalling from section 1.1.2, only those abnormal events that have any relevance to the system's supervisor are regarded as faults. Faults are assumed to be permanent. Thus a fault transition indicates only the beginning of a fault. Note that this approach assumes that it is precisely known how the system behaves if a fault occurs.

Finite state machines can also be used to model DES as set of communicating automata [Rozé & Cordier, 2002; Lamperti & Zanella, 2006]. In this setting, every component is equipped with input and output terminals to allow interaction among the components. These connections are described in terms of a *structural model*, which is represented as a graph whose nodes are components and whose edges are the connection links. In addition, a *behavioural model*, represented as FSM, is used to describe how each component reacts to incoming messages.

Process Algebra

The process algebra approach [Console, Picardi, & Ribaudo, 2002] is very similar to the previous one in that it supports the same compositional method to modelling. The DES is described with two models: One for the structure of the system, namely the enumeration of the component instances and their connections, and one for the behaviour of each component type. These models are defined by means of Performance Evaluation Process Algebra (PEPA) [Hillston, 1996], an algebraic description technique.

Petri Nets

DES can also be modelled as Petri nets, which are especially suitable for representing concurrent systems. These nets are described by *places*, *transitions* and directed *links* between them. Each place may contain several *tokens*. However, for diagnostic purposes it is sufficient to consider only places with at most one token [Aghasaryan *et al.*, 1997]. A *marking* of a Petri net, that is, an assignment of tokens to places corresponds to a system state. In order to diagnose a DES, transitions are labelled as observations and certain places are labelled as faults. The system is faulty if at least one fault place contains a token.

1.3.2 Classification of Diagnosis Algorithms

Generally, time efficiency is achieved at the expense of space efficiency and vice versa. We now describe how different diagnosis approaches resolve this time/space tradeoff. First we introduce two distinct classifications of diagnosis approaches: simulation-based and diagnoser approaches on the one hand and centralised, decentralised and distributed approaches on the other hand. Then we present a number of approaches achieving some level of both: time and space efficiency.

Simulation-based and Diagnoser based Approaches

Existing model-based methods can be classified into two categories: on-line *simulation-based* approaches compute the diagnosis from the behavioural model of the system while the latter is working; *diagnoser* approaches precompute all possible diagnoses off-line, that is while the system is not working, and retrieve, on-line, the candidate diagnosis explaining the current set of observations. Due to the size of the supervised applications, existing approaches usually suffer either from poor time performance or from space explosion. Given a system model as a set of individual components and their interactions, that is, a decentralised model, simulation-based approaches such as that of Baroni *et al.* (1999) track the possible system behaviours on-line as observations become available; the reliance on a decentralised model makes them space efficient, but the set of possible behaviours is so large that on-line computation can be time inefficient. In contrast, diagnoser based approaches such as that of Sampath *et al.* (1996) compile, off-line, a centralised system model into another finite state machine (the diagnoser) which efficiently maps observations to possible faults; here the space required by the centralised model, let alone that required by the diagnoser, constitutes a major problem.

Distributed, Decentralised, and Centralised Approaches

Diagnosis approaches can also be classified into *distributed*, *decentralised* and *centralised* methods[1]. For the latter, there is one global system model from which the diagnosis result is computed directly or indirectly [Sampath *et al.*, 1995]. In decentralised approaches such as [Pencolé & Cordier, 2005] there also exists such a global model, but this is given only indirectly as a set of components. For each of these components the local diagnosis information is computed and later combined to obtain the global diagnosis result. Due to the underlying global system model, all events emitted system wide are ordered, which allows the reasoning of global dependencies among faults. On the other hand, this makes decentralised approaches less suitable for modelling concurrent systems. For the latter, distributed methods like [Fabre, Benveniste, & Jard, 2002; Wang, Yoo, & Lafortune, 2004; Su & Wonham, 2005; Qiu & Kumar, 2006] are commonly used. Here only the observations from the same component or the same subsystem, also referred to as *site*, are ordered. Mostly, each site has its own local diagnoser associated to it. The global diagnosis, for instance, is computed by exchanging messages among these diagnosers. This differs from the decentralised approach, in which there is a centralised coordination of the local diagnosers. In comparison to centralised approaches, decentralised and distributed ones require more diagnosis time while requiring less space. In fact, due to the high space requirements of centralised methods they can hardly be applied to large scale systems.

Diagnosis Approaches Tackling the Time/Space Tradeoff

Clearly diagnosis methods considering the time/space tradeoff are needed. The authors in [Rozé & Cordier, 2002] approach it by presenting a single framework using communicating automata which can handle simulation-based and diagnoser approaches independently. However, most works in this direction aim at combining these two diagnosis methods. In the following we give a brief overview about them.

In [Debouk, Lafortune, & Teneketzis, 2000] the authors have proposed a framework consisting of a set of diagnosers, that each explain the observations from one site. The states of these diagnosers are labelled by sets of global states and fault labels, and the transitions by the events that can be observed by the site. When

[1]This is the classification adopted by the Artificial Intelligence community. In the field of Automatic Control the only distinction made is mostly the one between *centralised* (what we refer to as centralised and decentralised) and *decentralised* (what we refer to as distributed) approaches.

a site observes an event, transitions in the corresponding diagnoser are triggered. The resulting states are labelled with the diagnosis information of this site. For sites in which the event cannot be observed the local diagnosis information remains unchanged. The global diagnosis information is computed from the local diagnosis information of all sites using coordinated decentralised protocols.

The work [García *et al.*, 2005] presents another approach that computes the diagnosis information based on a set of diagnosers. Here the authors target systems that are composed of subsystems that do not interact with each other and of a complex global controller that interacts with several subsystems. The work shows how the global controller, whose events are all observable, can be decomposed to derive a set of minimum local controllers that each only interact with one subsystem. A diagnoser model is computed for each subsystem and the corresponding local controller. The global diagnosis information can straightforwardly be obtained as the Cartesian product of the corresponding local diagnoser states, since no interactions need to be considered. However, this approach is not applicable to systems in which the behaviour of one subsystem has an impact on another subsystem.

The continuous diagnosis approach introduced in [Lamperti & Zanella, 2003a] combines the classical diagnoser approach with the active systems approach. While the latter can only be used off-line, the continuous diagnosis approach enables the on-line computation of diagnosis information made of two parts. The first part is the snapshot diagnostic set, which consists of the faults that are possible after the last event has occurred. The second part is the historic diagnostic set that contains the complete diagnosis information consistent with the entire sequence of events observed. When a new event is observed, the new historic diagnostic set is computed based on the former snapshot and historic sets.

In contrast to the previous approaches, Pencolé and Cordier (2005) present a decentralised diagnosis framework that computes the complete fault propagation result and thus also returns all dependencies among faults. The diagnosis algorithm retrieves a finite state machine containing all events observed and the faults consistent with them, thereby allowing a deeper reasoning about the occurrence of faults. The approach is based on a set of subsystems whose states are labelled with a set of graphs each explaining the occurrence of one observation possible in that state. The subsystem transitions are labelled with the observable events. Once an event is observed, the on-line diagnosis approach proceeds by triggering the corresponding transition in the subsystem that emitted the event. It also keeps track of the subsystem states reached. The graphs labelling these states are synchro-

nised to account for the interactions between different components and compute the actual diagnosis information. This approach is very suitable for systems which require the diagnosis of fault dependencies. Depending on the extent to which state and transition independencies in the diagnosis result can be exploited, an efficient representation of the latter is also possible [Cordier & Grastien, 2007].

All previous approaches described in this subsection have one thing in common: they partially compile diagnosis knowledge off-line to allow for a faster on-line diagnosis. A complete compilation would require the consideration of *every* sequence of events and the representation of this compiled knowledge which is infeasible for large systems. The work of [Lamperti & Zanella, 2006] addresses this problem and performs an additional compilation on-line based on the *actual* event sequence observed. This special purpose knowledge can then be used to increase the efficiency of similar diagnosis tasks.

For performing diagnosis based on Petri nets, the authors of [Benveniste *et al.*, 2003] introduce diagnosis nets as a way to encode all solutions of a diagnosis problem. A solution is a marking of a Petri net, that is, the set of places with a token. In contrast to the diagnoser approach, the computation of the diagnosis nets is performed on-line by relying on Petri net structure only and thus is more space efficient.

Note that the space problem of diagnosis approaches can not only be solved at the expense of computation time, but also at the expense of adding inconsistent explanations to the diagnosis result. Such an approach is presented in [Su & Wonham, 2005] where the authors introduce the concepts of global and local consistency. Only if the diagnosis result is globally consistent it contains exactly the information that is consistent with the event sequence observed. Otherwise, when the space efficient local consistency check is performed, it may contain additional diagnosis candidates. Both consistency checks do not consider the computation time as this approach is entirely off-line.

We conclude this section with a comparison of the main approaches presented in this subsection. Figure 1.3 illustrates how these works relate to each other with respect to the degree of compilation performed off-line and with respect to the precision of the diagnosis result they compute (see page 3).

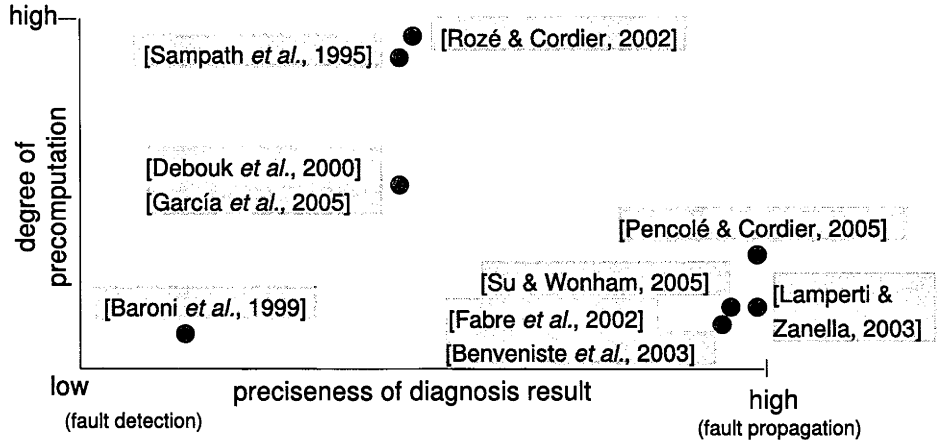


Figure 1.3: Comparison of diagnosis approaches

1.4 Diagnosability of DES

A system is diagnosable iff the occurrence of a fault guarantees that it can be detected with certainty after a finite number of subsequent observations. The diagnosability problem for DES has been introduced in [Sampath *et al.*, 1995] where the authors solve it by detecting some transition cycles of ambiguous states in a diagnoser. The main drawback of this method results from the diagnoser computation which is exponential in the number of states in the global model (determination) and as a consequence is doubly exponential in the number of components in the system. [Jiang *et al.*, 2001; Yoo & Lafortune, 2002] then propose new algorithms which are only polynomial in the number of states in G and which introduce the twin plant method. This method is elegant but impractical for large systems as the twin plant has a size quartic in the number of system states. Recent work addresses this issue by building local twin plants for system components, and synchronising them with each other until diagnosability is decided [Pencolé, 2004]. Still, in the worst case, all local twin plants need to be synchronised, again producing the global twin plant. Moreover all these approaches see the diagnosability problem as a test on a system and not as a deep analysis of the reasons why a system is not diagnosable.

1.5 Thesis Motivation & Contribution

Current model-based diagnosis approaches generally fail when confronted with large scale discrete-event systems. However, there is an increased need for automatically diagnosing such systems in many application domains. The work presented in this thesis aims at reducing the gap between existing and required diagnosis methods by tackling the efficiency problem at every step, that is

- at design time
  by improving diagnosability in order to reduce the number of diagnosis explanations that need to be considered on-line
- at compilation time
  by taking into account the requirements of the supervised system in order to choose the best algorithm among a spectrum of approaches and
- at monitoring time
  by using symbolic algorithms that speed up the on-line computations and reduce the space requirements of the diagnosis approaches.

1.5.1 Assisting in the Development of Diagnosable Systems

The gap between existing and required diagnosis methods can be reduced by making changes to the system itself to reduce the number of diagnosis explanations consistent with a sequence of observations. This requires in the first place to identify and analyse *all* causes that make the system not diagnosable. Until now the only work performed in this direction deals with the identification of a single nondiagnosable cause. This is done in the context of solving the diagnosability problem. Now, in order to assist a systems supervisor in respecifying a large scale system we

1. present an *efficient* diagnosability approach which is
2. *scalable* and which
3. computes the *minimum-cost solution* for making the whole system diagnosable.

1. Our diagnosability approach exploits the modularity of the system by organising its components into a special tree structure, known as a *jointree*, where each node of the tree is assigned a subset of the local twin plants. Once the jointree is

constructed we need only synchronise the twin plants in each jointree node, and all further computation takes the form of message passing along the edges of the jointree. The properties of the jointree guarantee that after two messages per edge, the FSMs at all nodes are collectively consistent. To further increase efficiency we also present additional techniques to reduce the number and size of the messages computed.

The question of efficiency is also raised in [Cimatti, Pecheur, & Cavada, 2003; Rintanen & Grastien, 2007; Pencolé, 2004]. The first of these works makes use of symbolic model-checking tools to test a restrictive diagnosability property. Here the global twin plant is encoded by means of binary decision diagrams. Still, for large systems even the symbolic representation of the global twin plant might not be feasible. The second approach can verify the nondiagnosability of a system using SAT, but cannot verify diagnosability. Finally, the third work shows how diagnosability can be decided without computing the global twin plant by iteratively synchronising local twin plants. This approach forms the basis of our work. However, it is only applicable to systems satisfying a restrictive property. Section 3.5 shows how this approach can be simulated with jointrees and gives a detailed comparison to our work. Rather than passing messages with bounded event sets the work presented in [Pencolé, 2004] requires the synchronisation of twin plants to check consistency, which is generally less efficient.

2. When dealing with large scale systems it is essential that our diagnosability algorithm is scalable in the sense that it is able to provide an approximate solution to the diagnosability problem whatever the computational resources are. The work presented in this thesis is the first one that considers scalability.

3. The number of problems can easily be too high to manually reason about them. We assist a human system designer by automatically deriving a characterisation of "best" system modifications to restore diagnosability. Here, it is assumed that cost estimates are available that reflect important characteristics of proposed system modifications, such as accessibility of subsystems.

Improving the diagnosability of systems is also the key motivation of work in the area of optimal sensor placement. Existing sensor placement algorithms are based on the global representation of the system model, which may not be computable for large systems. In contrast, our approach permits to determine an optimal sensor placement based only on computations carried out over subsystems. Furthermore, we improve upon previous approaches to solving the diagnosability problem by exploiting cost estimation to prune models of subsystems in order to

gain computational efficiency.

1.5.2 Diagnosis for Different Requirements of Applications

We also consider the different needs of applications with respect to their space and time requirements. Our unified framework therefore consists of a *spectrum* of approaches which differ in the degree of reasoning performed off-line and by the nature and the size of the underlying compiled models. In particular, we developed on-line diagnosis approaches based on the following models (sorted by the amount of computations performed off-line starting with no compilation):

1. component model,
2. decentralised diagnosis model,
3. global model,
4. centralised diagnosis model,
5. abstracted model,
6. nondeterministic diagnosis model, and
7. diagnoser model.

The model spectrum closest to ours is the one in [Sampath *et al.*, 1996]. It starts with a set of individual component models that are composed to obtain the global model from which the diagnoser is computed. Here models 1 and 3 are presented as computation steps. The work describes only one diagnosis approach, the one based on the diagnoser model.

Models 2 and 4-6 have not been introduced previously and all diagnosis approaches based on them are novel. Figure 1.4 illustrates how our diagnosis approaches relate to existing work. Note, that our methods cannot directly be compared to the ones presented in [Debouk, Lafortune, & Teneketzis, 2000; García *et al.*, 2005]. These authors follow a distributed approach and thus use different assumptions about the observable events and their order. In contrast our approaches are centralised (models 3-7) and decentralised (models 1-2).

Figure 1.4 shows that all our approaches compute the same diagnosis result. This allows us to conduct a fair comparison of how our different methods resolve the time/space tradeoff. The reason why we chose our diagnosis result to consist of

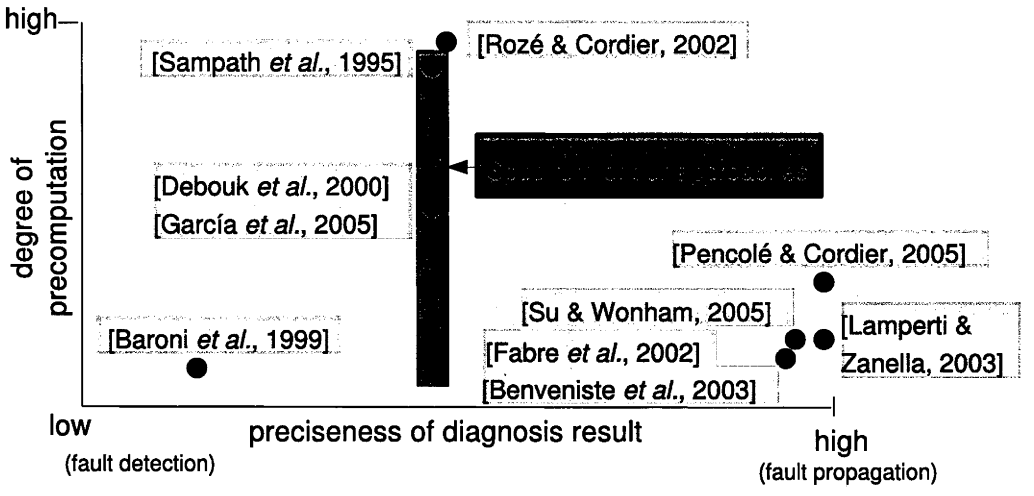


Figure 1.4: Spectrum of our diagnosis approaches contrasted to existing work.

the fault identification information was motivated by the fact that we expected for such diagnosis approaches the impact of symbolic techniques (see next subsection) to be the highest.

1.5.3 Increasing the On-line Diagnosis Efficiency

To increase efficiency, our diagnosis framework is implemented symbolically using binary decision diagrams (BDDs) [Bryant, 1986]. BDDs enable the compact encoding and the implicit manipulation of sets of states and transitions. Firstly, they allow us to reduce the space requirements of models with a high degree of compilation. Secondly, they help reducing the diagnosis time of approaches with a low degree of compilation by avoiding the individual consideration of all possible diagnosis explanations. Therefore, in our approach

- all models are represented as symbolic finite-state machines, and
- all computations are implemented via symbolic operations.

The idea of exploiting symbolic representations in the context of discrete-event systems diagnosis is not new [Cimatti, Pecheur, & Cavada, 2003; Cordier & Largouët, 2001; Sztipanovits & Misra, 1996], but it has traditionally been applied to different problems, e.g. checking diagnosability or off-line diagnosis, using off-the-shelf model-checkers. An exception is the work by Marchand and Rozé (2002) which recasts a form of diagnoser synthesis in terms of polynomial equations. In contrast to that work, we present the BDD-level encoding and computation of a

spectrum of diagnosis models ranging from component models to diagnoser models. For each of these models, we show how the diagnosis information (i.e. all faults and system states that explain a sequence of observations) can be derived on-line, by means of symbolic computations which maximise the benefits of BDDs for efficiently representing and manipulating large data sets.

1.6 Thesis Organisation

The thesis is organised as follows. Chapter 2 defines and evaluates our symbolic diagnosis approaches. Chapter 3 then presents our approach to diagnosability and shows how it can be used to assist a human system designer in respecifying the system to make it diagnosable. Finally we conclude with a summary of the main contributions of this thesis and remarks about future work in Chapter 4.

Chapter 2

A Symbolic Framework for Diagnosing Discrete-Event Systems

2.1 Introduction

For many years, automated fault diagnosis of dynamic event-driven systems has received constant and considerable attention from researchers in the fields of Artificial Intelligence and Control. Given a monitor continuously receiving observations from a system, automated diagnosis aims at identifying faults that explain the observations, and at providing an assistance to the operator in charge of the system's supervision.

In this chapter, we present a unified symbolic framework allowing for flexible and efficient diagnosis in applications with different time and space requirements. In this framework, we define and implement a spectrum of symbolic approaches which resolve the space/time complexity tradeoff in various ways. Our symbolic techniques are based on binary decision diagrams (BDDs) [Bryant, 1986] which are compact representations of Boolean functions. They enable the encoding and implicit manipulation of sets of states and transitions, without the need for explicit enumeration.

The chapter consists of two parts. First, to demonstrate the advantages of symbolic techniques for on-line diagnosis, we show how to directly compute a symbolic representation of well-known models ranging from component models to Sampath et al. diagnoser 1996, and how to retrieve the diagnosis information based on them.

Furthermore we analyse the time/space tradeoff for the main computation steps of the symbolic algorithms involved in this "direct" diagnosis approach. Our experiments on test cases derived from a telecommunication application reveal the superiority of our symbolic approaches in comparison to the enumerative ones. For instance, we obtain a symbolic diagnoser significantly smaller than the enumerative one, three orders of magnitude faster.

Based on our analysis of the direct approach, we define, in the second part, a symbolic framework that more closely exploits the advantages of BDDs in such a way that slow operations on models that hardly increase the model's size are computed off-line. In contrast to the direct approach, this "compiled" diagnosis approach speeds-up on-line diagnosis by precomputing the diagnosis information for each component individually. The resulting compiled diagnosis models are all composed of two parts: (i) a decentralised representation of the diagnosis information that is uniform across all models, and (ii) a representation of the system's behaviour that ranges from a pure decentralised description (comparable to the component models in the direct approach) to a deterministic centralised description (as e.g. the diagnoser model in the direct approach). Our experiments clearly demonstrate the superiority of the compiled symbolic approach in comparison with the direct one. For instance, the results reveal a significant speed-up of diagnosis time using our decentralised diagnosis model in place of the component-based one. They also show that one of our compiled models which is considerably smaller than the diagnoser (20 times smaller for our examples), additionally returns a symbolic diagnosis faster than the diagnoser.

This chapter is organised as follows. First we give an introduction to BDDs in Section 2.2. Section 2.3 then defines the diagnosis problem for discrete-event systems and introduces the models on which our direct diagnosis approach is based. In Section 2.4, we show how these models can be represented by BDDs, describe how we solve the diagnosis problem based on them, demonstrate their performance and analyse their space/time tradeoff. Section 2.5 presents the compiled diagnosis approach and its evaluation. Finally we review related work in Section 2.6 and summarise our contribution in Section 2.7.

2.2 Background: Binary Decision Diagrams

2.2.1 Representation of BDDs

Binary decision diagrams (BDDs) [Bryant, 1986] are compact representations of Boolean functions. They enable the encoding and implicit manipulation of sets of states and transitions, without the need for explicit enumeration. In a range of areas, such as static diagnosis, verification, controller synthesis, or AI planning, BDD-based representations have given rise to algorithms capable of exploiting the structure of the system, resulting in significant space and time gains.

BDDs are derived from Binary Decision Trees (BDTs). A BDT is a rooted, directed tree with two types of nodes: terminal and variable nodes. The terminal nodes are labelled 0 or 1 and have no outgoing edges. The variable nodes are marked with a variable v and have two outgoing edges. Figure 2.1 illustrates an example of a BDT.

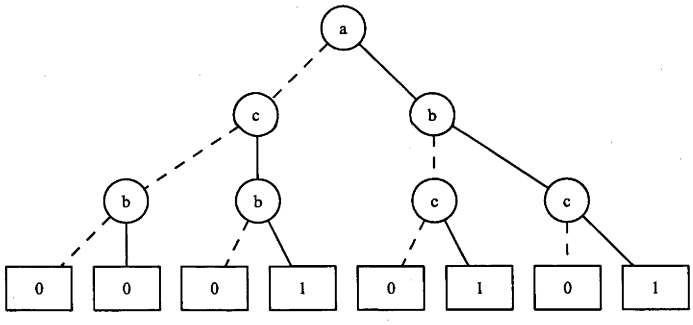


Figure 2.1: Binary Decision Tree representing the function $f = (a \vee b) \wedge c$. Dashed lines indicate the low-successor of a node and solid lines refer to the nodes' high-successor.

Every BDT represents a Boolean function f such that for every terminal node labelled with 0 the function $f(v_1, \ldots, v_n)$ returns 0 and respectively for every terminal node labelled with 1 the function returns 1. The two outgoing edges of a variable node labelled v_i point to the nodes $low(v_i)$ and $high(v_i)$: $low(v_i)$ is the root of the subtree representing the function f where v_i has been assigned the value 0, while $high(v_i)$ refers to the function in which v_i has been assigned the value 1. The BDT depicted in Figure 2.1 represents the function $f = (a \vee b) \wedge c$.

A BDD is a rooted, directed acyclic graph with variable and terminal nodes similar to a BDT. In contrast to the latter, a BDD has only one or two terminal nodes. Figure 2.2 illustrates the BDD of the same function described already by

the BDT in Figure 2.1.

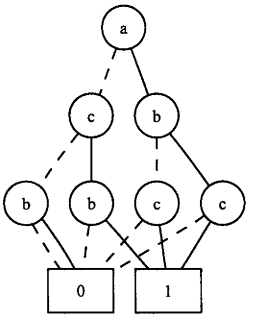


Figure 2.2: Binary Decision Diagram representing the function $f = (a \vee b) \wedge c$.

In order to express operations on Boolean functions in terms of efficient graph algorithms, the BDD needs to be reduced and ordered. This is the case if

1. on all paths of the graph the variables respect a given linear order: $v_1 < v_2 < \ldots < v_n$
2. no two distinct nodes v_1 and v_2 have the same variable name and the same low- and high-successor: $\Big(var(v_1) = var(v_2)\Big) \wedge \Big(low(v_1) = low(v_2)\Big) \wedge \Big(high(v_1) = high(v_2)\Big) \rightarrow v_1 = v_2$; and
3. no variable node has identical low- and high-successor.

Henceforth the term OBDD is used to refer to a reduced and ordered BDD. The graph depicted in Figure 2.2 is not an OBDD for three reasons: First, on the path at the most left the variable ordering is $a < c < b$ while the variables on the right most path are ordered $a < b < c$. Second, the subtree c with $low(c) = 0$ and $high(c) = 1$ is displayed twice by the two right variable nodes c. Third, the two outgoing edges of the left most variable node b point to the same node. To transform this BDD into an OBDD, the variables need to be ordered, the duplicated variable nodes removed and the nodes pointing to an identical node need to be deleted. The next two subsections explain the transformation.

2.2.2 Variable Ordering of BDDs

The shape and size of an OBDD depends significantly on the variable ordering. Figure 2.3 depicts an extreme case of how the ordering affects the size of the graph. The same function $(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$ is illustrated twice: once with the

variable ordering $a_1 < b_1 < a_2 < b_2 < a_3 < b_3$ (see left graph of Figure 2.3) and once with the ordering $a_1 < a_2 < a_3 < b_1 < b_2 < b_3$ (see right graph of Figure 2.3).

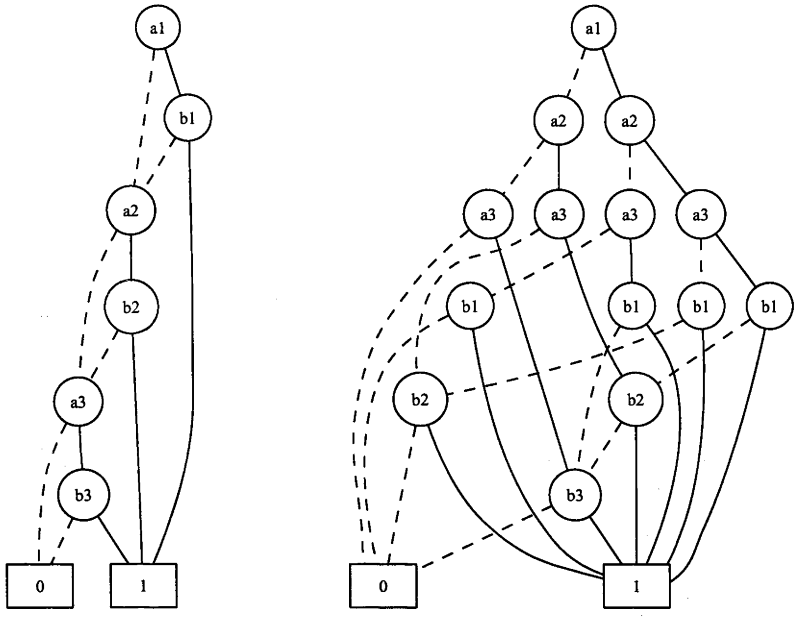


Figure 2.3: Example of variable ordering dependency

In the first case, the variables are ordered according to their occurrence in the function. From every second level in the graph only two branch destinations are required: one to the terminal node 1 and one to the next level where every disjunction up to this point yields 0. For the other case it is necessary to construct the complete binary tree for the first three levels, since for each assignment to the a variables, the function value depends in a unique way on the assignment to the b variables.

More generally, the OBDD of the function $(a_1 \wedge b_1) \vee \ldots \vee (a_n \wedge b_n)$ contains $2n + 2$ vertices if the variables follow the order $a_1 < b_1 < a_2 < b_2 < \ldots < a_n < b_n$ and 2^{n+1} vertices if the ordering is $a_1 < a_2 < \ldots < a_n < b_1 < b_2 < \ldots < b_n$. Thus choosing an appropriate ordering can dramatically decrease the size of an OBDD. In [Friedman & Supowit, 1990] the authors present an algorithm for an optimal variable ordering based on a dynamic programming approach. The optimal variable ordering for the simple BDD depicted in Figure 2.2 is: $a < b < c$. Figure 2.4 illustrates the ordered BDD[1].

However, due to its exponential run time, this algorithm can only be applied

[1] At this point the two BDDs of Figures 2.2 and 2.4 still have the same size. The next subsection shows, why the latter graph will lead to an optimal BDD.

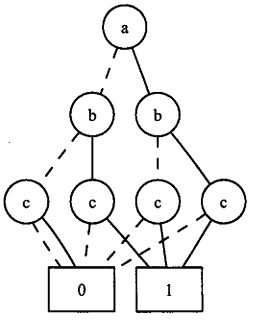


Figure 2.4: Ordered BDD representing the function $f = (a \vee b) \wedge c$.

to functions with a small number of variables. In fact, as stated in [Bryant, 1986] the problem of computing an ordering that minimises the size of the graph is itself a coNP-Complete problem. In practice the ordering is chosen either manually by a human with some understanding of the problem domain or automatically by a program using some heuristics.

2.2.3 Reduction of ordered BDDs

The reduction of an ordered BDD requires on the one hand the removal of duplicated variable nodes and on the other hand the deletion of nodes with identical successors.

A duplicated variable node v_1 is deleted following the *merging rule*. All arcs leading to v_1 are redirected to the identical node. The left graph of Figure 2.5 illustrates the BDD for the function $f = (a \vee b) \wedge c$ after all identical nodes have been removed.

A node v_2 for which both outgoing edges lead to the same node can be deleted by applying the *deletion rule*. All incoming edges to v_2 are redirected to the common successor node. Figure 2.5 presents the OBDD of the function $f = (a \vee b) \wedge c$.

The two reduction rules are sufficient to obtain the canonical representation for each function and each variable ordering [Bryant, 1986]. Applying these rules levelwise bottom-up, leads to an efficient reduction of BDDs. The authors in [Sieling & Wegener, 1993] present a reduction algorithm with linear run time $\mathbb{O}(|G|)$, where $|G|$ denotes the number of nodes in the BDD G. Note that it is possible if not necessary to reduce a graph during its construction. Thus the computation of large unreduced BDDs can be avoided.

Constructing OBDDs instead of BDDs leads to the canonical representation of

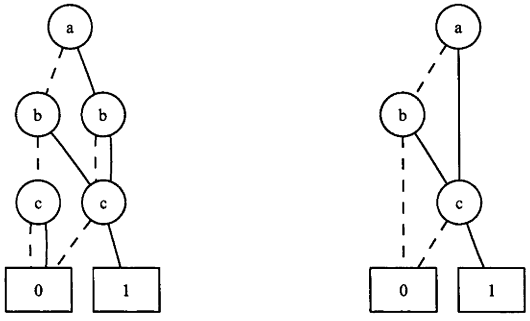


Figure 2.5: Ordered BDD without duplicated variable nodes representing the function $f = (a \vee b) \wedge c$ (left) and reduced and ordered BDD representing the function $f = (a \vee b) \wedge c$ (right).

a Boolean function. Since identical subexpressions in an OBDD are represented only once, an OBDD can be exponentially more compact than its corresponding truth table representation. For instance, the OBDD for the constants 0 and 1 respectively consists of exactly one terminal node. The uniqueness property implies the possibility of testing in constant time whether an OBDD represents a tautological function in which case the OBDD consists of the single terminal node 1 or whether the represented function is satisfiable in which case the OBDD consists of any structure other than a single terminal node 0. In contrast, these problems are NP-complete for Boolean expressions. In the remainder of this chapter we will use BDDs to denote ordered BDDs.

2.2.4 Application of BDDs

BDDs are useful in compactly representing finite state machines (FSMs). To encode state and event sets it is necessary to introduce $Nr(Q) = \lceil \log_2 |Q| \rceil$ Boolean variables for each set Q. Thus the events labelling the transitions can be encoded with the Boolean variables $b^{\Sigma} = \{b_1^{\Sigma}, \ldots, b_{Nr(\Sigma)}^{\Sigma}\}$ and the states with the variables $b^{X} = \{b_1^{X}, \ldots, b_{Nr(X)}^{X}\}$. A state of the FSM is then simply given by a Boolean function (represented by a BDD) over these state variables. For instance, in a 6 state FSM, the state x_2 would be given by the conjunction $\neg b_3^X \wedge b_2^X \wedge \neg b_1^X$, and the set of states $\{x_2, x_5\}$ by the DNF $(\neg b_3^X \wedge b_2^X \wedge \neg b_1^X) \vee (b_3^X \wedge \neg b_2^X \wedge b_1^X)$.

Transitions require the introduction of another set of state variables $b^{X'} = \{b_1^{X'}, \ldots, b_{Nr(X)}^{X'}\}$, called the primed variables, which are used to represent the target states of the transitions. Each transition can then be given as a conjunction involving the state variables, event variables, and primed variables. For instance,

$BOOL\ IsDef(bdd)$

- Returns *true* if *bdd* does not represent $false$

$BDD\ GetConj(bdd)$

- Returns a single arbitrary disjunct (a conjunction of literals) of the DNF represented by *bdd*, for example:

  $bdd \leftarrow (b_1 \wedge b_2) \vee (\neg b_1 \wedge \neg b_2)$

  $GetConj(bdd) = (b_1 \wedge b_2)$

$BDD\ AbstractVar(bdd, \{b_1, \ldots, b_m\})$

- Deletes all occurrences of the Boolean variables $\{b_1, \ldots, b_m\}$ from *bdd*, for example:

  $bdd \leftarrow (b_1 \wedge b_2 \wedge b_3) \vee (\neg b_1 \wedge \neg b_2 \wedge \neg b_3)$

  $AbstractVar(bdd, \{b_3\}) = (b_1 \wedge b_2) \vee (\neg b_1 \wedge \neg b_2)$

$BDD\ ExtractVar(bdd, \{b_1, \ldots, b_m\})$

- Deletes all occurrences of the Boolean variables that are NOT $\{b_1, \ldots, b_m\}$ from *bdd*, for example:

  $bdd \leftarrow (b_1 \wedge b_2 \wedge b_3) \vee (\neg b_1 \wedge \neg b_2 \wedge \neg b_3)$

  $ExtractVar(bdd, \{b_1, b_2\}) = (b_1 \wedge b_2) \vee (\neg b_1 \wedge \neg b_2)$

$BDD\ SwapVar(bdd, \{a_1, \ldots, a_n\}, \{b_1, \ldots, b_m\})$

- Swaps the Boolean variables $(a_1, b_1), \ldots, (a_n, b_m)$ in *bdd*, for example:

  $bdd \leftarrow (b_1 \wedge \neg b_2 \wedge b_3) \vee (\neg b_1 \wedge b_2 \wedge \neg b_3)$

  $SwapVar(bdd, \{b_1\}, \{b_2\}) = (b_2 \wedge \neg b_1 \wedge b_3) \vee (\neg b_2 \wedge b_1 \wedge \neg b_3)$

Table 2.1: Used BDD operations

in a FSM consisting of 6 states and 3 events, the transition $t = x_2 \xrightarrow{\sigma_1} x_5$ can be encoded as $t = (\neg b_3^X \wedge b_2^X \wedge \neg b_1^X) \wedge (\neg b_2^\Sigma \wedge b_1^\Sigma) \wedge (b_3^{X'} \wedge \neg b_2^{X'} \wedge b_1^{X'})$. The transition relation, i.e, a set of transitions T, can be given as a DNF which the BDD data structure will hopefully greatly reduce.

When using the compact representation and efficient manipulation of BDDs in the context of our direct diagnosis methods, our representation of the component models will essentially follow the usual symbolic FSM representation described above, while all other models will be derived from these component models via symbolic computations. This will be detailed in the next section. To increase the readability of our symbolic algorithms, we introduce some basic BDD operations shown in Table 2.1. Note that the table contains two similar functions: $AbstractVar$ and $ExtractVar$. Let bdd denote a BDD defined over the set of variables B, and let V be a subset of B. The following equivalence holds: $AbstractVar(bdd, V) = ExtractVar(bdd, B \setminus V)$.

Algorithm 1 illustrates the use of most of these BDD operations to compute all states X_{reach} that are reachable from a state set X via the transitions in T. Note that we give identical names to sets and to the corresponding Boolean functions, e.g. X. This should not cause confusion.

As described above, X is defined over the Boolean variables b^X and T is defined over the variables $b^\Sigma \cup b^X \cup b^{X'}$. The reachable state set is computed using breadth first search. Initially X_{reach} is set to false and the set of states X_{new} from which transitions still need to be triggered is set to the start states X (lines 2-3).

Algorithm 1 $CompReach(b^\Sigma, b^X, b^{X'}, X, T)$

1: INPUT: state set X, transition set T and the Boolean variables over which they are defined

Initialise

2: $X_{reach} \leftarrow false$
3: $X_{new} \leftarrow X$
4: **while** there are new states (that is as long as $IsDef(X_{new})$) **do**
5: $\quad T_{new} \leftarrow X_{new} \wedge T$
6: $\quad X_{targ} \leftarrow ExtractVar(T_{new}, b^{X'})$
7: $\quad X_{targ} \leftarrow SwapVar(X_{targ}, b^X, b^{X'})$
8: $\quad X_{new} \leftarrow X_{targ} \wedge \neg X_{reach}$
9: $\quad X_{reach} \leftarrow X_{reach} \vee X_{new}$
10: **end while**
11: OUTPUT: all states X_{reach} reachable from states in X by triggering transitions in T

Until a fixed point is reached, all transitions T_{new} starting in X_{new} are triggered (operator $\wedge$)(line 5) and the targets states that have not yet been encountered are added to X_{reach} (operator $\vee$)(line 9). To obtain the target states X_{targ} of the transitions T_{new}, we abstract the latter from its start states and events using function $ExtractVar$ (line 6). Originally the targets X_{targ} are defined over the variables $b^{X'}$. In order to compute the transitions starting in states of X_{targ} in the next loop iteration, we swap the state variables to represent X_{targ} over the variables b^X (line 7). Finally, to guarantee the termination of the algorithm, we only consider those targets from which transitions have not yet been triggered. Hence we subtract all previously encountered states X_{reach} from X_{targ} (operator $\wedge\neg$)(line 8).

Note that all transitions starting from a set of states can be computed at once (line 5). In fact, the whole procedure does not require the consideration of individual states or transitions. It is this property of BDDs that we aim to exploit in the context of on-line diagnosis.

2.3 Background: Diagnosis Problem and Direct Diagnosis Models

This section describes how we model the systems to be diagnosed. Then we define the diagnosis problem and a spectrum of models, the direct diagnosis models, that can all be used to solve this problem. To illustrate our concepts we start by introducing an example application which we will use throughout this chapter.

2.3.1 Example Application

Our example is derived from a telecommunication network [Rozé & Cordier, 2002]. A supervision centre is in charge of continuously monitoring the system; it receives a flow of alarms and analyses them on-line to identify possible faults (see Figure 2.6). The supervised system is composed of two control stations (CS1 and CS2) and one switch (SW). The switch is used to route data through the network. The purpose of the control stations is to manage the switch by reconfiguring it or reinitialising it. Only one control station manages the switch at a given time, the other is for replacement in case the station in charge fails to work.

A fault *SWfail* can occur in the switch. In that case both control stations are notified (event *NotifySWfail*) and the alarm *SWobs* is observed. If any of the

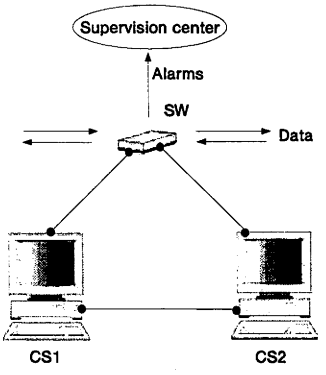


Figure 2.6: Extract of a telecommunication network

control stations *CSi* becomes faulty (event *CSifail*), it emits the alarm *CSiobs* and the other control station takes over (event *NotifyCSifail*). However, in case the switch becomes faulty while being managed by *CSi*, the control station cannot emit *CSiobs* if it becomes faulty.

2.3.2 Modelling of the System

To encode a diagnosis problem symbolically and to be able to compare results with previous work, we have chosen to start from the classical formalism for fault diagnosis in discrete-event systems initially proposed by Sampath and colleagues 1995. In this formalism, the behavioural model is described as a set of finite state machines (FSMs): a FSM represents the nominal, faulty, and observable behaviour of each component. These FSMs are generally obtained following the modelling of the diagnosis problem with a higher level description language which separates the description of the system (i.e. system behaviour) from the description of the diagnosis problem (what is observable, what is faulty) [Lamperti & Zanella, 2004]. The higher description language is out of the scope of this thesis.

Component Model

The component model is described using a FSM in which the transitions correspond to events occurring on the component. Each component can react to *fault* events by changing states and emitting *observable* events. Fault events are not observable. They are a subset of the *unobservable* events. Without loss of generality we assume that the shared events that are used to describe the interactions between components are also unobservable. Finally, in order to model the component's be-

haviour completely, we introduce *normal* events that are unobservable and local to the component.

Definition 1 (Model of a component) *The model of a component is a finite state machine* $G_i = \langle X_i, \Sigma_i, x_{0_i}, T_i \rangle$, *where*

- X_i *is the set of states* ($X_i = \{x_{1_i}, \ldots, x_{m_i}\}$);
- Σ_i *is the set of events* ($\Sigma_i = \{\sigma_{1_i}, \ldots, \sigma_{p_i}\}$);
  $\Sigma_i = \Sigma_{o_i} \cup \Sigma_{u_i}$, *where* Σ_{o_i} *are the observable and* Σ_{u_i} *are the unobservable events;*
  $\Sigma_{s_i} \subseteq \Sigma_{u_i}$ *are the events that are shared among several components;*
  $\Sigma_{f_i} \subseteq \Sigma_{u_i}$ *are the fault events;*
  $\Sigma_{n_i} = \Sigma_{u_i} \setminus (\Sigma_{f_i} \cup \Sigma_{s_i})$ *are the normal events;*
- x_{0_i} *is the initial state;*
- T_i *is the transition set* ($T_i \subseteq X_i \times \Sigma_i \times X_i$).

Figure 2.7 illustrates the component models for our running example (a simplified version thereof for the sake of readability). The components interact with each other via the event *NotifySWfail*. Furthermore the control stations interact with each other via the event *NotifyCSifail*. Initially the components are in states x_0, y_0, and z_0 respectively. In the modelled setting, *CS1* is initially managing *SW*. If a fault occurs in *CS1*, it emits the alarm *CS1obs* and the shared *NotifyCS1fail* event is issued, that is, it is sent by *CS1* and received by *CS2*. *CS2* then changes its state to z_1 and manages the switch. Note that after the switch has become faulty and *NotifySWfail* has been received by the control station *CSi* in charge, this control station changes its state (to y_2 or z_3) in such a way that the alarm *CSiobs* can not be emitted.

In our example, every component G_i can always receive events of the form *Notifyjfail* $\in \Sigma_{s_i}$ sent by component $G_j \neq G_i$. However, to simplify the figures, we depict only events that change the behaviour and hence the state of G_i. For instance, *CS1* can receive one event, namely *NotifySWfail*, and only changes its behaviour if this event is received in state y_0. The transition loops $(y_i \xrightarrow{NotifySWfail} y_i) \in T_i$ for all states $y_i \neq y_0$ are not shown.

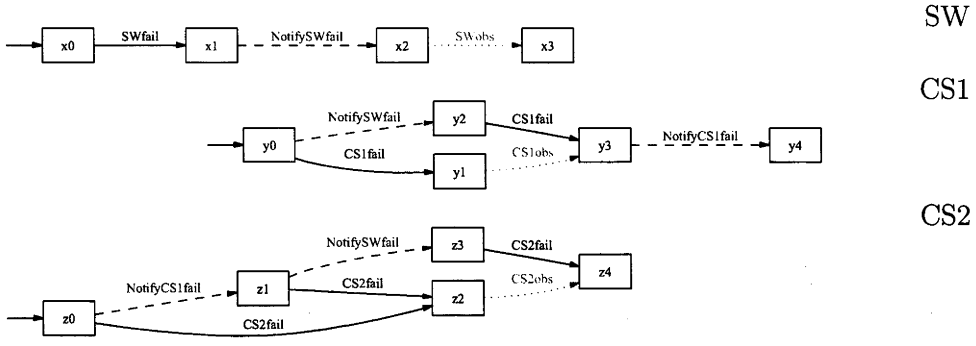


Figure 2.7: A simplified version of component models for the example depicted in Figure 2.6. Solid lines denote fault transitions and dotted lines observable transitions. Dashed lines refer to transitions labelled with shared events.

Global Model

While each component model represents the behaviour of a single part of the system, the global model describes the behaviour of the *whole* system.

Definition 2 (Global model) *The global model of the system is the finite state machine* $G = \langle X, \Sigma, x_0, T \rangle$. *It is defined as the synchronised product of the* n *component models* $G_i = \langle X_i, \Sigma_i, x_{0_i}, T_i \rangle$, *such that*

- X *is the set of system states* ($X = \prod_{i=1}^{n} X_i$);
- $\Sigma = \Sigma_o \cup \Sigma_u$ *are the events with* $\Sigma_f \subseteq \Sigma_u$, $\Sigma_n \subseteq \Sigma_u$ *and* $\Sigma_s \subseteq \Sigma_u$
  Σ_o, Σ_u, Σ_f, Σ_n *and* Σ_s *are the unions of the component's observable, unobservable, fault, normal and shared event sets;*
- x_0 *is the initial state* ($x_0 = \prod_{i=1}^{n} x_{0_i}$);
- T *is the transition set*
  $(T = \{(x_1, \ldots, x_n) \xrightarrow{\sigma} (x'_1, \ldots, x'_n) \mid$
  $\forall i \in 1 \ldots n$ *such that* $\sigma \in \Sigma_i, x_i \xrightarrow{\sigma} x'_i \in T_i$ *and*
  $\forall i \in 1 \ldots n$ *such that* $\sigma \notin \Sigma_i, x_i = x'_i\}$).

The composition ensures that a transition labelled σ is possible in a given global state $(x_1, \ldots, x_n)$ iff it is possible in the respective individual states of all components in which σ is defined [Sampath *et al.*, 1996]. Recall that only shared events are defined for more than one component and hence can lead to a state change of several components. Figure 2.8 shows a part of the global model for the components shown in Figure 2.7.

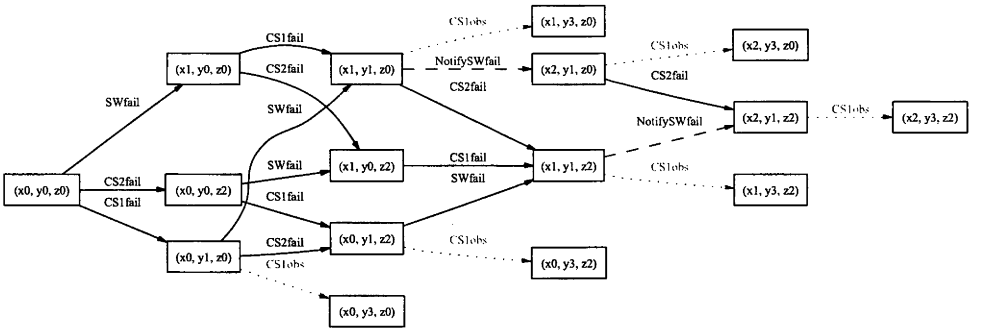


Figure 2.8: (Part of) global model for the component models depicted in Figure 2.7.

2.3.3 Diagnosis Problem for Discrete-Event Systems

The diagnosis problem for discrete-event systems consists in determining all system states and faults that are consistent with a sequence of observations. We formally define this diagnosis information using the concept of event paths.

Definition 3 (Event Paths) *Let* $FSM = \langle X, \Sigma, x_0, T \rangle$ *denote a finite state machine. An event path* $P_{\Sigma'} = x_1 \xrightarrow{\sigma_1} x_2 \cdots \xrightarrow{\sigma_{q-1}} x_q$ *with* $\Sigma' \subseteq \Sigma$ *is a path in the FSM such that* $\sigma_i \in \Sigma'$ $\forall i \in \{1, \ldots q-1\}$ *(note that the path may consist of a single state only). The following functions are defined for every event path:*

- *Start :* $P_{\Sigma'} \mapsto x$ *returns the start state of a path*
- *Targ :* $P_{\Sigma'} \mapsto x$ *returns the target state of a path*
- *EvSet :* $P_{\Sigma'} \times \dot{\Sigma} \mapsto \Sigma$ *returns the set of events in* $\dot{\Sigma} \subseteq \Sigma'$ *of a path*
- *EvSeq :* $P_{\Sigma'} \times \dot{\Sigma} \mapsto \Sigma^*$ *returns the sequence of events in* $\dot{\Sigma} \subseteq \Sigma'$ *of a path*

For instance, from the unobservable event path $P_{\Sigma_{u_{CS2}}} = z_0 \xrightarrow{NotifyCS1fail} z_1 \xrightarrow{NotifySWfail} z_3 \xrightarrow{CS2fail} z_4$ of the component $CS2$ depicted in Figure 2.7 we can retrieve the following:
$EvSet(P_{\Sigma_{u_{CS2}}}, \Sigma_{s_{CS2}}) = \{NotifyCS1fail, NotifySWfail\}$ and
$EvSeq(P_{\Sigma_{u_{CS2}}}, \Sigma_{s_{CS2}}) = [NotifyCS1fail, NotifySWfail]$.

We can also describe path sets as a sequence of event paths and transitions. For instance, the above example path belongs to the path set: $P^1_{\Sigma_{s_{CS2}}} \xrightarrow{CS2fail} P^2_{\Sigma_{s_{CS2}}}$, where $P^2_{\Sigma_{s_{CS2}}}$ consists of the single state z_4.

Now, to determine the diagnosis information ϕ_S we need to look at every path P in the global model from the initial state x_0 to a state x whose observable event sequence $EvSeq(P, \Sigma_o)$ corresponds to the event sequence S actually observed. A

tuple (x, l) is part of the diagnosis, if the fault label l corresponds to the faulty event set $EvSet(P, \Sigma_f)$ of path P.

Definition 4 (Diagnosis information) *The diagnosis information* $\phi_S \in 2^{X \times 2^{\Sigma_f}}$ *that is consistent with a global model* $G = \langle X, \Sigma, x_0, T \rangle$ *and a sequence of observable events* $S = [o_1, \ldots, o_k]$ *is defined as follows:*

$$\phi_S = \{(x, l) \mid P^1_{\Sigma_u} \xrightarrow{o_1} P^2_{\Sigma_u} \cdots \xrightarrow{o_{k-1}} P^k_{\Sigma_u} \xrightarrow{o_k} x \text{ is path in } G \text{ with } l = \bigcup_{j=1}^{k} EvSet(P^j_{\Sigma_u}, \Sigma_f) \text{ and } Start(P^1_{\Sigma_u}) = x_0\}$$

In the following we will refer to every entry $(x, l) \in \phi_S$ as diagnosis candidate. Only if the diagnosis information contains exactly one diagnosis candidate it is possible to determine exactly in which state the system is and which faults have occurred. Otherwise ϕ contains the set of possibilities that explain the observation sequence. For instance, from the existence of the two following paths in the model depicted in Figure 2.8:

- $(x_0, y_0, z_0) \xrightarrow{CS1fail} (x_0, y_1, z_0) \xrightarrow{CS1obs} (x_0, y_3, z_0)$
- $(x_0, y_0, z_0) \xrightarrow{CS1fail} (x_0, y_1, z_0) \xrightarrow{SWfail} (x_1, y_1, z_0) \xrightarrow{CS1obs} (x_1, y_3, z_0)$,

we can conclude that the two following diagnosis candidates belong to $\phi_{[CS1obs]}$:

- $((x_0, y_3, z_0), \{CS1fail\})$
- $((x_1, y_3, z_0), \{CS1fail, SWfail\})$.

The aim of on-line diagnosis approaches is to provide timely diagnosis information. The diagnosis information can directly be computed from the component model or from the global model (see Section 2.4.3). Alternatively, it (or part of it) can be precomputed off-line, and reused on-line for increased efficiency. Since the diagnosis information is usually composed of a high number of candidates, precomputation often leads to larger diagnosis models and increased space requirements. The amount of precomputation performed should therefore depend on the particular time and space efficiency requirements of an application. Hence we present a spectrum of approaches that differ in the amount of compilation performed off-line.

2.3.4 Spectrum of Direct Diagnosis Models

Our direct diagnosis approach is inspired from Sampath's diagnoser approach (1996), in which the diagnosis is retrieved on-line from a single model that effi-

ciently maps observations to faults, the diagnoser. This diagnoser is derived from the global model which in turn is computed from the component models.

In contrast to above approach, we define a spectrum of models and describe the on-line computation of the diagnosis information from each of them. We have already introduced the component and global models; they allow the retrieval of diagnosis information on the basis of relatively small space requirements. We now present two additional models, the abstracted model and the classical diagnoser model, which are more space demanding but are suitable to efficiently diagnose systems on-line. Figure 2.9 illustrates the relationships between the different models.

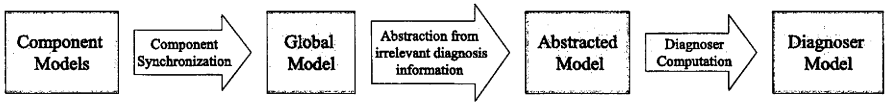


Figure 2.9: Spectrum of direct diagnosis models

In addition to the deterministic diagnoser model defined by Sampath, we choose to introduce a nondeterministic model, the abstracted model, as it allows a more intensive exploitation of the efficient symbolic triggering of transition sets conferred by BDDs. As we will show in section 2.4.3, the abstracted model provides a particularly interesting trade-off between model size and computation time.

Abstracted Model

The abstracted model is derived from the global model by abstracting all unobservable non-fault transitions and the order in which faults can occur, since they have no impact on the diagnosis information (see Definition 4). Hence, in this model, all sequences of unobservable transitions of the global model are replaced by a single transition labelled with the union of the faults in that sequence. In case the sequence consists only of normal and shared events this label is empty.

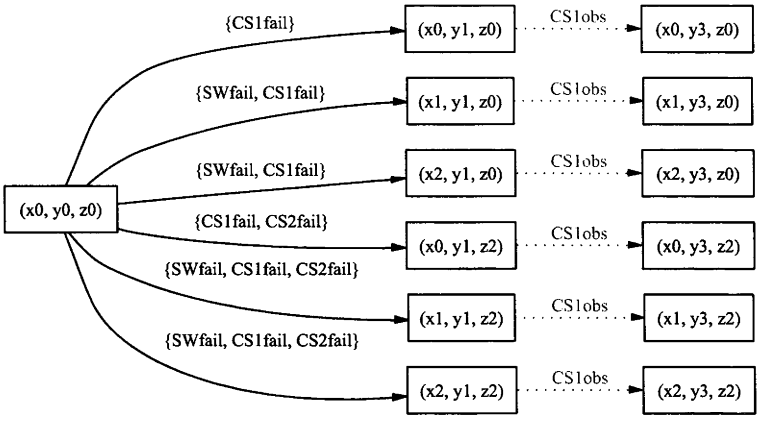


Figure 2.10: Abstracted model for the part of the global model shown in Figure 2.8.

Definition 5 (Abstracted model) *Let $G = \langle X, \Sigma, x_0, T\rangle$ be the global model. The abstracted model is the finite state machine $\widetilde{G} = \langle \widetilde{X}, \Sigma_o, \widetilde{F}, x_0, T_o, \widetilde{T}_F\rangle$, where*

- $\widetilde{X} \subseteq X$ *is the set of states*
  $\widetilde{X} = \{x_0\} \cup \{x \in X \mid \exists \sigma \in \Sigma_o, \exists x' \in X \text{ s.t. } x \xrightarrow{\sigma} x' \in T \text{ or } x' \xrightarrow{\sigma} x \in T\}$
- $\widetilde{F} \subseteq 2^{\Sigma_f}$ *are the fault labels;*
- $T_o \subseteq \widetilde{X} \times \Sigma_o \times \widetilde{X}$ *are the global observable transitions*
  $T_o = \{x \xrightarrow{\sigma} x' \in T \mid \sigma \in \Sigma_o\}$;
- $\widetilde{T}_F \subseteq \widetilde{X} \times \widetilde{F} \times \widetilde{X}$ *are the fault transitions defined as follows:*
  $$\Big\{x \xrightarrow{l} x' \mid \exists \text{ path } x \xrightarrow{\sigma_1} x_1 \cdots \xrightarrow{\sigma_{k-1}} x_{k-1} \xrightarrow{\sigma_k} x' \text{ in } G \text{ with } x, x' \in \widetilde{X}, \sigma_1, \ldots, \sigma_k \in \Sigma_u \text{ and } l = \{\sigma_1, \ldots, \sigma_k\} \cap \Sigma_f\Big\}.$$

Figure 2.10 represents the abstracted model for the part of the global model shown in Figure 2.8. Only the start and target states of observable transitions can be reached in the abstracted model. Thus, for instance, state (x_0, y_0, z_2) of the global model in Figure 2.8 does not appear in Figure 2.10.

The abstracted model enables a more efficient retrieval of diagnosis information than the global model, because we need to consider at most one abstracted fault transition per diagnosis candidate. For instance, to retrieve the diagnosis information $\phi_{[CS1obs]}$ that is consistent with observing $CS1obs$ in the initial state, we trigger all fault transitions starting in $\widetilde{x}_0$ and leading to a start state $\widetilde{x}$ of a

transition labelled $CS1obs$. For each transition sequence $\widetilde{x}_0 \xrightarrow{l} \widetilde{x} \xrightarrow{CS1obs} \widetilde{x}'$ we obtain one diagnosis candidate $(\widetilde{x}', l) \in \phi_{[CS1obs]}$. To obtain the diagnosis information consistent with $S = [CS1obs, CS2obs]$, we need to consider all transition sequences of the form $\widetilde{x}_0 \xrightarrow{l} \widetilde{x} \xrightarrow{CS1obs} \widetilde{x}' \xrightarrow{l'} \widetilde{x}'' \xrightarrow{CS2obs} \widetilde{x}'''$, each of which leads to the diagnosis candidate $(\widetilde{x}''', l \cup l') \in \phi_S$. Recall that when using the global model, we have to consider unobservable paths of undetermined length for each diagnosis candidate.

Diagnoser Model

While the abstracted model allows a more efficient diagnosis than the global model, it still requires the on-line aggregation of fault labels based on the events observed. In contrast, the diagnoser is a deterministic finite state machine whose transitions are only labelled with observations and whose states are directly labelled by the diagnosis information that is consistent with the observations. On-line, the diagnoser efficiently maps sequences of observations to the correct diagnosis information: it suffices to follow the path labelled by the actual observations and look up the label of the resulting diagnoser state.

Definition 6 (Diagnoser) *Let $\widetilde{G} = \langle \widetilde{X}, \Sigma_o, \widetilde{F}, x_0, T_o, \widetilde{T}_F \rangle$ denote the abstracted model. The diagnoser model is the deterministic finite state machine $\hat{G} = \langle \hat{X}, \Psi, \Sigma_o, \hat{x}_0, \hat{R}, \hat{T} \rangle$, where*

- *$\hat{X}$ is the set of diagnoser states ($\hat{X} = \{\hat{x}_0, \ldots, \hat{x}_{q-1}\}$);*
- *$\hat{x}_0$ is the initial diagnoser state;*
- *Ψ is the set of possible diagnosis candidates ($\Psi = X \times F$);*
- *$\hat{R}$ is the diagnoser state labelling function ($\hat{R} : \hat{X} \mapsto 2^{\Psi}$);*
- *$\hat{T}$ is the set of diagnoser transitions ($\hat{T} \subseteq \hat{X} \times \Sigma_o \times \hat{X}$);*
- *$\hat{R}$ and $\hat{T}$ satisfy:*
  $\hat{R}(\hat{x}_0) = \{(x_0, \emptyset)\}$ and
  $\hat{x} \xrightarrow{\sigma} \hat{x}' \in \hat{T}$ iff

$$\hat{R}(\hat{x}') = \{(x', l') \mid \exists (x, l) \in \hat{R}(\hat{x}) \text{ such that either } (x \xrightarrow{\sigma} x') \in T_o \text{ and } l' = l, \text{ or } \exists (x \xrightarrow{l''} x'') \in \widetilde{T}_F \text{ and } \exists (x'' \xrightarrow{\sigma} x') \in T_o \text{ and } l' = l \cup l''\}.$$

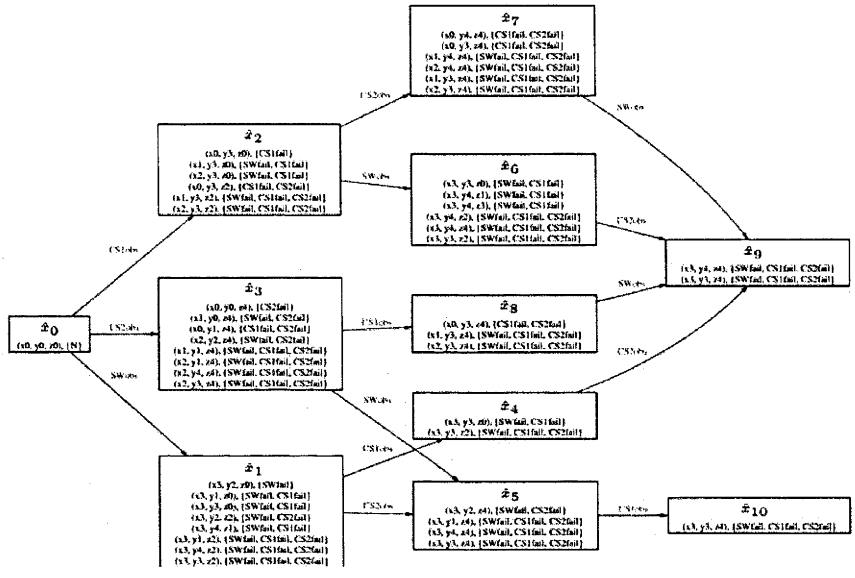


Figure 2.11: Diagnoser model for the component models shown in Figure 2.7.

Figure 2.11 depicts the diagnoser for the component models shown in Figure 2.7. The diagnoser allows for efficient on-line diagnosis whereas the previous models require the diagnosis information to be computed on-line. However, the large diagnoser size constitutes a major problem, and this is exacerbated by the fact that all states and their labels are represented explicitly. In the next section, we will show how we can compactly represent them by means of binary decision diagrams.

2.4 The Symbolic Direct Diagnosis Approach

In this section, we show how to exploit binary decision diagrams both to efficiently represent and compute each of the models introduced above and to efficiently retrieve the diagnosis information from them. Furthermore, we present experimental results that demonstrate how the different diagnosis methods resolve the time/space complexity tradeoff.

2.4.1 Spectrum of Symbolic Direct Diagnosis Models

We now explain how we represent and compute the models introduced in the previous sections using BDDs. Our presentation will emphasise the computation steps that are made either very efficient by the use of symbolic representations (for

instance the test of whether a diagnoser state has been computed already), or that are made rather inefficient (such as the accumulation of fault labels).

Symbolic Component Models

The component model is the basic behavioural model that is used to define and compute all other models of the spectrum. Its symbolic representation is inspired from the classical symbolic FSM representation introduced in Section 2.2. In order to facilitate diagnostic reasoning it is slightly more specific.

Definition 7 (Symbolic model of a component) *The symbolic model* $G_i = \langle b_i^X, b_i^{X'}, b_i^O, b_i^N, b_i^F, b_i^S, X_i, x_{0_i}, T_{o_i}, T_{n_i}, T_{f_i}, T_{s_i} \rangle$ *of a component is described via six BDDs* X_i, x_{0_i}, T_{o_i}, T_{n_i}, T_{f_i} *and* T_{s_i} *over the Boolean variables* $b_i^X, b_i^{X'}, b_i^S, b_i^O, b_i^N, b_i^F$ *where*

- b_i^X *(resp.* $b_i^{X'}$*) are the state variables (resp. primed variables) used to represent start states (resp. target states) of transitions*
- b_i^S *are the shared event variables,*
- b_i^O *are the observable event variables,*
- b_i^N *are the normal event variables,*
- b_i^F *are the fault event variables,*
- X_i *is the Boolean function over* b_i^X *characterising the states,*
- x_{0_i} *is the Boolean function over* b_i^X *characterising the initial state, and*
- T_{o_i}, T_{n_i}, T_{f_i} *and* T_{s_i} *are the Boolean functions over* $b_i^X \cup b_i^{X'}$ *and either* b_i^O, b_i^N, b_i^F, b_i^S, *characterising the observable, normal, fault or shared transition relation, respectively.*

Instead of encoding every component event with the same set of Boolean variables as it is usually done, four different sets are introduced: one set of variables is used to encode events with the same type (fault, normal, shared, observable). In this way, event types are fully distinguishable and, moreover, depending on the type of events, the encoding of events can be different (see below for details). The transition set is also partitioned into four sets, and represented with four BDDs, where each BDD represents the set of transitions labelled with events of the same type. Hence, only one set of event variables is used in each BDD, which leads to a more compact representation.

The symbolic representation also includes the BDD X_i that characterises the set of states of the component model. This BDD is not strictly necessary since it is implicitly represented by the initial state x_{0_i} and the transition sets. However, it is required to perform efficient computations on the states of the component.

Symbolic Event Encoding of a System

Definition 7 provides the generic definition of a symbolic component model, without committing to a particular encoding of events. However, event types are not similar: a normal event will mainly be involved in the triggering of transitions whereas a fault event will additionally be part of the diagnosis information. Consequently, the choice of a suitable encoding for the various event types is a key point to obtain an efficient and generic symbolic framework. Another important factor in choosing how to encode events is the fact that every event, even the ones local to a component, need to be globally distinguishable from the others to obtain a description of the whole system. This leads us to consider the following encodings for each respective event type:

- *Observable events*: any observable event σ belongs to a single component G_i. From a computational point of view, such an event is used to trigger a transition from that component only. To encode σ, there are two solutions:
  1. either σ is encoded with a set of global variables that represent the entire set of observable events of the system. In that case:

$$b_i^O = b^O, \forall i \in \{1, \ldots, n\}$$

  2. or σ is encoded with a set of local variables $b_{loc_i}^O = \{b_{1_i}^o, \ldots, b_{Nr(\Sigma_{n_i})_i}^o\}$ that are dedicated to the encoding of component G_i's observable events, together with an *identifier* of component G_i encoded by a set of global variables $b^C = \{b_1^C, \ldots, b_{Nr(n)}^C\}$. This identifier is necessary to make the event σ globally distinguishable. In that case:

$$b_i^O = b^C \cup b_{loc_i}^O$$

  Both solutions are possible: in practice, their implementation is efficient, although the second solution requires more variables. The main drawback of the first solution is that the encoding does not record that G_i is the emitter of σ, which is a useful information especially for computing the diagnosis information directly from the component models. In the following, the second

solution is chosen.

- *Normal events*: the encoding of a normal event is the same as that of an observable event.

- *Shared events*: as opposed to the previous types of events, a shared event does not belong to a single component. Such an event is mainly used to trigger synchronised transitions (several simultaneous transitions in different components). An efficient synchronisation operation requires an encoding of the shared event via a global set of variables b^S:

$$b_i^S = b^S, \forall i \in \{1, \ldots, n\}$$

- *Fault events*: fault events are used in two cases: they represent an event on a transition like the other event types but they are also part of the diagnosis information (set of faults). To represent faults with an unique encoding, this encoding must allow a representation of any fault event (for the triggering of transitions) and any sets of fault events (for the representation of the diagnosis information (see Definition 4)). This requires the introduction of one Boolean variable per fault event. There is a one to one correspondence between fault events and these variables which ensures that the fault events are globally distinguishable. For instance, let Σ_{f_i} be the set of fault events $\{f_{1_i}, \ldots, f_{k_i}\}$ that can occur in component G_i, the fact that either f_{1_i} and f_{2_i} or just f_{1_i} has occurred, is encoded by: $(b^f_{1_i} \wedge b^f_{2_i} \bigwedge_{j>2} \neg b^f_{j_i}) \vee (b^f_{1_i} \bigwedge_{j>1} \neg b^f_{j_i})$. Now let $encoding(x, B)$ denote the encoding of x over the Boolean variables B. The encoding of the G_i's fault events Σ_{f_i} is then defined as follows:

$$b_i^F = \{b^f_{j_i} \mid f_j \in \Sigma_{f_i}\}; \qquad \forall f_j \in \Sigma_{f_i}, encoding(f_j, b_i^F) = b^f_{j_i} \wedge \bigwedge_{b^f_{k_i} \in b_i^F, k \neq j} \neg b^f_{k_i}.$$

Symbolic Global Model

Given the set of symbolic component models
$G_i = \langle b_i^X, b_i^{X'}, b_i^O, b_i^N, b_i^F, b_i^S, X_i, x_{0_i}, T_{o_i}, T_{n_i}, T_{f_i}, T_{s_i} \rangle$, we can now define and compute the symbolic global model which represents the global behaviour of the system as stated in Definition 2.

Definition 8 (Symbolic global model) *The* symbolic global model $G = \langle b^X, b^{X'}, b^O, b^N, b^F, b^S, X, x_0, T_o, T_n, T_f, T_s \rangle$ *is described via six BDDs* X, x_0, T_o, T_n, T_f *and* T_s *over the Boolean variables* $b^X, b^{X'}, b^O, b^N, b^F$ *and* b^S *where*

- b^X *are the state variables* ($b^X = \cup_{i=1}^{n} b_i^X$);
- $b^{X'}$ *are the primed variables* ($b^{X'} = \cup_{i=1}^{n} b_i^{X'}$);
- b^O *are the observable event variables* ($b^O = \cup_{i=1}^{n} b_i^O$);
- b^N *are the normal event variables* ($b^N = \cup_{i=1}^{n} b_i^N$);
- b^F *are the fault event variables* ($b^F = \cup_{i=1}^{n} b_i^F$);
- b^S *are the shared event variables* ($b^S = \cup_{i=1}^{n} b_i^S$);
- X *is the Boolean function over* b^X *characterising the global states* ($X \subseteq \wedge_{i=1}^{n} X_i$),
- x_0 *is the Boolean function over* b^X *characterising the initial state* ($x_0 = \wedge_{i=1}^{n} x_{0_i}$), *and*
- T_o, T_n, T_s *and* T_f *are the Boolean functions over* $b^X \cup b^{X'}$ *and either* b^O, b^N, b^S *or* b^F *characterising either the observable, normal, shared or fault transition relation, respectively.*

The computation of the global model is based on a synchronised product of the component models (see Definition 2) in which, when a non-shared event occurs in a component G_i, the state of the other components is *steady* (model of an asynchronous system). However, if we are to exploit the BDD representation to implement the synchronisation of the components efficiently, via the $\wedge$ operator, a synchronous product is required. In a synchronous system, when a transition is triggered in a component G_i, a transition is triggered in every other component. In order to implement a synchronous product equivalent to the product defined in Definition 2, we help ourselves to a well-known translation from an asynchronous to a synchronous system (see [Arnold, 1987] for details). Using our symbolic component models, this translation is performed as follows.

- For every state x of a component model G_i, a transition $x \xrightarrow{\epsilon} x$ is added. The event ϵ is the empty event. Thus, if G_i is in state x and a non-shared event is triggered in another component (which means that state x is steady), the transition $x \xrightarrow{\epsilon} x$ is triggered at the same time. Symbolically, the transition

$x \xrightarrow{\epsilon} x$ is represented by the conjunction of the source state x and the target state x, that is by $encoding(x, b_i^X) \wedge encoding(x, b_i^{X'})$. In the algorithm below, this set of empty transitions of component G_i is represented by the BDD $steadyState(i)$.

- For every shared event σ, a transition $x \xrightarrow{\sigma} x$ is added for all states x that belong to a component model in which the event σ is not defined. Thus, if such a component is in state x and a shared event σ is triggered in another component, the transition $x \xrightarrow{\sigma} x$ is triggered at the same time. Symbolically, the transition $x \xrightarrow{\sigma} x$ is represented in the usual way, that is by $encoding(x, b_i^X) \wedge encoding(\sigma, b^S) \wedge encoding(x, b_i^{X'})$. This set of transitions is denoted $extendedTransitions(i)$.

Algorithm 2 presents the computation of the symbolic global model. First the initial state is retrieved as a conjunction of the local initial states (line 4) and we perform the synchronous translation of the model (lines 5–7). Next, the global transitions are retrieved for all new states X_{new} at once using an $\wedge$ operator. For example, in line 12, T_{sNew} contains the set of transitions of the global model that are labelled with shared events and that can be triggered from a state in X_{new}. Since the composition is synchronous, this set is obtained by simple $\wedge$ operations. Lines 15 and 16 retrieve the observable and normal transitions of the global model whose source state is in X_{new} and that are labelled with an event from component G_i. Line 17 retrieves the fault transitions. A fault event in T_{f_i} is represented over the Boolean variables $b_i^F = \{b^f, f \in \Sigma_{f_i}\}$ only (see section 2.4.1). Since T_f represents the fault events with the set b^F instead of the subset b_i^F, it now also needs to represent the fact that none of the faults in $b^F \setminus b_i^F$ has occurred. Therefore the conjunction $\bigwedge_{b^f \in b^F \setminus b_i^F} \neg b^f$ is added to complete the global representation of a fault event from G_i. After the global transitions originating at X_{new} are computed, we determine those of their target states that need to be considered in the next iteration of the algorithm. We do this by first retrieving all target states (line 22) and then identifying those among them that have not previously been computed (line 23).

Note that the global model could be computed more efficiently if the computation of states was not restricted to those reachable from the initial state as it is the case in Algorithm 2. Then only one loop iteration (lines 10–24) would be required to obtain all transitions among states of the complete state set $X = \prod_{i=1}^{n} X_i$. However, since this computation is done off-line, it does not lead to an overhead for the on-line diagnosis. The only point of concern is therefore whether the restriction to

Algorithm 2 *BuildGlob*($G_i = \langle b_i^X, b_i^{X'}, b_i^O, b_i^N, b_i^F, b_i^S, X_i, x_{0_i}, T_{o_i}, T_{n_i}, T_{f_i}, T_{s_i} \rangle$)

1: INPUT: symbolic component models G_i; $i = 1, \ldots, n$

Initialise

2: $x_0 \leftarrow true$; $X \leftarrow false$;
$T_o \leftarrow false$; $T_n \leftarrow false$; $T_f \leftarrow false$; $T_s \leftarrow false$;
3: **for all** components i ($1 \leq i \leq n$) **do**
4: $x_0 \leftarrow x_0 \wedge x_{0_i}$
5: $steadyState(i) \leftarrow initialiseSteadyStates(X_i)$
6: $extendedTransitions(i) \leftarrow initialiseExtendedTransitions(X_i, b^S)$
7: $T_{extend_i} \leftarrow T_{s_i} \vee extendedTransitions(i)$
8: **end for**
9: $X_{new} \leftarrow x_0$

Compute global transitions

10: **while** there are new states (that is as long as $IsDef(X_{new})$) **do**
11: $X \leftarrow X \vee X_{new}$
12: $T_{sNew} \leftarrow X_{new} \wedge \bigwedge_{i \in \{1,\ldots,n\}} T_{extend_i}$
13: $T_s \leftarrow T_s \vee T_{sNew}$
14: **for all** components i ($1 \leq i \leq n$) **do**
15: $T_{oNew} \leftarrow X_{new} \wedge T_{o_i} \wedge \bigwedge_{j \in \{1,\ldots,n\}, j \neq i} steadyState(j)$
16: $T_{nNew} \leftarrow X_{new} \wedge T_{n_i} \wedge \bigwedge_{j \in \{1,\ldots,n\}, j \neq i} steadyState(j)$
17: $T_{fNew} \leftarrow X_{new} \wedge T_{f_i} \wedge \bigwedge_{b^f \in b^F \setminus b_i^F} \neg b^f \wedge \bigwedge_{j \in \{1,\ldots,n\}, j \neq i} steadyState(j)$
18: $T_o \leftarrow T_o \vee T_{oNew}$
19: $T_n \leftarrow T_n \vee T_{nNew}$
20: $T_f \leftarrow T_f \vee T_{fNew}$
21: **end for**

Extract new states

22: $X_{targ} \leftarrow ExtractVar(T_o \vee T_n \vee T_f \vee T_s, b^{X'})$
23: $X_{new} \leftarrow SwapVar(X_{targ}, b^X, b^{X'}) \wedge \neg X$
24: **end while**
25: OUTPUT: global states X, initial state x_0 and transitions T_o, T_n, T_f and T_s

the reachable states leads to a larger or smaller symbolic representation. For the examples we used (see page 53) this restriction reduced the number of states to 25% and the number of transitions to 24% which overall reduced the size of the symbolic global model representation to 36%.

Note that there is not necessarily a connection between the size of the state set and its symbolic representation (given that they are both encoded using the same number of Boolean variables). We decided to apply the restriction to the reachable state set already to the computation of the global model, since it becomes eventually necessary (see Section 2.4.1).

Symbolic Abstracted Model

With the computation of the symbolic global model we have already defined two BDDs, namely x_0 and T_o, that are used to represent the abstracted model symbolically.

Definition 9 (Symbolic abstracted model) *Given the global model* $G = \langle b^X, b^{X'}, b^O, b^N, b^F, b^S, X, x_0, T_o, T_n, T_f, T_s\rangle$, *the abstracted model* $\widetilde{G} = \langle b^X, b^{X'}, b^O, b^F, \widetilde{X}, x_0, T_o, \widetilde{T}_f\rangle$ *is described via four BDDs* $\widetilde{X}, x_0, T_o$ *and* $\widetilde{T}_f$ *over the Boolean variables* $b^X, b^{X'}, b^O$ *and* b^F *where*

- $\widetilde{X}$ *is the Boolean function over* b^X *characterising the states of the abstracted model, and*
- $\widetilde{T}_f$ *is the Boolean function over* $b^X \cup b^F \cup b^{X'}$ *characterising the abstracted fault transition relation.*

To compute the abstracted fault transitions we need to determine, for each state $x \in \widetilde{X}$, (i) all target states $X_{reach} \in \widetilde{X}$ reachable from x by triggering only unobservable transitions, and (ii) all fault events that have occurred along a path from x to $x_{reach} \in X_{reach}$. For this purpose, we start by combining all unobservable transitions into a single transition set T_u. Each transition in T_u is labelled with the fault information, that is either with the fault event that triggered the transition (for transitions in T_f), or with the empty fault event $F_\emptyset = \bigwedge_{j=1}^{|\Sigma_f|} \neg b_j^f$ (for transitions in T_n and T_s). Thus T_u is defined as follows: $T_u = T_f \vee (ExtractVar(T_n \vee T_s, b^X \cup b^{X'}) \wedge F_\emptyset)$.

Then, using T_u, we compute FX_{reach} (defined over variables $b^F \cup b^X$), that is, all states $x_{reach} \in X_{reach}$ and the faults that have occurred along a path from x to

x_{reach}, by modifying the reachability algorithm (Algorithm 1) shown on page 27. Now, at each iteration (lines 4–10), we compute not only the set of states X_{new} from which transitions still need to be triggered but also the set FX_{new} which additionally includes all faults F that have occurred along a path from x to a state in X_{new}. The modified reachability algorithm is given below:

$CompReachModified(b^{\Sigma}, b^{X}, b^{X'}, X, T_u)$

$X_{reach} \leftarrow false$
$X_{new} \leftarrow X$
$FX_{new} \leftarrow X \wedge F_{\emptyset}$
while there are new states (that is as long as $IsDef(FX_{new})$) **do**
  $FX_{targ} \leftarrow AddFault(T_{new}, FX_{new})$
  $FX_{new} \leftarrow FX_{targ} \wedge \neg FX_{reach}$
  $X_{new} \leftarrow ExtractVar(FX_{new}, b^{X})$
  $FX_{reach} \leftarrow FX_{reach} \vee FX_{new}$
end while

The procedure makes use of the function $AddFault$ to add every fault f that occurred in a state $x_{new} \in X_{new}$ (i.e. the fault for which there is a transition $x_{new} \xrightarrow{f} x_{targ}$ with $x_{targ} \in X_{targ}$) to the set of faults F that have occurred on a path from state x to x_{new}. This accumulation of faults performed by $AddFault$ is illustrated in Figure 2.12 and detailed below.

Given T_{new} and FX_{new}, $AddFault$ first extracts all states X_{nt} (origins and targets) of T_{new}. Since the faults are then abstracted (i.e. X_{nt} is not defined over variables in b^F) we can associate the previous faults F to this state set by applying the $\wedge$ operator to sets X_{nt} and FX_{new} resulting in FX_1 (see top left of Figure 2.12). Next, we add the new faults (kept in T_{new}) to FX_1 using function $UpdateFault(oldFault, newFault)$. To add a fault f' (encoded as $b^f_{i'} \wedge_{i' \neq j} \neg b^f_j$) to a BDD B we need to change the sign of variable $b^f_{i'}$ in B. This is done by abstracting variable $b^f_{i'}$ from B and conjoining it with $b^f_{i'}$ (i.e., $AbstractVar(B, b^f_{i'}) \wedge b^f_{i'}$). Now, a variable can only be abstracted from an entire BDD (not from part of it). Therefore, in order to add a new fault f' we first need to isolate all states $X'_{nt} \subseteq X_{nt}$ that require this addition. This means that we have to compute all state pairs (x'_{new}, x'_{targ}) for which there is a transition $x'_{new} \xrightarrow{f'} x'_{targ}$. The function $UpdateFault(oldFault, newFault)$ performs these computations for all the faults that need to be added.

The Figure shows an example in which the state x_1 (encoded as b^X_1 and $b^{X'}_1$ resp.) is reached from state x_0 (encoded as $\neg b^X_1$ and $\neg b^{X'}_1$ resp.) via a transition labelled

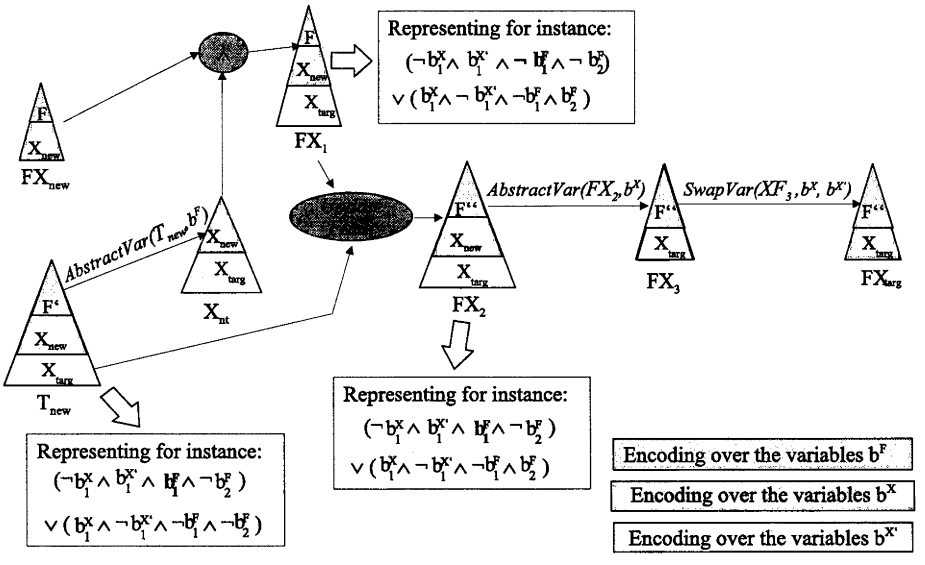


Figure 2.12: Computation steps of function *AddFault* that determines the faults associated with states X_{targ} by combining the faults F associated with their predecessor states X_{new} and the new fault F' labelling T_{new}.

f_1 (encoded as $b_1^F \wedge \neg b_2^F$) and state x_0 is reached from state x_1 via a transition labelled with the empty fault label (that is a shared or normal transition). Hence only the fault label of state x_0 is modified by changing the sign of variable b_1^F.

Note that fault sets cannot be updated at once, since for each new fault we have to retrieve the corresponding previous fault sets first. Hence function *UpdateFault*, that is the symbolic update of fault sets is not very efficient. However, the computation of the abstracted model is done off-line and therefore does not slow down the on-line diagnosis based on this model. Our experiments show that the off-line aggregation of successive faults reduces the diagnosis time (see Section 2.4.3).

Symbolic Diagnoser Model

We now define the symbolic diagnoser representation and describe its computation based on the abstracted model.

Diagnoser Representation

Here, in contrast to the previous symbolic models, we do not only require an encoding for the model's states and transitions but also an encoding Φ for the state labelling function $\hat{R}$ of the diagnoser.

Definition 10 (Symbolic diagnoser model) *Given the symbolic abstracted model* $\widetilde{G} = \langle b^X, b^{X'}, b^O, b^F, \widetilde{X}, x_0, T_o, \widetilde{T}_f \rangle$, *the symbolic model of a diagnoser* $\hat{G} = \langle b^{\hat{X}}, b^{\hat{X}'}, b^X, b^O, b^F, \hat{X}, \hat{x}_0, \Phi, \hat{T} \rangle$ *is described via four BDDs* $\hat{X}$, $\hat{x}_0$, Φ, *and* $\hat{T}$, *involving the new Boolean variables* $b^{\hat{X}}$ *and* $b^{\hat{X}'}$ *where*

- $b^{\hat{X}}$ *are the diagnoser state variables* ($b^{\hat{X}} = \{b^{\hat{x}}_1, \ldots, b^{\hat{x}}_{Nr(\hat{X})}\}$);
- $b^{\hat{X}'}$ *are the primed diagnoser state variables* ($b^{\hat{X}'} = \{b^{\hat{x}'}_1, \ldots, b^{\hat{x}'}_{Nr(\hat{X})}\}$);
- $\hat{X}$ *is the Boolean function over* $b^{\hat{X}}$ *characterising the diagnoser states;*
- $\hat{x}_0$ *is the Boolean function over* $b^{\hat{X}}$ *characterising the initial diagnoser state,*
- Φ *is the Boolean function over* $b^{\hat{X}} \cup b^X \cup b^F$ *characterising the diagnoser state labelling relation, and*
- $\hat{T}$ *is the Boolean function over* $b^{\hat{X}} \cup b^O \cup b^{\hat{X}'}$ *characterising the transition relation.*

The label $\hat{R}(\hat{x})$ of a diagnoser state $\hat{x}$ is computed by conjoining the BDDs representing $\hat{x}$ and Φ and abstracting the state variables $b^{\hat{X}}$ from the result. In the following we use function $GetInfo$ to perform these operations. The function is defined as follows: $GetInfo(bdd, bdd_1, \{b_1, \ldots, b_m\}) = AbstractVar(bdd \wedge bdd1, \{b_1, \ldots, b_m\})$. Hence we get $\hat{R}(\hat{x}) = GetInfo(\Phi, \hat{x}, b^{\hat{X}})$.

Diagnoser Computation

The encoding of a diagnoser state $\hat{x}$ depends on the total number q of diagnoser states which is *a priori* unknown and even difficult to estimate. In the worst case $q = 2^{|\Psi|}$ and therefore $2|X||2^{\Sigma_f}|$ new variables are theoretically needed. However, in practice, q will be much smaller and introducing that many variables will lead to an unnecessarily costly representation. To avoid the introduction of all $2|X||2^{\Sigma_f}|$ variables, we start with one single variable to encode the initial diagnoser state, and continually increase the number of variables, as needed during the construction. Every time a new variable $b^{\hat{x}}_j$ is needed, we update each BDD bdd containing variables in $b^{\hat{X}}$ (respectively $b^{\hat{X}'}$) by conjoining them with $\neg b^{\hat{x}}_j$ (respectively $\neg b^{\hat{x}'}_j$). Each update can be performed in $\mathbb{O}(|bdd|)$.

Algorithm 3 shows the symbolic diagnoser computation starting from the abstracted model. First, in step 3, the initial diagnoser state $\hat{x}_0$ is computed using the function $GetNewState$ which returns the encoding of a new diagnoser state

over the Boolean variables $b^{\hat{X}}$ and if needed, handles the introduction of new variables as discussed above. The diagnosis information of $\hat{x}_0$ is the conjunction of the initial global state and the empty fault label $F_\emptyset$, and is added to the state labelling relation Φ. We also initialise the set of states $\hat{X}_{new}$ from which transitions still need to be computed.

Algorithm 3 $BuildDiag(\widetilde{G} = \langle b^X, b^{X'}, b^O, b^F, \widetilde{X}, x_0, T_o, \widetilde{T}_f \rangle)$

1: INPUT: symbolic global model G
2: Initialise
$\hat{T} \leftarrow false; \quad \hat{X} \leftarrow false$
3: Compute initial state and its information
$\hat{x}_0 \leftarrow GetNewState()$
$\Phi \leftarrow \hat{x}_0 \wedge x_0 \wedge F_\emptyset$
$\hat{X}_{new} \leftarrow \hat{x}_0$
4: **while** there are new states, that is as long as $IsDef(\hat{X}_{new})$ **do**
5: Get new state and its information
$\hat{x}_{new} \leftarrow GetConj(\hat{X}_{new})$
$\hat{X}_{new} \leftarrow \hat{X}_{new} \wedge \neg \hat{x}_{new}$
$\hat{x}_{newInfo} \leftarrow GetInfo(\Phi, \hat{x}_{new}, b^{\hat{X}})$
$\hat{X} \leftarrow \hat{X} \vee \hat{x}_{new}$
6: Compute the diagnosis information of all states reachable from $\hat{x}_{new}$ and the corresponding observation
$\widetilde{T}_{new} \leftarrow \widetilde{T}_f \wedge ExtractVar(\hat{x}_{newInfo}, b^X)$
$reachUnobs \leftarrow \hat{x}_{newInfo} \vee AddFault(\widetilde{T}_{new}, \hat{x}_{newInfo})$
$reachObs \leftarrow SwapVar(AbstractVar(reachUnobs \wedge T_o, b^X), b^X, b^{X'})$
7: Determine all states reached and the corresponding observation
$reachTarg \leftarrow GetStates(reachObs, \Phi)$
8: Determine the new states reached and add them to $\hat{X}_{new}$
$newTarg = ExtractVar(reachTarg, b^{\hat{X}}) \wedge \neg \hat{X}$
$\hat{X}_{new} \leftarrow \hat{X}_{new} \vee newTarg$
9: Add new states and their labels to Φ
$\Phi \leftarrow \Phi \vee AbstractVar(newTarg \wedge reachTarg, b^O)$
10: Add new transitions to the transition set $\hat{T}$
$obsTarg \leftarrow ExtractVar(SwapVar(reachTarg, b^{\hat{X}}, b^{\hat{X}'}), b^O \cup b^{\hat{X}'})$
$\hat{T} \leftarrow \hat{T} \vee (\hat{x}_{new} \wedge obsTarg)$
11: **end while**
12: OUTPUT: symbolic diagnoser states $\hat{X}$ and $\hat{x}_0$, their labelling Φ, and diagnoser transitions $\hat{T}$

Until a fixed point is reached, a new diagnoser state $\hat{x}_{new}$ is retrieved and removed from $\hat{X}_{new}$ (step 5). The diagnosis information $\hat{x}_{newInfo}$ of the new state is obtained from the labelling relation Φ.

Next, the diagnosis information of all target states of $\hat{x}_{new}$ is computed (step 6).

For this purpose we need to consider all global states X_{unobs} in which the system could be before the next event is observed. These are retrieved by first triggering all unobservable transition sequences of the global model starting in states of $\hat{x}_{newInfo}$, that is, the set $\widetilde{T}_{new}$ of abstracted fault transitions. The new faults that have occurred since the last observation are added using function *AddFault* (see Figure 2.12). Second, we trigger all observable transitions starting in states of X_{unobs} to obtain the diagnosis information of all target states of $\hat{x}_{new}$ and the observations that are consistent with it. The resulting BDD *reachObs* is defined over the variables $b^O \cup b^X \cup b^F$.

In step 7 of the algorithm, the target diagnoser states of $\hat{x}_{new}$ are obtained using function *GetStates*, which for each observation o retrieves the diagnoser state labelled with $GetInfo(reachObs, o, b^O)$ as described below. The resulting BDD *reachTarg* is now defined over the variables $b^O \cup b^{\hat{X}} \cup b^X \cup b^F$.

Next, we identify the new diagnoser states encoded in *reachTarg* (step 8), and add them together with their labels to the labelling relation Φ (step 9). Finally, we encode all target diagnoser states of $\hat{x}_{new}$ returned by the function *GetStates* in step 7 over the variables $b^{\hat{X}'}$ in order to add all transitions starting in $\hat{x}_{new}$ to the transition set $\hat{T}$ (step 10).

Termination of Diagnoser Computation

To guarantee the termination of the algorithm, it is crucial to detect in function *GetStates* (step 7) whether a diagnoser state reached from $\hat{x}_{new}$ via a transition labelled o has already been computed, that is, to detect whether there exists a state $\hat{x}$ such that $GetInfo(\Phi, \hat{x}, b^{\hat{X}})$ is the same as $\hat{x}_{info} = GetInfo(reachObs, o, b^O)$. In an enumerative approach this requires the consideration of every existing diagnoser state to check whether its diagnosis information equals $\hat{x}_{info}$. Symbolically we do not need to look at these states individually and instead compute the following diagnoser state sets:

- S_1: set of diagnoser states whose labels are subsets of $\hat{x}_{info}$
- S_2: set of diagnoser states whose labels are supersets of $\hat{x}_{info}$

The diagnoser state $\hat{x}$ labelled with $\hat{x}_{info}$ can be obtained by intersecting these sets ($\hat{x} = S_1 \cap S_2$). Note that since no two diagnoser states have the same label, the intersection yields at most one state. In case the intersection is empty the diagnoser state set does not contain a state labelled $\hat{x}_{info}$.

Since state labels of single diagnoser states are encoded as disjunctions it is symbolically not possible to directly determine whether the labels of a set of states $\hat{X}_1$ are a subset of labels of another state set $\hat{X}_2$. Hence we cannot directly retrieve S_1 and S_2. Therefore, to compute $S_1 \cap S_2$ we first compute $\hat{X} \setminus S_1$, that is all states in the diagnoser state set $\hat{X}$ that do not belong to S_1. Second, instead of retrieving $\hat{X} \setminus S_2$ which would again require the consideration of the whole state set $\hat{X}$, we compute $S_1 \setminus S_2$ as shown on the top right of Figure 2.13. Now symbolically, the operation $\setminus$ is equivalent to the operations $\wedge\neg$. Hence we can obtain $S_1 \wedge S_2$ by computing first $\neg S_1$ and second $S_1 \wedge \neg S_2$ and by combining these results using two $\neg$ and one $\wedge$ operator as shown on the bottom right of Figure 2.13.

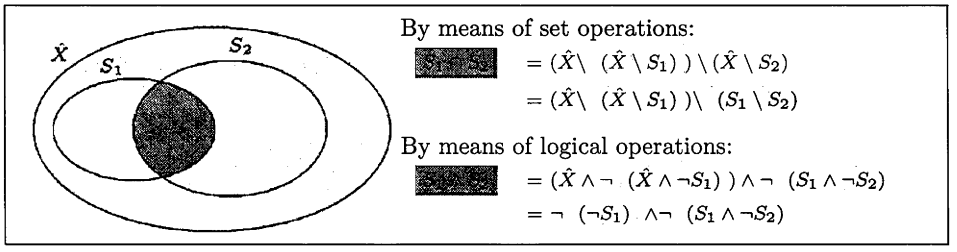


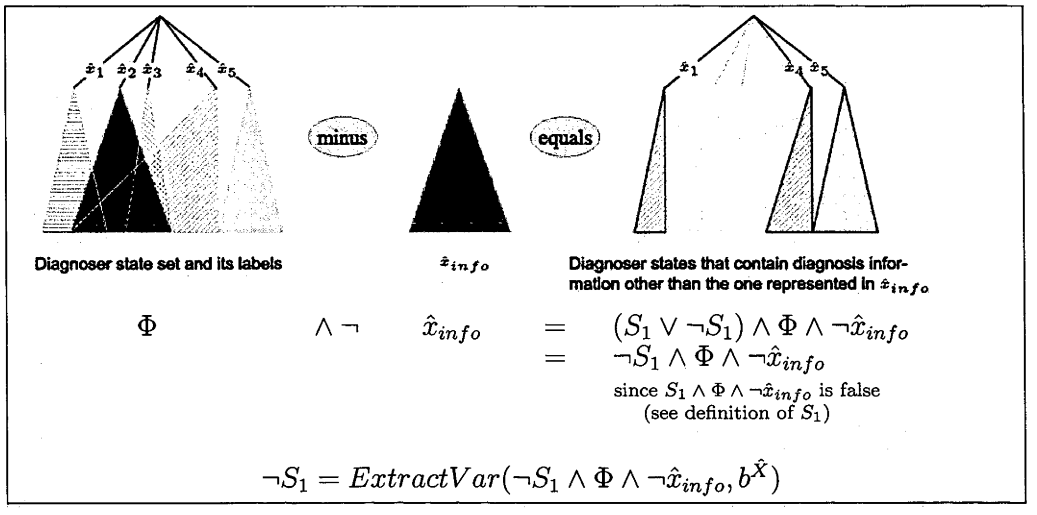
Figure 2.13: Derivation of an efficient symbolic computation of the intersection of the state sets S_1 and S_2.

The two computation steps of retrieving $\neg S_1$ and $S_1 \wedge \neg S_2$ are illustrated in Figure 2.14. The top left BDD in the Figure represents the diagnoser labelling relation Φ for states $\hat{x}_1, \ldots, \hat{x}_5$ where the state labels are shown as triangles. The base (bottom line) of these triangles represents the truth values of label entries. Hence we can illustrate the different ways in which state labels can relate to each other, namely:

- two states can share a set of label entries, which is the case if triangle bases overlap (see state pair $(\hat{x}_1, \hat{x}_2)$);
- the label of one diagnoser state can be a subset of another state's label, which is the case if one triangle base is completely covered by another (see state pairs $(\hat{x}_3, \hat{x}_2)$ and $(\hat{x}_2, \hat{x}_4)$);
- two diagnoser states share no label entries, which is the case if the triangle bases do not overlap (see state pair $(\hat{x}_2, \hat{x}_5)$).

The figure illustrates all possible ways in which the label of a state $(\hat{x}_2)$ can relate to the labels of other states $(\hat{x}_1, \hat{x}_3, \hat{x}_4$, and $\hat{x}_5)$.

The top of Figure 2.14 describes the symbolic computation of $\neg S_1$ and the bottom illustrates the computation of $S_1 \wedge \neg S_2$. In order to determine $\neg S_1$, that is



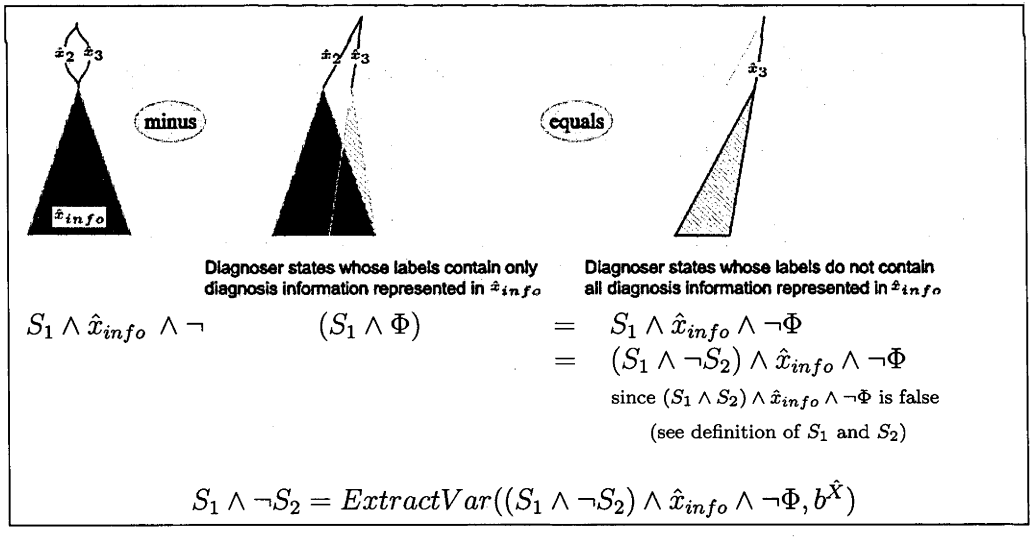


Figure 2.14: Computation of state set $\neg S_1$ at the top and $S_1 \wedge \neg S_2$ at the bottom. All terms represent BDDs. Lines denote encodings over the state variables $b^{\hat{X}}$ and triangles encodings over the variables $b^X \cup b^F$.

all states whose labels are not a subset of $\hat{x}_{info}$, we subtract $\hat{x}_{info}$ from the diagnoser state set (see top of Figure 2.14). The resulting BDD is depicted on the top right of the Figure. It shows that states $\hat{x}_1, \hat{x}_4$, and $\hat{x}_5$ do not belong to subset S_1 and hence need not be considered further (see the middle BDD at the bottom).

To compute $S_1 \wedge \neg S_2$, we first label all states in S_1 with $\hat{x}_{info}$ (see graph at the bottom left). From this BDD we subtract the labels of the remaining diagnoser states. This yields all states that belong to S_1 but not to S_2 ($\hat{x}_3$ as depicted on the bottom right).

After retrieving the state sets $\neg S_1$ and $S_1 \wedge \neg S_2$, we can now efficiently compute $S_1 \wedge S_2$ using only two $\neg$ and one $\wedge$ operation as shown in Figure 2.13. For the example illustrated in Figure 2.14 we obtain $\hat{x}_{info} = GetInfo(\Phi, \hat{x}_2, b^{\hat{X}})$.

Note that these symbolic computations can not only be used to search for a diagnoser state labelled with particular diagnosis information but for any problem of the following form:

> Given a set of elements $E = \{e_1, \ldots, e_n\}$ and a set of containers $C = \{C_1, \ldots, C_m\}$ such that C_i contains the elements $E_i \subseteq E$ for all $i = \{1, \ldots, m\}$. Further let the entire set C be encoded using a single BDD. How can this representation be used (without computing the individual C_i's) to obtain all containers that have exactly the same elements as a new container C'?

In our case, the set E is composed of the different diagnosis information and the containers correspond to the diagnoser states. Alternatively, for instance, one could regard the set E as a set of attributes and the set C as a set of products having each some of the attributes. Our algorithm can then be used to search for all the products that have a specific set of attributes, without the need of considering each product individually.

BDD variable order for diagnoser representation

We close this section with a remark on the BDD variable ordering used for the diagnoser representation. As shown in Section 2.2.2 this can significantly impact the BDD size. However, for our purposes it is equally important to be able to efficiently use the diagnoser for the on-line diagnosis. In fact, for our examples we found that the variable ordering had a much higher impact on the diagnosis time (factor of 80 between fastest and slowest diagnosis) than on the diagnoser size

(factor of 1.5 between largest and smallest diagnoser representation). We therefore decided not to use any of the BDD variable ordering heuristics aiming at reducing the BDD size, but rather chosen the "fastest" of the encodings we considered (which also was the almost "biggest" one).

The labelling function Φ is encoded using the following variable order (from root to leaves): $b^{\hat{X}}, b^{X}, b^{F}$. The transition variables are ordered as follows: b^{C}, b^{O}, $(b^{\hat{X}}, b^{\hat{X}'})$, where the state variables are interleaving. For instance the ordering for a four state diagnoser with two different observations and composed of two components is: $b_1^C, b_1^O, b_1^{\hat{X}}, b_1^{\hat{X}'}, b_2^{\hat{X}}, b_2^{\hat{X}'}$. This strategy of first encoding the component identifiers, then the event variables and finally the interleaving states is also used for the symbolic representations of the remaining models.

2.4.2 Comparison of Symbolic and Enumerative Diagnosers

With the computation of the symbolic diagnoser, we have obtained a symbolic model which compactly represents the entire diagnosis information. We now consider the impact of the symbolic representation on model size and computation time.

Our symbolic approach is implemented in C++ on top of the CUDD BDD package [Somenzi, 2005]. In order to measure the impact of symbolic techniques on the diagnoser representation and computation, we have also implemented Sampath's diagnoser approach in an enumerative way. In contrast to the original implementation in the UMDES-LIB software library,[2] our enumerative diagnoser implementation reports the space requirements for the generated diagnoser transitions and states. In the symbolic case, the size of the diagnoser was determined by counting the nodes in the BDDs representing it and multiplying this number by the space requirements to represent one BDD node. Our two implementations enable us to present experimental evidence that the symbolic approach yields important gains in synthesis time and space, considering a system derived from our example application as a case study [Rozé & Cordier, 2002].

In our study, there are 9 observable events, 11 fault types, and 8 other unobservable events. The switch model has 12 states and 18 transitions, the primary control station 13 states and 15 transitions and the backup control station 19 states and 28 transitions. This yields a global model of 1062 states and 2911 transitions. In order to observe how the two approaches (symbolic/enumerative) scale, we also consid-

[2] `http://www.eecs.umich.edu/umdes/toolboxes.html`

ered "lighter" versions of the example, where groups of fault types are fusioned. This yields 5 different versions $V_1 \ldots V_5$, with a number of fault types ranging from 3 to the original 11.

Experiments were run on a 1GHz Pentium III with 512 Mbytes of memory. The left chart on Figure 2.15 compares the time taken by the symbolic and enumerative methods to produce the diagnoser. After 2 days of computation, neither the UMDES demonstration tool nor our enumerative implementation was able to compute a diagnoser for the two larger versions, while the symbolic approach remained feasible. In fact, here the computation was performed at least three orders of magnitude faster. In order to determine the space requirements for the enumerative diagnoser representation for these examples, we retrieved this diagnoser on the basis of the symbolic one. This was done by examining the paths of the BDDs used to represent the diagnoser.

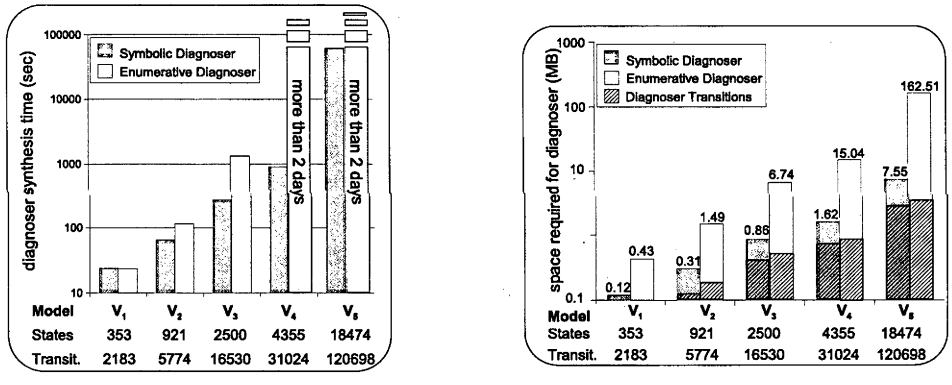


Figure 2.15: Time and space performance comparison

The right chart in Figure 2.15 compares the space needed to store the resulting diagnosers. The superiority of the symbolic method increases with the model size, and exceeds an order of magnitude for the largest version. From the results, it can be conjectured that the space requirements of the symbolic approach for large models will often only represent a negligible portion of those of the enumerative setting. The Figure also shows that, for both approaches, most of the space was used to store labels (note the logarithmic scale). In particular, our symbolic encoding only performs marginally better than the enumerative approach with respect to storing diagnoser transitions. This is due to the fact that the Boolean variables introduced to represent transition sets allow the representation of all transitions that are theoretically possible, while in practice, the diagnoser is deterministic and not all observations are possible in each given diagnoser state. For instance, the diagnoser for version V_5 only contains 0.004% of the theoretically possible transi-

tions.

The superiority of the symbolic diagnoser representation results from the fact that the set of diagnosis candidates, that is the set of possible faults and system states that are consistent with a sequence of observations, is very large and can be encoded with relatively few Boolean variables. Hence, here we are able to exploit the advantages provided by BDDs to compactly represent large sets of data. In general, we believe that symbolic methods are particularly suitable for diagnosis approaches that require the representation of such large sets. This is the case for all fault identification approaches which we target in this chapter. Note that this also holds for representing the fault information of the failure identifiers we presented in [Schumann & Pencolé, 2006]. In contrast to the diagnoser these FSM are only labelled by the faults (and not the corresponding system states) and hence have fewer states than the latter.

2.4.3 Symbolic On-Line Diagnosis

In the previous section, we have shown that the symbolic precomputation of the complex diagnosis information, that is the computation of diagnoser state labels, is significantly faster than the enumerative precomputation. In this section, we exploit the symbolic representation to efficiently compute the diagnosis information on-line, given the events actually observed. We describe the procedure for computing this information using each of the models introduced. Finally, we present an experimental analysis of the extent to which each of the symbolic on-line diagnosis algorithms resolves the tradeoff between space and diagnosis time and compare them to the corresponding enumerative diagnosis approaches.

On-line Diagnosis Algorithms

On-line diagnosis aims to detect faults while the system is working. Each time an event σ is observed, the diagnosis information $\hat{x}'_{info}$ is derived based on σ, one of the models, and the previous diagnosis information $\hat{x}_{info}$. In an enumerative approach, this on-line computation of $\hat{x}'_{info}$ is very slow since all diagnosis entries of $\hat{x}_{info}$ need to be considered individually (except for the diagnoser). In this section we show how we can avoid this individual consideration by symbolically retrieving $\hat{x}'_{info}$.

For on-line diagnosis based on the diagnoser, this update requires the computation of the diagnoser state $\hat{x}'$ that is labelled with the new diagnosis information. Given the diagnoser state $\hat{x}$ labelled with the previous diagnosis information and a

new observation o, we can obtain $\hat{x}'$ using function $GetInfo(\hat{T}, \hat{x} \wedge o, b^{\hat{X}} \cup b^{O})$. After encoding $\hat{x}'$ over the variables $b^{\hat{X}}$ we simply retrieve the new diagnosis information $\hat{x}'_{info}$ from the state labelling relation Φ, that is, $\hat{x}'_{info} \leftarrow GetInfo(\Phi, \hat{x}', b^{\hat{X}})$.

With the remaining models, $\hat{x}'_{info}$ is not precomputed and needs to be determined on-line. However, its on-line computation is very similar to its off-line computation. For the diagnosis based on the abstracted model, we first compute the diagnosis information *reachUnobs* consistent with the previous events and the fact that time has passed, as described in step 6 of Algorithm 3: $reachUnobs \leftarrow \hat{x}_{info} \vee AddFault(\widetilde{T}_f \wedge ExtractVar(\hat{x}_{newInfo}, b^X), \hat{x}_{info})$. After triggering all transitions labelled o from states in *reachUnobs*, we can extract the new diagnosis information as follows: $\hat{x}'_{info} \leftarrow SwapVar(ExtractVar(reachUnobs \wedge T_o \wedge o, b^{X'} \cup b^F), b^X, b^{X'})$.

On-line diagnosis based on the global model is analogous to that based on the abstracted model. Only the computation of *reachUnobs* requires an additional step, namely the computation of all abstracted fault transitions starting in states of $\hat{x}_{info}$, as described in Section 2.4.1.

To compute the diagnosis information on the basis of the component models, we consider, for all of them, all unobservable event paths starting in states contained in $\hat{x}_{info}$. For the component in which o is defined, we also consider all transitions labelled o that start in one of the target states of the unobservable paths. Figure 2.16 shows the relevant component parts if $\hat{x}_{info} = \{((x_0, y_0, z_0), \emptyset)\}$ and $o = CS1obs$ is observed in our working example. The new diagnosis information is retrieved by computing, on-line, the diagnoser based on these component parts. Note that this diagnoser consists of exactly one transition, which is labelled by the new event observed.

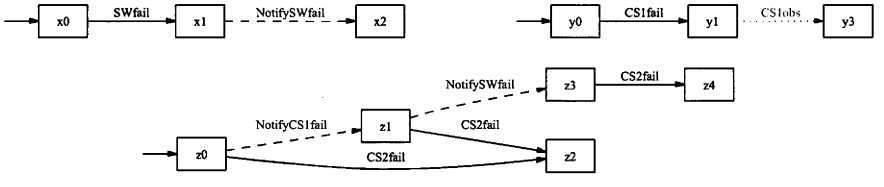


Figure 2.16: Parts of the component models depicted in Figure 2.7 that are relevant to the update of the initial diagnosis information given the new observation $CS1obs$.

In summary, while the diagnosis based on the diagnoser consists in triggering a single transition labelled with the new event observed, the diagnosis based on the other models consists in determining the label of the target diagnoser state on-line, by exploring the part of the model that is relevant to the update of the diagno-

sis information. These approaches only compute (but do not store) the relevant diagnoser path — the path labelled by the events actually observed. Naturally, this targeted computation has reduced space requirements compared to algorithms based on the diagnoser. On the other hand, it also leads to an increase in diagnosis time. The next section gives experimental evidence of this tradeoff.

Experimental Evaluation of On-line Diagnosis Approaches

We describe the performance in diagnosis time and space of our on-line diagnosis algorithms, using the same case study as in Section 2.4.2. For that purpose, we generated by simulation 100 arbitrary scenarios (possible sequences of observations) of 10000 observations each, and used them as input to all four symbolic models. The experiments reported here were run on a 3.2 GHz Pentium IV with 1 Gbyte of memory.

The left graph of Figure 2.17 compares the time performance of the various on-line diagnosis methods. To start with, it is worth noting that symbolic diagnoser methods are slower than enumerative ones. This is due to the fact that the retrieval of the diagnosis information only requires the triggering of a single transition per observation. While BDDs are well suited to efficiently trigger transition sets, the enumerative triggering of a single transition is almost instantaneous and hence faster. For the remaining diagnosis methods, the symbolic implementations are faster than the enumerative ones, owing to the efficient symbolic triggering of transition sets that allows the consideration of all diagnosis candidates at once.

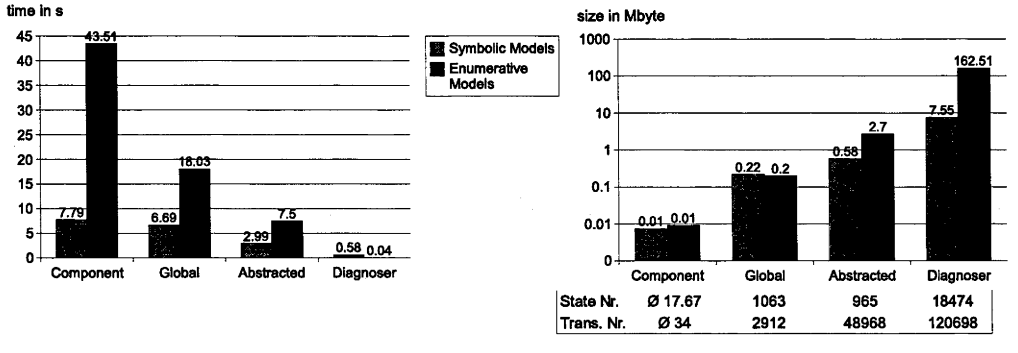


Figure 2.17: Average diagnosis times based on the different models of our example application based on 100 scenarios each of a sequence of 10000 observations on the left and their model sizes on the right.

The differences in symbolic diagnosis times across the spectrum correlate with the extent to which the accumulation of faults (function *AddFault* described on page 46) is performed on-line (see also Figure 2.18). Even though a fault can be simultaneously added to a set of fault labels, *AddFault* still requires the individual consideration of faults since we need to compute for each fault f_i the set of fault labels F_{f_i} to which f_i has to be added (in general, $F_{f_i} \neq F_{f_j}$ for $f_i \neq f_j$). This is the main bottleneck of the symbolic computation. The component and global models yield similar diagnosis times because *AddFault* is applied the same number of times in both cases and symbolic synchronisation is very fast. In contrast, the abstracted model yields a faster diagnosis times because *AddFault* only needs to be applied once per observation, although in that case fault sets rather than single faults need to be added. With the diagnoser, *AddFault* is never called.

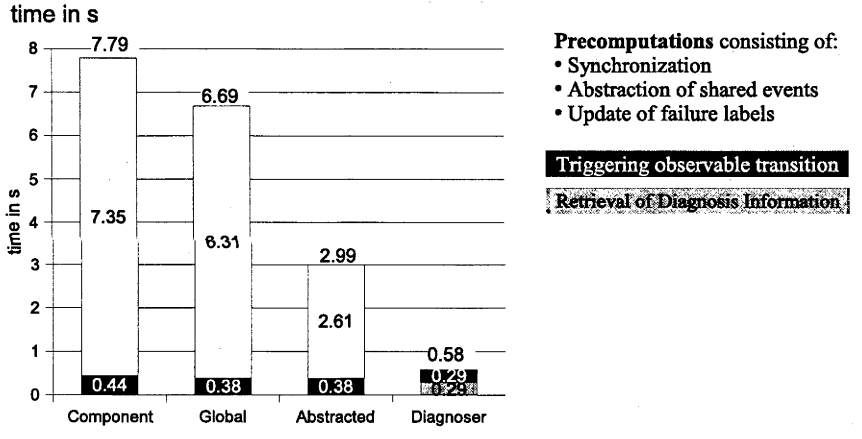


Figure 2.18: Composition of the diagnosis times depicted in Figure 2.17.

Taken in conjunction with the diagnosis times, the corresponding model sizes (see right graph of Figure 2.17) illustrate the time/space tradeoff of the methods across the spectrum and the superiority of the symbolic approach. Comparing the symbolic models (resp. the enumerative ones), we can state, that the faster the on-line diagnosis based on a model, the larger the model size. For all models, the symbolic representation is about the same or smaller than the enumerative one; yet except for the diagnoser, the symbolic run-times are significantly better. Importantly, the symbolic diagnoser is as small as $\frac{1}{20}$ the size of the enumerative one. Its size is rather comparable to that of the enumerative abstracted model, yet it is an order of magnitude faster than the latter.

Note that the representation of the symbolic abstracted model requires almost three times as munch memory than the global model. This results from the fact

that here more BDD nodes are needed to represent the fault events. In fact, for the global model fault events can be encoded with only $\frac{1}{2} \cdot |\Sigma_f + 1|^2$ BDD nodes, because only $|\Sigma_f|$ different fault labels need to be represented (see Figure 2.19). In contrast, for the abstracted model up to $2^{|\Sigma_f|}$ different fault labels might need to be represented which could require $2^{|\Sigma_f|} - 1$ BDD nodes.

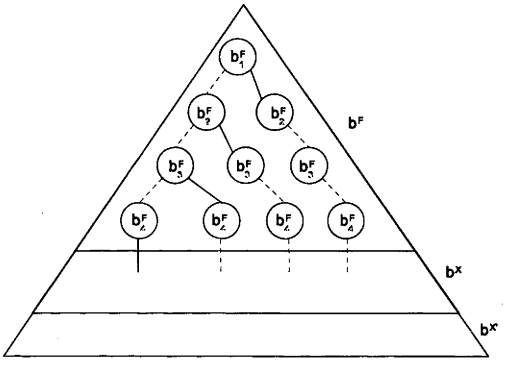


Figure 2.19: Representation of fault events in the global model.

2.4.4 Summary

In this section, we have presented a symbolic direct diagnosis approach consisting of a spectrum of models that differ in the degree to which the global diagnosis information is precomputed. These models range from the small component models that do not incorporate any precomputations, to the diagnoser model in which the diagnosis information for the entire observable behaviour of the system is precomputed. In comparison to an enumerative implementation, the symbolic precomputation of diagnosis information is not only significantly faster but also leads to a considerably smaller diagnoser model.

When used on-line to retrieve the diagnosis information, the symbolic diagnoser only incurs in a small time overhead compared to the enumerative one. Therefore, an enumerative approach is mainly useful for small applications for which the computation of the large diagnoser is feasible. In contrast, the symbolic diagnoser is also computable for larger systems and can be used to efficiently diagnose them.

The abstracted model constitutes an interesting alternative to the diagnoser. This nondeterministic model exploits the property of BDDs of efficiently triggering transition sets at once. The abstracted model is considerably smaller than the diagnoser and can be computed for a wider range of applications.

The component based on-line diagnosis algorithm is a method that requires only

very small space and is applicable to large systems. However, the diagnosis time is significantly higher than when using the abstracted or diagnoser models. The slow diagnosis results from the low efficiency of symbolically updating fault labels. In the next section, we will show that the off-line computation of fault labels leads to dramatic improvements in the on-line performance of the component, global, and abstracted models. With such an approach we aim at diagnosing even larger systems efficiently.

We conclude this section with a note on diagnosing safety-critical applications. Here time and space efficiency of a diagnosis approach might be less important than the robustness of the method. Thus a more rugged enumerative approach might be preferred over one using a BDD package such as CUDD which has a larger size and complexity.

2.5 The Symbolic Compiled Diagnosis Approach

In the previous section, we have shown that the efficiency of symbolic on-line diagnosis largely depends on the extent to which the fault information is already compiled and abstracted from the model. This suggests that we can increase the diagnosis efficiency by abstracting the entire fault information from all the models, and representing it independently of the nominal behaviour.

In this section, we define a spectrum of such compiled diagnosis models and determine the extent to which efficiency increases when the local diagnosis information is precomputed. We also describe how these compiled models can be used to retrieve the diagnosis information on-line and prove that this information is indeed correct. Finally we analyse the performance in time and space of this compiled diagnosis approach in comparison to the direct one.

2.5.1 Spectrum of Compiled Diagnosis Models

Our framework comprises four models. The main difference with the models defined before lies in the local precomputation of diagnosis information. While in the previous approach, we first synchronise the component models before we precompute any diagnosis information, we now swap the two computation steps and first retrieve, off-line, the diagnosis information for each component before we synchronise their behaviours (see Figure 2.20).

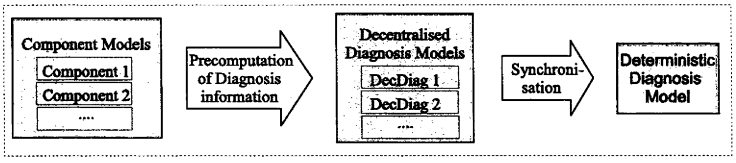


Figure 2.20: Symbolic model computation

The models involved in this approach are shown in Figure 2.21. The decentralised diagnosis models are obtained from the component models by precomputing the diagnosis information locally. The synchronisation of these models results in the centralised diagnosis model, which abstracted from the shared events leads to the nondeterministic diagnosis model. Finally we determinise the latter to obtain the deterministic diagnosis model.

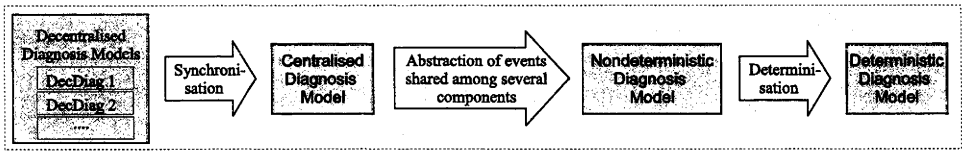


Figure 2.21: Compiled diagnosis models and their relationships.

All these models are best viewed as consisting of two parts: the local diagnosis information which is represented the same way for all the models, and a description of the system behaviour in which the models differ. The latter part determines how efficient the local diagnosis information can be accessed. It is related to the original direct diagnosis models and can be derived from them by abstracting the fault information (for the three models on the left of Figure 2.21) followed by a determinisation (for the two models on the left of the figure). The deterministic diagnosis model is equivalent to the diagnoser. We define it in order to prove that the compiled diagnosis approach is indeed correct (see Section 2.5.3).

Decentralised Diagnosis Model

In the component models of the direct approach introduced earlier, the nominal and fault behaviours of a component are represented as FSMs in an homogeneous way. We now define (Definition 11) a decentralised diagnosis model in which the fault behaviours are abstracted and accounted for in the state labels of a FSM. The latter is a sort of local diagnoser for the component, with a couple of important differences from the notion of diagnoser described earlier for the global model. These differences are motivated by the fact that we will ultimately need to retrieve the global diagnosis information on the basis of the local one:

- We retain the shared events labelling the transitions of the FSM. This is necessary to identify the states of the FSM labelled with a local diagnosis information that is consistent with the global behaviour of the system.
- We use two state labelling functions, namely:
  - the usual or "classical" labelling function $\mathtt{R}_i$ which retrieves the diagnosis information that is consistent with any sequence of events S_i observed by component G_i, and
  - the *quiet* labelling function $\mathtt{R}'_i$ which retrieves the diagnosis information that is consistent with S_i, followed by any sequence of unobservable nonshared events of G_i.

Hence, to determine which system states and faults are possible immediately after the last observation of S_i, we use the classical function. In contrast, to determine the diagnosis information that is consistent with S_i and any future behaviour that does not involve other components and that cannot be observed, we use the quiet labelling function.

Definition 11 (Decentralised diagnosis model) *The decentralised diagnosis model of a component* $G_i = \langle X_i, \Sigma_i, x_{0_i}, T_i \rangle$ *is the deterministic finite state machine* $\mathtt{G}_i = \langle \mathtt{X}_i, \Psi_i, \Sigma'_i, \mathtt{R}_i, \mathtt{R}'_i, \mathtt{x}_{0_i}, \mathtt{T}_i \rangle$ *where*

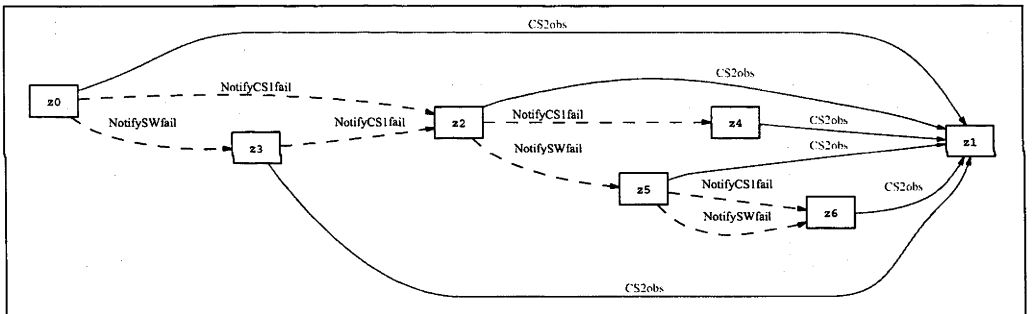
- $\mathtt{X}_i$ *is the set of states* ($\mathtt{X}_i = \{\mathtt{x}_{0_i}, \ldots, \mathtt{x}_{q_i}\}$);
- Ψ_i *is the set of possible local diagnosis candidates* ($\Psi_i = X_i \times 2^{\Sigma_{f_i}}$);
- Σ'_i *is the set of events* ($\Sigma'_i = \Sigma_{s_i} \cup \Sigma_{o_i}$);
- $\mathtt{x}_{0_i}$ *is the initial state;*
- $\mathtt{T}_i \subseteq \mathtt{X}_i \times \Sigma'_i \times \mathtt{X}_i$ *is the set of transitions;*
- $\mathtt{R}_i$ *and* $\mathtt{R}'_i$ *are two state labelling functions that associate a state to its possible local diagnosis information* ($\mathtt{R}_i : \mathtt{X}_i \mapsto 2^{\Psi_i}$ *and* $\mathtt{R}'_i : \mathtt{X}_i \mapsto 2^{\Psi_i}$);
- $\mathtt{R}_i$, $\mathtt{R}'_i$ *and* $\mathtt{T}_i$ *satisfy:*
  $\mathtt{R}_i(\mathtt{x}_{0_i}) = \{(x_{0_i}, \emptyset)\}$ *and*
  $\mathtt{x}_i \xrightarrow{\sigma} \mathtt{x}'_i \in \mathtt{T}_i$ *iff*

$$\mathtt{R}_i(\mathtt{x}'_i) = \{(x'_i, l_i \cup l'_i) \mid \exists (x_i, l_i) \in \mathtt{R}_i(\mathtt{x}_i) \text{ such that } P_{\Sigma''_i} \xrightarrow{\sigma} x'_i \text{ is path in } G_i \text{ with } Start(P_{\Sigma''_i}) = x_i \text{ and } l'_i = EvSet(P_{\Sigma''_i}, \Sigma_{f_i}) \text{ and } \Sigma''_i = \Sigma_i \setminus \Sigma'_i\}.$$

$$\mathtt{R}'_i(\mathtt{x}_i) = \mathtt{R}_i(\mathtt{x}_i) \cup \{(x_j, l_i \cup l'_i) \mid P_{\Sigma''_i} \text{ is path in } G_i \text{ with } \Sigma''_i = \Sigma_i \setminus \Sigma'_i \text{ and } Start(P_{\Sigma''_i}) = x_i,\ Targ(P_{\Sigma''_i}) = x_j,\ l'_i = EvSet(P_{\Sigma''_i}, \Sigma_{f_i}),\ (x_i, l_i) \in \mathtt{R}_i(\mathtt{x}_i)\};$$

Figure 2.22 shows the decentralised diagnosis model for $CS2$. Again, abstracting fault information leads to an increase in model size (see the comparison to the component model of $CS2$ in Figure 2.7). In fact, the number of decentralised diagnosis states is exponential in the number of component states and local fault events. However, since the number of states of a single component is usually small, the computation of $\mathtt{G}_i$ remains feasible.

Note that the size of the decentralised diagnosis model relates to the size of the component model in the same way as the size of the diagnoser relates to that of the global model. This results from the fact that the number of different states in $\mathtt{G}_i$ only depends on the number of different classical state labels. Two decentralised diagnosis states are identical iff their classical labels are identical (see definition of $\mathtt{T}_i$). In that case, the quiet labels are necessarily identical (see definition of $\mathtt{R}'_i$). However two different states can also have identical quiet labels (see for instance states $\mathtt{z}_0$ and $\mathtt{z}_3$ in Figure 2.22).



$$\mathtt{R}_i(\mathtt{z}_0) = \{(z_0, \emptyset)\} \qquad \mathtt{R}'_i(\mathtt{z}_0) = \{(z_0, \emptyset),\ (z_2, \{CS2fail\})\}$$
$$\mathtt{R}_i(\mathtt{z}_1) = \mathtt{R}'_i(\mathtt{z}_1) = \{(z_4, \{CS2fail\})\}$$
$$\mathtt{R}_i(\mathtt{z}_2) = \mathtt{R}'_i(\mathtt{z}_2) = \{(z_1, \emptyset),\ (z_2, \{CS2fail\})\}$$
$$\mathtt{R}_i(\mathtt{z}_3) = \mathtt{R}'_i(\mathtt{z}_3) = \{(z_0, \emptyset),\ (z_2, \{CS2fail\})\}$$
$$\mathtt{R}_i(\mathtt{z}_4) = \mathtt{R}'_i(\mathtt{z}_4) = \{(z_2, \{CS2fail\})\}$$
$$\mathtt{R}_i(\mathtt{z}_5) = \mathtt{R}'_i(\mathtt{z}_5) = \{(z_3, \emptyset),\ (z_2, \{CS2fail\}),\ (z_4, \{CS2fail\})\}$$
$$\mathtt{R}_i(\mathtt{z}_6) = \mathtt{R}'_i(\mathtt{z}_6) = \{(z_2, \{CS2fail\}),\ (z_4, \{CS2fail\})\}$$

Figure 2.22: Decentralised diagnosis model of $CS2$ depicted in Figure 2.7.

Now unsurprisingly, the definition of the symbolic decentralised diagnosis model is similar to that of the symbolic diagnoser.

Definition 12 (Symbolic decentralised diagnosis model) *Given a component* $G_i = \langle b_i^X, b_i^{X'}, b_i^O, b_i^N, b_i^F, b_i^S, X_i, x_{0_i}, T_{o_i}, T_{n_i}, T_{f_i}, T_{s_i} \rangle$, *the symbolic decentralised diagnosis model* $\mathtt{G}_i = \langle b_i^{\mathtt{X}}, b_i^{\mathtt{X}'}, b_i^X, b_i^O, b_i^F, b^S, \Phi_i, \Phi_i', \mathtt{x}_{0_i}, \mathtt{T}_{o_i}, \mathtt{T}_{s_i} \rangle$ *is described via five BDDs* Φ_i, Φ_i', $\mathtt{x}_{0_i}$, $\mathtt{T}_{o_i}$ *and* $\mathtt{T}_{s_i}$ *involving the new Boolean variables* $b_i^{\mathtt{X}}$ *and* $b_i^{\mathtt{X}'}$, *where*

- $b_i^{\mathtt{X}}$ *are the decentralised state variables* ($b_i^{\mathtt{X}} = \{b_{1_i}^{\mathtt{x}}, \ldots, b_{Nr(\mathtt{X}_i)_i}^{\mathtt{x}}\}$),
- $b_i^{\mathtt{X}'}$ *are the primed decentralised state variables* ($b_i^{\mathtt{X}'} = \{b_{1_i}^{\mathtt{x}'}, \ldots, b_{Nr(\mathtt{X}_i)_i}^{\mathtt{x}'}\}$),
- Φ_i *and* Φ_i' *are two Boolean functions over* $b_i^{\mathtt{X}} \cup b_i^X \cup b_i^F$ *encoding the state labelling functions* $\mathtt{R}_i$ *and* $\mathtt{R}_i'$ *for all states,*
- $\mathtt{x}_{0_i}$ *is the Boolean function over* $b_i^{\mathtt{X}}$ *characterising the initial state,*
- $\mathtt{T}_{o_i}$ *is the Boolean function over* $b_i^{\mathtt{X}} \cup b_i^O \cup b_i^{\mathtt{X}'}$ *characterising the observable transition relation, and*
- $\mathtt{T}_{s_i}$ *is the Boolean function over* $b_i^{\mathtt{X}} \cup b^S \cup b_i^{\mathtt{X}'}$ *characterising the shared transition relation.*

Consequently, its computation is very similar to that of the diagnoser, as shown in Algorithm 3 on page 48. Only step 6 of that algorithm, namely the computation of the diagnosis information of all target states starting in $\mathtt{x}_{new_i}$ and of the observations labelling the transitions starting in $\mathtt{x}_{new_i}$, needs to be extended in order to

- compute the quiet state labels Φ_i' and to
- compute both, observable and shared transitions.

Thus, given a new state $\mathtt{x}_{new_i}$, we first compute all component states that can be reached by triggering local unobservable transitions. These states along with their corresponding fault labels, represented by the BDD $reachUnobs_i$, comprise the quiet state label of $\mathtt{x}_{new_i}$. To add this label to Φ_i' we add the conjunction of $\mathtt{x}_{new_i}$ and $reachUnobs_i$ as follows: $\Phi_i' \leftarrow \Phi_i' \vee (\mathtt{x}_{new_i} \wedge reachUnobs_i)$.

Next we trigger all observable and shared transitions starting in states of $reachUnobs_i$. After abstracting the start variables b_i^X and swapping the state variables we continue with the remaining computations similar to steps 7–10 of Algorithm 3.

Centralised Diagnosis Model

Computing the global diagnosis information on the basis of decentralised diagnosis models requires both: a synchronisation of the individual transition BDDs and a synchronisation of the BDDs representing the state labels. We now define these synchronisation rules.

Definition 13 (Centralised diagnosis model) *The centralised diagnosis model of the system is the deterministic finite state machine* $\mathtt{G} = \langle \mathtt{X}, \Psi, \Sigma', \mathtt{R}, \mathtt{x}_0, \psi_0, \mathtt{T} \rangle$. *It is defined as the synchronous composition of the decentralised diagnosis models* $\mathtt{G}_i = \langle \mathtt{X}_i, \Psi_i, \Sigma'_i, \mathtt{R}_i, \mathtt{R}'_i, \mathtt{x}_{0_i}, \mathtt{T}_i \rangle$ *such that*

- $\mathtt{X}$ *is the set of system states* ($\mathtt{X} = \prod_{i=1}^{n} \mathtt{X}_i$);
- Ψ *is the set of possible global diagnosis information* ($\Psi = \prod_{i=1}^{n} \Psi_i$);
- $\Sigma' = \Sigma_o \cup \Sigma_s$ *are the events* ($\Sigma_o = \bigcup_{i=1}^{n} \Sigma_{o_i}$ *and* $\Sigma_s = \bigcup_{i=1}^{n} \Sigma_{s_i}$);
- $\mathtt{R}$ *is the diagnosis labelling function* ($\mathtt{R} : \Sigma_o \times \mathtt{X} \mapsto 2^{\Psi}$). *Let* σ *denote an observation defined in component* G_j *and* $\mathtt{x} = \prod_i \mathtt{x}_i$ *a state in* $\mathtt{G}$. *Then* $\mathtt{R}$ *satisfies:*

$$\mathtt{R}(\sigma, \mathtt{x}) = \mathtt{R}_j(\mathtt{x}_j) \times \prod_{i, i \neq j} \mathtt{R}'_i(\mathtt{x}_i)$$

- $\mathtt{x}_0$ *is the initial state* ($\mathtt{x}_0 = \prod_{i=1}^{n} \mathtt{x}_{0_i}$);
- ψ_0 *is the initial global diagnosis information* ($\psi_0 = \{(x_0, \emptyset)\}$);
- $\mathtt{T}$ *is the transition set* ($\mathtt{T} = \{(\mathtt{x}_1, \ldots, \mathtt{x}_n) \xrightarrow{\sigma} (\mathtt{x}'_1, \ldots, \mathtt{x}'_n) \mid$
  $\forall i \in 1 \ldots n \text{ s.t. } \sigma \in \Sigma'_i, \mathtt{x}_i \xrightarrow{\sigma} \mathtt{x}'_i \in \mathtt{T}_i$ *and*
  $\forall i \in 1 \ldots n \text{ s.t. } \sigma \notin \Sigma'_i, \mathtt{x}_i = \mathtt{x}'_i\}$).

Thus the global diagnosis information is derived as composition of the local diagnosis information. In general, it does not only depend on the global state itself but also on the observation leading to that state, or more precisely on the component G_j that observed the last event. Only for this decentralised diagnosis model $\mathtt{G}_j$ the information is obtained using the classical labelling function $\mathtt{R}_j$. The local diagnosis information of the other components is retrieved via the quiet function $\mathtt{R}'_i$.

Figure 2.23 shows a part of the global diagnosis model for our example application. Consider for instance state $(\mathtt{x}_0, \mathtt{y}_1, \mathtt{z}_0)$ that is only reached from the initial state via the transition labelled $CS1obs$. In case this event is observed the global

diagnosis information is the following:

$$\begin{aligned}
\mathtt{R}(CS1obs, (\mathtt{x}_0, \mathtt{y}_1, \mathtt{z}_0)) &= \mathtt{R}'_{SW}(\mathtt{x}_0) \times \mathtt{R}_{CS1}(\mathtt{y}_1) \times \mathtt{R}'_{CS2}(\mathtt{z}_0) \\
&= \Big\{(x_0, \emptyset),\ (x_1, \{SWfail\})\Big\} \times \Big\{(y_3, \{CS1fail\})\Big\} \times \\
&\quad \Big\{(z_0, \emptyset),\ (z_2, \{CS2fail\})\Big\} \\
&= \Big\{\Big((x_0, y_3, z_0), \{CS1fail\}\Big), \\
&\quad \Big((x_1, y_3, z_0), \{SWfail, CS1fail\}\Big), \\
&\quad \Big((x_0, y_3, z_2), \{CS1fail, CS2fail\}\Big), \\
&\quad \Big((x_1, y_3, z_2), \{SWfail, CS1fail, CS2fail\}\Big)\Big\}
\end{aligned}$$

Symbolically, we can very efficiently retrieve the global diagnosis information using simple $\wedge$ operations. Therefore, computing this information on-line hardly adds to the diagnosis time. Consequently, to decrease the storage space, we choose to only synchronise the transition BDDs and not the state labelling functions.

Definition 14 (Symbolic centralised diagnosis model)
Given a set of n symbolic decentralised diagnosis models $\mathtt{G}_i = \langle b_i^{\mathtt{X}}, b_i^{\mathtt{X}'}, b_i^X, b_i^O, b_i^F, b^S, \Phi_i, \Phi'_i, \mathtt{x}_{0_i}, \mathtt{T}_{o_i}, \mathtt{T}_{s_i}\rangle$, the symbolic centralised diagnosis model $\mathtt{G} = \langle b^{\mathtt{X}}, b^{\mathtt{X}'}, b^X, b^O, b^S, b^F, \Phi_1, \ldots, \Phi_n, \Phi'_1, \ldots, \Phi'_n, \mathtt{x}_0, \mathtt{T}_o, \mathtt{T}_s\rangle$ is described using the $2n$ BDDs $\Phi_1, \ldots, \Phi_n, \Phi'_1, \ldots, \Phi'_n$, and the 3 BDDs $\mathtt{x}_0$, $\mathtt{T}_o$ and $\mathtt{T}_s$, involving the Boolean variables $b^{\mathtt{X}}$ and $b^{\mathtt{X}'}$, where

- *$b^{\mathtt{X}}$ are the state variables ($b^{\mathtt{X}} = \cup_{i=1}^n b_i^{\mathtt{X}}$);*
- *$b^{\mathtt{X}'}$ are the primed state variables ($b^{\mathtt{X}'} = \cup_{i=1}^n b_i^{\mathtt{X}'}$);*
- *$\mathtt{x}_0$ is the Boolean function over $b^{\mathtt{X}}$ characterising the initial state ($\mathtt{x}_0 = \wedge_{i=1}^n \mathtt{x}_{0_i}$),*
- *$\mathtt{T}_o$ is the Boolean function over $b^{\mathtt{X}} \cup b^O \cup b^{\mathtt{X}'}$ characterising the observable transition relation, and*
- *$\mathtt{T}_s$ is the Boolean function over $b^{\mathtt{X}} \cup b^S \cup b^{\mathtt{X}'}$ characterising the transition relation of shared events.*

The synchronisation of the transitions of the decentralised diagnosis models is analogous to the synchronisation of the component transitions. Hence the computation of the global diagnosis model is similar to that of the global model (see Algorithm 2 on page 43). The global diagnosis information can now be retrieved by triggering the transitions of a single symbolic model and synchronising the local diagnosis information (see Section 2.5.4).

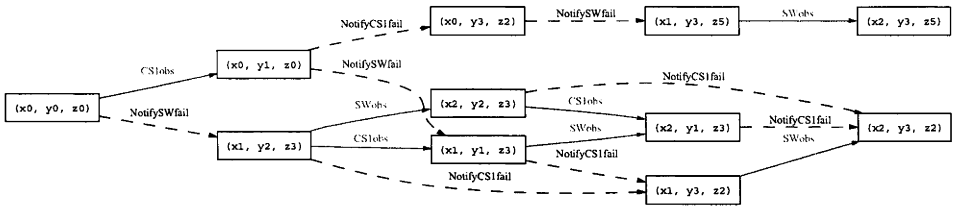


Figure 2.23: (Part of) centralised diagnosis model for the component models depicted in Figure 2.7.

Nondeterministic Diagnosis Model

The centralised diagnosis model still includes the shared events. However, after the synchronisation, these events are irrelevant and can therefore be abstracted to allow a more efficient computation of the global diagnosis information. This leads us to the nondeterministic diagnosis model.

Definition 15 (Nondeterministic diagnosis model) *Given a centralised diagnosis model* $\mathtt{G} = \langle \mathtt{X}, \Psi, \Sigma', \mathtt{R}, \mathtt{x}_0, \psi_0, \mathtt{T} \rangle$, *the nondeterministic diagnosis model is the finite state machine* $\widetilde{\mathtt{G}} = \langle \widetilde{\mathtt{X}}, \Psi, \Sigma_o, \mathtt{R}, \mathtt{x}_0, \psi_0, \widetilde{\mathtt{T}} \rangle$, *where:*

- $\widetilde{\mathtt{X}} \subseteq \mathtt{X}$ *is the set of states*
  $(\widetilde{\mathtt{X}} = \{\mathtt{x}_0\} \cup \{\mathtt{x} \in \mathtt{X} \mid \mathtt{x}$ *is a target state of an observable transition*$\})$;

- $\widetilde{\mathtt{T}}$ *is the transition set* $(\widetilde{\mathtt{T}} = \{\widetilde{\mathtt{x}} \xrightarrow{\sigma} \widetilde{\mathtt{x}}' \mid \sigma \in \Sigma_o$ *and* $P_{\Sigma_u} \xrightarrow{\sigma} \widetilde{\mathtt{x}}'$ *is a path in* $\mathtt{G}$ *with* $Start(P_{\Sigma_u}) = \widetilde{\mathtt{x}}\})$.

Figure 2.24 shows a part of the nondeterministic diagnosis model for our running example. Since the model's states are a subset of the centralised model's states, we can retrieve the global diagnosis information using the same labelling function $\mathtt{R}$ as we defined for the centralised model. The symbolic definition of the nondeterministic diagnosis model is given below.

Definition 16 (Symbolic nondeterministic diagnosis model) *Given the symbolic centralised model*
$\mathtt{G} = \langle b^{\mathtt{X}}, b^{\mathtt{X}'}, b^{X}, b^{O}, b^{S}, b^{F}, \Phi_1, \ldots, \Phi_n, \Phi'_1, \ldots, \Phi'_n, \mathtt{x}_0, \mathtt{T}_o, \mathtt{T}_s \rangle$,
the symbolic nondeterministic diagnosis model
$\widetilde{\mathtt{G}} = \langle b^{\mathtt{X}}, b^{\mathtt{X}'}, b^{X}, b^{O}, b^{F}, \Phi_1, \ldots, \Phi_n, \Phi'_1, \ldots, \Phi'_n, \mathtt{x}_0, \widetilde{\mathtt{T}} \rangle$ *is described via* $2n + 2$ *BDDs:* $\Phi_1, \ldots, \Phi_n, \Phi'_1, \ldots, \Phi'_n$, $\mathtt{x}_0$ *and* $\widetilde{\mathtt{T}}$ *where*

- $\widetilde{\mathtt{T}}$ *is the Boolean function over* $b^{\mathtt{X}} \cup b^{O} \cup b^{\mathtt{X}'}$ *characterising the transition relation.*

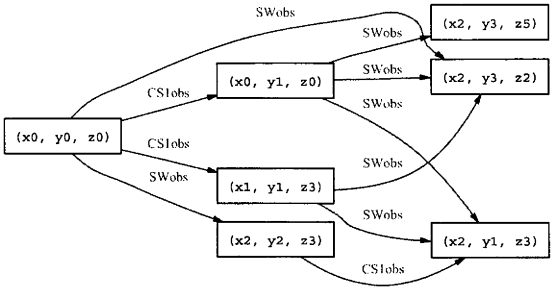


Figure 2.24: (Part of) nondeterministic diagnosis model for the component models depicted in Figure 2.7.

To compute the model's transitions, we first retrieve the state set $\widetilde{\mathtt{X}}$ by extracting the targets of the observable centralised diagnosis transitions and adding the initial state: $\widetilde{\mathtt{X}} \leftarrow \mathit{SwapVar}(\mathit{ExtractVar}(\mathtt{T}_o, b^{\mathtt{X}'}), b^{\mathtt{X}}, b^{\mathtt{X}'}) \vee \mathtt{x}_0$.

Then we compute for each state $\widetilde{\mathtt{x}} \in \widetilde{\mathtt{X}}$ the targets $\widetilde{\mathtt{X}}_{targ}$ of all shared event paths starting in $\widetilde{\mathtt{x}}$ using function *CompReach* described on page 27. In contrast to the abstracted model computation this function requires no modification since the fault information is represented separately. The target states are thus obtained as: $\widetilde{\mathtt{X}}_{targ} \leftarrow \mathit{CompReach}(b^S, b^{\mathtt{X}}, b^{\mathtt{X}'}, \widetilde{\mathtt{x}}, \mathtt{T}_s)$. To compute the transitions $\widetilde{\mathtt{T}}_{\widetilde{\mathtt{x}}} \subseteq \widetilde{\mathtt{T}}$ starting in $\widetilde{\mathtt{x}}$ we need to look at all observable transitions $\mathtt{T}_o$ of the centralised diagnosis model that start in one of the states in $\widetilde{\mathtt{X}}_{targ}$. In the nondeterministic model they now also start in $\widetilde{\mathtt{x}}$. Hence $\widetilde{\mathtt{T}}_{\widetilde{\mathtt{x}}}$ is obtained as: $\widetilde{\mathtt{x}} \wedge \mathit{AbstractVar}(\mathtt{T}_o \wedge \widetilde{\mathtt{X}}_{targ}, b^{\mathtt{X}})$.

With the nondeterministic diagnosis model, we have provided an efficient basis for computing the global states whose labels point to the diagnosis information. The retrieval of these states requires only as many triggerings of transition sets as there are observations in the sequence S.

Deterministic Diagnosis Model

In order to show that the compiled diagnosis models can be used to retrieve the correct diagnosis information (see Section 2.5.3), we now define the deterministic diagnosis model, which is equivalent to the diagnoser. The model $\hat{\mathtt{G}}$ is the determinisation of the nondeterministic diagnosis model $\widetilde{\mathtt{G}}$. Therefore, each of its states $\hat{\mathtt{x}}$ corresponds to a set of states $\Upsilon(\hat{\mathtt{x}}) \subseteq \widetilde{\mathtt{X}}$ in the nondeterministic diagnosis model.

Definition 17 (Deterministic diagnosis model) *Based on the nondeterministic diagnosis model* $\widetilde{\mathtt{G}} = \langle \widetilde{\mathtt{X}}, \Psi, \Sigma_o, \mathtt{R}, \mathtt{x}_0, \psi_0, \widetilde{\mathtt{T}} \rangle$, *the deterministic diagnosis model of the system is defined as the finite state machine* $\hat{\mathtt{G}} = \langle \hat{\mathtt{X}}, \Psi, \Sigma_o, \Upsilon, \hat{\mathtt{R}}, \hat{\mathtt{x}}_0, \psi_0, \hat{\mathtt{T}} \rangle$, *where:*

- $\hat{\mathtt{X}}$ *is the set of states* ($\hat{\mathtt{X}} = \{\hat{\mathtt{x}}_0, \ldots, \hat{\mathtt{x}}_q\}$);
- $\hat{\mathtt{x}}_0$ *is the initial state;*
- Υ *is the state labelling function* ($\Upsilon : \hat{\mathtt{X}} \mapsto 2^{\widetilde{\mathtt{X}}}$);
- $\hat{\mathtt{R}}$ *is the diagnosis labelling function* ($\hat{\mathtt{R}} : \Sigma_o \times \hat{\mathtt{X}} \mapsto 2^{\Psi}$);
- $\hat{\mathtt{T}} \subseteq \hat{\mathtt{X}} \times \Sigma_o \times \hat{\mathtt{X}}$ *is the set of transitions;*
- Υ, $\hat{\mathtt{R}}$, *and* $\hat{\mathtt{T}}$ *satisfy:*
  $\Upsilon(\hat{\mathtt{x}}_0) = \{\mathtt{x}_0\}$;
  $\hat{\mathtt{R}}(\sigma, \hat{\mathtt{x}}) = \bigcup_{\mathtt{x} \in \Upsilon(\hat{\mathtt{x}})} \mathtt{R}(\sigma, \mathtt{x})$; *and*
  $\hat{\mathtt{x}} \xrightarrow{\sigma} \hat{\mathtt{x}}' \in \hat{\mathtt{T}}$ *iff* $\Upsilon(\hat{\mathtt{x}}') = \{\widetilde{\mathtt{x}}' \mid \exists \widetilde{\mathtt{x}} \in \Upsilon(\hat{\mathtt{x}})$ *such that* $(\widetilde{\mathtt{x}} \xrightarrow{\sigma} \widetilde{\mathtt{x}}') \in \widetilde{\mathtt{T}}\}$.

Figure 2.25 shows the deterministic diagnosis model for our example application. In the graph, states are only labelled with the global states returned by function Υ. On the basis of these states, we can also retrieve the corresponding diagnosis information. For instance, after the event $CS1obs$ was observed in the initial state the diagnosis information is computed as follows:

$$
\begin{aligned}
\hat{\mathtt{R}}(CS1obs, \hat{\mathtt{x}}_2)) &= \mathtt{R}(CS1obs, (\mathtt{x}_0, \mathtt{y}_1, \mathtt{z}_0)) \cup \mathtt{R}(CS1obs, (\mathtt{x}_1, \mathtt{y}_1, \mathtt{z}_3)) \\
&= (\mathtt{R}'_{SW}(\mathtt{x}_0) \times \mathtt{R}_{CS1}(\mathtt{y}_1) \times \mathtt{R}'_{CS2}(\mathtt{z}_0)) \cup \\
&\quad (\mathtt{R}'_{SW}(\mathtt{x}_1) \times \mathtt{R}_{CS1}(\mathtt{y}_1) \times \mathtt{R}'_{CS2}(\mathtt{z}_3)) \\
&= \Big(\Big\{(x_0, \emptyset),\ (x_1, \{SWfail\})\Big\} \times \Big\{(y_3, \{CS1fail\})\Big\} \times \\
&\qquad \Big\{(z_0, \emptyset),\ (z_2, \{CS2fail\})\Big\}\Big) \cup \\
&\quad \Big(\Big\{(x_2, \{SWfail\})\Big\} \times \Big\{(y_3, \{CS1fail\})\Big\} \times \\
&\qquad \Big\{(z_0, \emptyset),\ (z_2, \{CS2fail\})\Big\}\Big) \\
&= \Big\{(x_0, \emptyset),\ (x_1, \{SWfail\}), (x_2, \{SWfail\})\Big\} \times \\
&\quad \Big\{(y_3, \{CS1fail\})\Big\} \times \Big\{(z_0, \emptyset),\ (z_2, \{CS2fail\})\Big\}
\end{aligned}
$$

The symbolic deterministic diagnosis model, which is defined below, does not represent the labelling function $\hat{\mathtt{R}}$ explicitly, since it can be efficiently retrieved based on Γ and the state labels of the decentralised diagnosis models.

Definition 18 (Symbolic deterministic diagnosis model)
Given the nondeterministic diagnosis model $\widetilde{\mathtt{G}} = \langle b^{\mathtt{X}}, b^{\mathtt{X}'}, b^{X}, b^{O}, b^{F}, \Phi_1, \ldots, \Phi_n, \Phi'_1, \ldots, \Phi'_n, \mathtt{x}_0, \widetilde{\mathtt{T}} \rangle$, *the symbolic deterministic diagnosis model* $\hat{\mathtt{G}} = \langle b^{\hat{\mathtt{X}}}, b^{\hat{\mathtt{X}}'}, b^{\mathtt{X}}, b^{X}, b^{O}, b^{F}, \Phi_1, \ldots, \Phi_n, \Phi'_1, \ldots, \Phi'_n, \Gamma, \hat{\mathtt{x}}_0, \hat{\mathtt{T}} \rangle$ *is described using the* $2n$ *BDDs* $\Phi_1, \ldots, \Phi_n, \Phi'_1, \ldots, \Phi'_n$, *and the 3 BDDs* Γ, $\hat{\mathtt{x}}_0$ *and* $\hat{\mathtt{T}}$, *involving the Boolean variables* $b^{\hat{\mathtt{X}}}$ *and* $b^{\hat{\mathtt{X}}'}$, *where*

- $b^{\hat{\mathtt{X}}}$ *are the model's state variables* ($b^{\hat{\mathtt{X}}} = \{b^{\hat{\mathtt{x}}}_1, \ldots, b^{\hat{\mathtt{x}}}_{Nr(\hat{\mathtt{X}})}\}$);
- $b^{\hat{\mathtt{X}}'}$ *are the model's primed state variables* ($b^{\hat{\mathtt{X}}'} = \{b^{\hat{\mathtt{x}}'}_1, \ldots, b^{\hat{\mathtt{x}}'}_{Nr(\hat{\mathtt{X}})}\}$);
- Γ *is the Boolean function over* $b^{\hat{\mathtt{X}}} \cup b^{\mathtt{X}}$ *characterising the model's state labels;*
- $\hat{\mathtt{x}}_0$ *is the Boolean function over* $b^{\hat{\mathtt{X}}}$ *characterising the initial state; and*
- $\hat{\mathtt{T}}$ *is the Boolean function over* $b^{\hat{\mathtt{X}}} \cup b^{O} \cup b^{\hat{\mathtt{X}}'}$ *characterising the transition relation.*

The symbolic computation of this model is very similar to the classical diagnoser computation shown in Algorithm 3 on page 48. However, the complexity of the algorithm is significantly reduced. To retrieve the labels of all target states of a new state, we only need to apply a single Boolean operation: $reachObs \leftarrow \hat{\mathtt{x}}_{newInfo} \wedge \widetilde{\mathtt{T}}$. Recall that in the original approach we first had to trigger unobservable transitions and handle the aggregation of fault labels before observable transitions were considered (see step 6).

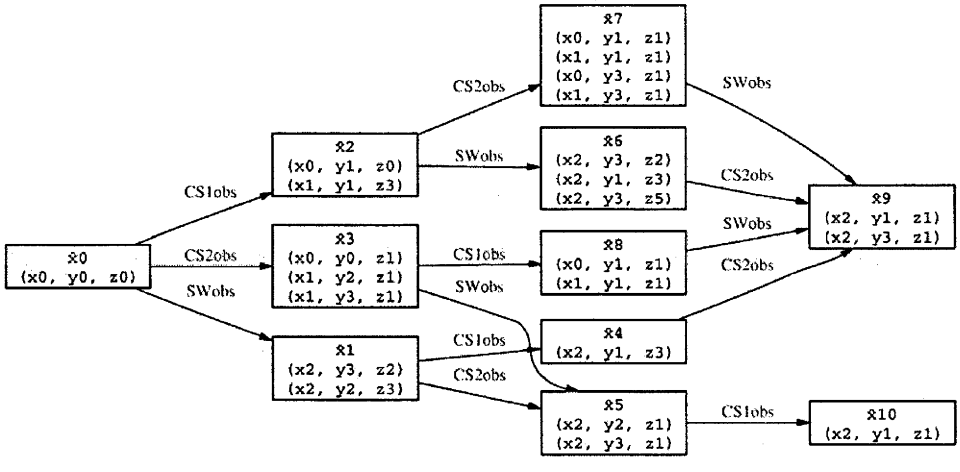


Figure 2.25: Deterministic diagnosis model for the component models in Figure 2.7.

2.5.2 Experimental Comparison of Model Sizes

We have defined four symbolic models that each consist of two parts: the local diagnosis information and a description of the system behaviour from which the fault information is abstracted. Now, we experimentally compare the sizes of these different compiled diagnosis models for our example application. We also analyse the extent to which the abstraction of fault information and its decentralised representation affects the size of the behavioural models, by comparing each compiled model with its analogue in the direct approach (i.e., we compare the size of the component models with that of the decentralised diagnosis models, the size of the global model with that of the centralised diagnosis model, etc).

The right chart of Figure 2.26 shows the number of states and transitions of the various compiled diagnosis models, together with the size required for their symbolic representation. The smallest of them is the decentralised model whose size is about 50 KB, 30 of which are used to store the transitions and 20 of which are required to represent the state labels. Since the diagnosis information is represented in a decentralised way in all compiled models, they all use 20 KB for storing this information, which is negligible considering the total model sizes. Note that a global representation of the labels R would require 0.32 MB for this example, that is 16 times more.

In comparison to the direct system description as a set of components, which requires less than 10 KB of memory (see chart on the left), the decentralised diagnosis model is larger. As mentioned earlier, its number of states is exponential in the number of states and faults of the corresponding component. However, since its size increases only linearly in the number of components, its computation is feasible even for large scale systems.

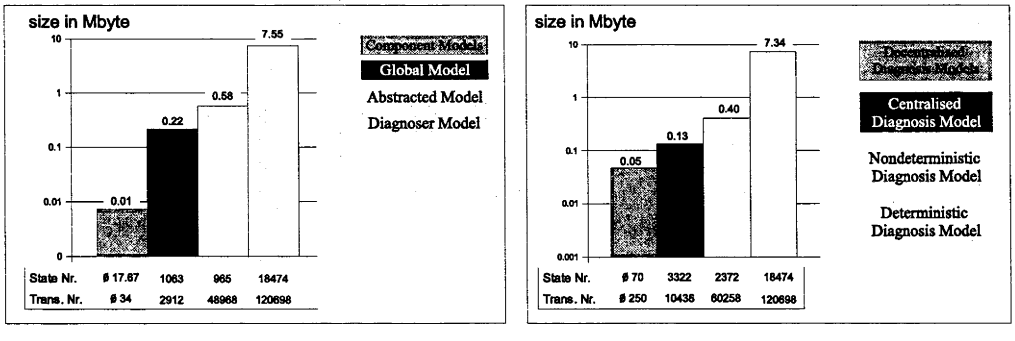


Figure 2.26: Sizes of the direct (on the left) and compiled (on the right) diagnosis models for the example application

Comparing the global model with the centralised one and the abstracted model with the nondeterministic diagnosis model, we can state that, even though they contain more states, the symbolic models of the compiled diagnosis approach are smaller. Once again, we see that BDDs are very suitable to represent large sets of data, where *large* corresponds to the set size in relation to the number of Boolean variables needed to represent the sets. Recall that, in contrast to the direct diagnosis approach, the behavioural models of the compiled diagnosis approach do not require any variables to represent faults. From an enumerative point of view, we can state that the increase of the number of transitions from the centralised to the nondeterministic diagnosis model is smaller than from the global to the abstracted model. Again this results from the fact that the models of the compiled diagnosis approach do not contain any fault transitions.

The deterministic diagnosis model has almost the same size as the diagnoser model. Hence, the decentralised representation of diagnosis information did not lead to a reduced size. This is due to the fact that the retrieval of the global diagnosis information based on this model requires the additional representation of the state labelling function Γ.

Finally, we observe that size varies significantly from one model to another. Therefore, as in the direct approach, the space complexity of on-line diagnosis algorithms will differ considerably depending on the model they are based on. In Section 2.5.4, we will focus on the time complexity of these algorithms, to complete our assessment of their overall efficiency. This will enable us to identify, based on the time and space requirements of an application, the diagnosis approach that is best suited.

2.5.3 Correctness of Compiled Diagnosis Approach

We have defined several diagnosis models that can all be used for the on-line diagnosis (see Section 2.5.4). Before we compare these diagnosis algorithms, it is necessary to show that they compute indeed the correct diagnosis information.

We therefore prove the equivalence of diagnoser and deterministic diagnosis model. Since all diagnoser states are labelled with the correct diagnosis information, this will guarantee the correctness of the diagnosis information retrieved from the deterministic diagnosis model. This also holds if this model is computed only partially, and in particular, if it is only computed for the event sequence actually observed as it is the case for the on-line diagnosis algorithms based on the remaining

three models of the compiled approach.

Theorem 2.5.1 *There exists a path* $\hat{p} = \hat{x}_0 \xrightarrow{o_1} \hat{x}_1 \xrightarrow{o_2} \hat{x}_2 \cdots \xrightarrow{o_k} \hat{x}_k$ *in the diagnoser model* $\hat{G} = \langle \hat{X}, \Psi, \Sigma_o, \hat{x}_0, \hat{R}, \hat{T} \rangle$ *iff there exists a path* $\hat{\mathrm{p}} = \hat{\mathrm{x}}_0 \xrightarrow{o_1} \hat{\mathrm{x}}_1 \xrightarrow{o_2} \hat{\mathrm{x}}_2 \cdots \xrightarrow{o_k} \hat{\mathrm{x}}_k$ *in the deterministic diagnosis model* $\hat{\mathrm{G}} = \langle \hat{\mathrm{X}}, \Psi, \Sigma_o, \Upsilon, \hat{\mathrm{R}}, \hat{\mathrm{x}}_0, \psi_0, \hat{\mathrm{T}} \rangle$ *with* $\hat{R}(\hat{x}_0) = \psi_0$ *and* $\hat{R}(\hat{x}_h) = \hat{\mathrm{R}}(o_h, \hat{\mathrm{x}}_h)\ \forall h = \{1, \ldots, k\}$.

We prove the theorem by induction over the number of observations k.
Base case: k = 0:

$$\begin{aligned} \hat{R}(\hat{x}_0) &= \{(x_0, \emptyset)\} \\ &= \psi_0 \end{aligned}$$

Induction hypothesis:
$\exists \hat{p} = \hat{x}_0 \xrightarrow{o_1} \hat{x}_1 \xrightarrow{o_2} \hat{x}_2 \cdots \xrightarrow{o_k} \hat{x}_k$ in $\hat{G} \Leftrightarrow$
$\exists \hat{\mathrm{p}} = \hat{\mathrm{x}}_0 \xrightarrow{o_1} \hat{\mathrm{x}}_1 \xrightarrow{o_2} \hat{\mathrm{x}}_2 \cdots \xrightarrow{o_k} \hat{\mathrm{x}}_k$ in $\hat{\mathrm{G}}$
with $\hat{R}(\hat{x}_0) = \psi_0$ and $\hat{R}(\hat{x}_h) = \hat{\mathrm{R}}(o_h, \hat{\mathrm{x}}_h)\ \forall h = \{1, \ldots, k\}$.

Induction proposition:
$\exists \hat{p} = \hat{x}_0 \xrightarrow{o_1} \hat{x}_1 \xrightarrow{o_2} \hat{x}_2 \cdots \xrightarrow{o_k} \hat{x}_k \xrightarrow{o_{k+1}} \hat{x}_{k+1}$ in $\hat{G} \Leftrightarrow$
$\exists \hat{\mathrm{p}} = \hat{\mathrm{x}}_0 \xrightarrow{o_1} \hat{\mathrm{x}}_1 \xrightarrow{o_2} \hat{\mathrm{x}}_2 \cdots \xrightarrow{o_k} \hat{\mathrm{x}}_k \xrightarrow{o_{k+1}} \hat{\mathrm{x}}_{k+1}$ in $\hat{\mathrm{G}}$
with $\hat{R}(\hat{x}_{k+1}) = \hat{\mathrm{R}}(o_{k+1}, \hat{\mathrm{x}}_{k+1})$.

Induction proof:
We prove the proposition in two steps. First we show that each diagnosis entry $(x', l') \in \hat{R}(\hat{x}_{k+1})$ is also an element of $\hat{\mathrm{R}}(o_{k+1}, \hat{\mathrm{x}}_{k+1})$ and second we show that each diagnosis entry $(x', l') \in \hat{\mathrm{R}}(o_{k+1}, \hat{\mathrm{x}}_{k+1})$ is also an element of $\hat{R}(\hat{x}_{k+1})$.

1. $\hat{R}(\hat{x}_{k+1}) \subseteq \hat{\mathrm{R}}(o_{k+1}, \hat{\mathrm{x}}_{k+1})$

Let $x' = \prod_{i=1}^{n} x'_i$ denote a global state and $l' = \bigcup_{i=1}^{n} l'_i$ a fault label such that $(x', l') \in \hat{R}(\hat{x}_{k+1})$

$\Rightarrow$ There exists a diagnosis entry $(x, l) \in \hat{R}(\hat{x}_k)$ such that
$p = P_{\Sigma_u} \xrightarrow{o_{k+1}} x'$ is a path in G (see Definition 6 on page 36) with (1)
- $l' = l \cup EvSet(P_{\Sigma_u}, \Sigma_f)$,
- $x = \prod_{i=1}^{n} x_i = Start(P_{\Sigma_u})$ and
  l be composed of $l = \prod_{i=1}^{n} l_i$.

$\Rightarrow (x, l) \in \hat{\mathtt{R}}(o_k, \hat{\mathtt{x}}_k)$ (see induction hypothesis)

$\Rightarrow$ Since $\hat{\mathtt{R}}(o_k, \hat{\mathtt{x}}_k) = \bigcup_{\mathtt{x} \in \Upsilon(\hat{\mathtt{x}}_k)} \mathtt{R}(o_k, \mathtt{x})$, there exists a state
$\mathtt{x} = \prod_i \mathtt{x}_i \in \Upsilon(\hat{\mathtt{x}}_k)$ with $(x, l) \in \mathtt{R}(o_k, \mathtt{x})$ (see Definition 17) (2)

$\Rightarrow$ There exists a component q such that $o_k \in \Sigma_{o_q}$ and
$\mathtt{R}(o_k, \mathtt{x}) = \mathtt{R}_q(\mathtt{x}_q) \times \prod_{i \neq q} \mathtt{R}'_i(\mathtt{x}_i)$ (see Definition 13) (3)

$\Rightarrow (x_q, l_q) \in \mathtt{R}_q(\mathtt{x}_q)$ and $(x_i, l_i) \in \mathtt{R}'_i(\mathtt{x}_i) \quad \forall i \neq q$

$\Rightarrow$ Since $(x, l) \in \hat{R}(\hat{x}_k)$ and path $p = P_{\Sigma_u} \xrightarrow{o_{k+1}} x'$ is in G (see (1)) it follows from Definition 2 that

- there exists a set of path $P_{\Sigma_{\bar{o}}} = \{P_{\Sigma_{\bar{o}_1}}, \ldots, P_{\Sigma_{\bar{o}_n}}\}$ in the components $G_1, \ldots, G_n$ with $\Sigma_{\bar{o}_i} = \Sigma_i \setminus \Sigma_{o_i}$ such that
  - $l' = l \cup \bigcup_{i=1}^{n} EvSet(P_{\Sigma_{\bar{o}_i}}, \Sigma_{f_i})$
  - $x_i = Start(P_{\Sigma_{\bar{o}_i}})$ and
  - Let $x'' = \prod_{i=1}^{n} x''_i$ denote the target state of P_{Σ_u}, then
    $$x''_i = Targ(P_{\Sigma_{\bar{o}_i}}) \quad \forall i = \{1, \ldots, n\}$$
- there exists a component G_j such that
  - $o_{k+1} \in \Sigma_{o_j}$ and
  - $(x''_j \xrightarrow{o_{k+1}} x'_j) \in T_j$

  and $x'_i = x''_i \quad \forall i \neq j$ (4)

$\Rightarrow$ From the above and from Definition 11 it follows that $\forall i \in \{1, \ldots, n\}$

- the path $P_{\Sigma_{\bar{o}_i}} = P^1_{\Sigma''_i} \xrightarrow{s_i^2} P^2_{\Sigma''_i} \cdots \xrightarrow{s_i^{m_i}} P^{m_i}_{\Sigma''_i}$ in G_i with
  $\Sigma''_i = \Sigma_{\bar{o}_i} \setminus \Sigma_{s_i}$ and $s_i^h \in \Sigma_{s_i}$ $\forall h = \{2, \ldots, m_i\}$
  corresponds to a path
  $\mathtt{P}_{\Sigma_{s_i}} = \mathtt{x}_i \xrightarrow{s_i^2} \mathtt{x}_i^2 \cdots \xrightarrow{s_i^{m_i}} \mathtt{x}_i^{m_i}$ in $\mathtt{G}_i$ such that
  - $(x''_i, l'_i) \in \mathtt{R}'_i(\mathtt{x}_i^{m_i})$ with (5)
    $$l'_i = l_i \cup \bigcup_{h=1}^{m_i} EvSet(P^h_{\Sigma''_i}, \Sigma_{f_i}) \quad \text{(since (3))}$$
    $$= l_i \cup EvSet(P_{\Sigma_{\bar{o}_i}}, \Sigma_{f_i})$$
- there exists a state $\mathtt{x}'_j \in \mathtt{X}_j$ such that
  $(\mathtt{x}_j^{m_j} \xrightarrow{o_{k+1}} \mathtt{x}'_j) \in \mathtt{T}_j$ with $(x'_j, l'_j) \in \mathtt{R}_j(\mathtt{x}'_j)$ (6)

$\Rightarrow$ There exists a path $\mathtt{p} = \mathtt{P}_{\Sigma_s} \xrightarrow{o_{k+1}} \mathtt{x}'$ in $\mathtt{G}$ corresponding to path $p = P_{\Sigma_u} \xrightarrow{o_{k+1}} x'$ in G (see (1)) with (see Definition 13)

- $EvSeq(\mathtt{P}_{\Sigma_s}, \Sigma_s) = EvSeq(P_{\Sigma_u}, \Sigma_s)$
- $\mathtt{x} = Start(\mathtt{P}_{\Sigma_s}) = \prod_i Start(\mathtt{P}_{\Sigma_{s_i}})$ (state $\mathtt{x}$ as defined in (2))
- $Targ(\mathtt{P}_{\Sigma_s}) = \prod_i Targ(\mathtt{P}_{\Sigma_{s_i}})$ and
- $\mathtt{x}' = \mathtt{x}'_j \times \prod_{i,i\neq j} Targ(\mathtt{P}_{\Sigma_{s_i}})$ and hence
  $\prod_i (x'_i, l'_i) = (x', l') \in \mathtt{R}(o_{k+1}, \mathtt{x}')$ (see (4), (5) and (6))

$\Rightarrow$ $\mathtt{x} \xrightarrow{o_{k+1}} \mathtt{x}' \in \widetilde{\mathtt{T}}$ (see Definition 15)

$\Rightarrow$ From the above and since $\mathtt{x} \in \Upsilon(\hat{\mathtt{x}}_k)$ (see (2)) it follows that $\hat{\mathtt{x}}_k \xrightarrow{o_{k+1}} \hat{\mathtt{x}}_{k+1} \in \hat{\mathtt{T}}$ and $\mathtt{x}' \in \Upsilon(\hat{\mathtt{x}}_{k+1})$ (see Definition 17)

$\Rightarrow$ $\mathtt{R}(o_{k+1}, \mathtt{x}') \subseteq \hat{\mathtt{R}}(o_{k+1}, \hat{\mathtt{x}}_{k+1})$ (see Definition 17)

$\Rightarrow$ $(x', l') \in \hat{\mathtt{R}}(o_{k+1}, \hat{\mathtt{x}}_{k+1})$

$\Rightarrow$ $\hat{R}(\hat{x}_{k+1}) \subseteq \hat{\mathtt{R}}(o_{k+1}, \hat{\mathtt{x}}_{k+1})$ $\square$

2. $\hat{R}(\hat{x}_{k+1}) \supseteq \hat{\mathtt{R}}(o_{k+1}, \hat{\mathtt{x}}_{k+1})$

Let $x' = \prod_{i=1}^{n} x'_i$ denote a state and $l' = \bigcup_{i=1}^{n} l'_i$ a fault label such that $(x', l') \in \hat{\mathtt{R}}(o_{k+1}, \hat{\mathtt{x}}_{k+1})$.

$\Rightarrow$ $\exists \mathtt{x}' = \prod_{i=1}^{n} \mathtt{x}'_i \in \Upsilon(\hat{\mathtt{x}}_{k+1})$ such that $(x', l') \in \mathtt{R}(o_{k+1}, \mathtt{x}')$ (see Definition 17)

$\Rightarrow$ $\exists \mathtt{x} = \prod_{i=1}^{n} \mathtt{x}_i \in \Upsilon(\hat{\mathtt{x}}_k)$ such that $(\mathtt{x} \xrightarrow{o_{k+1}} \mathtt{x}') \in \widetilde{\mathtt{T}}$ (see Definition 17) (7)

$\Rightarrow$ there exists a path $\mathtt{p} = \mathtt{P}_{\Sigma_s} \xrightarrow{o_{k+1}} \mathtt{x}'$ in $\mathtt{G}$ such that (see Definition 15)
$\mathtt{x} = Start(\mathtt{P}_{\Sigma_s})$
Let further $\mathtt{x}'' = \prod_{i=1}^{n} \mathtt{x}''_i$ denote the target state of $\mathtt{P}_{\Sigma_s}$

$\Rightarrow$ there exists

- a set of paths $\mathtt{P} = \{\mathtt{P}_{\Sigma_{s_1}}, \ldots, \mathtt{P}_{\Sigma_{s_n}}\}$ in $\mathtt{G}_1, \ldots, \mathtt{G}_n$ (see Definition 11)
  - with $Start(\mathtt{P}_{\Sigma_{s_i}}) = \mathtt{x}_i$ and
  - $Targ(\mathtt{P}_{\Sigma_{s_i}}) = \mathtt{x}''_i$ for all $i = \{1, \ldots, n\}$ and

- a decentralised diagnosis model $\mathtt{G}_j$ with transition $(\mathtt{x}''_j \xrightarrow{o_{k+1}} \mathtt{x}'_j) \in \mathtt{T}_j$ and $\mathtt{x}''_i = \mathtt{x}'_i \ \forall i \neq j$

  $\Rightarrow \mathtt{R}(o_{k+1}, \mathtt{x}') = \mathtt{R}_j(\mathtt{x}'_j) \times \prod_{i \neq j} \mathtt{R}'_i(\mathtt{x}'_i)$ (see Definition 13)

  $\Rightarrow (x'_j, l'_j) \in \mathtt{R}_j(\mathtt{x}'_j)$ and $(x'_i, l'_i) \in \mathtt{R}'_i(\mathtt{x}'_i) \quad \forall i \neq j$

$\Rightarrow$ From the above and Definition 11 it follows that

- $\forall i = \{1, \ldots, n\}$ the path $\mathtt{P}_{\Sigma_{s_i}} = \mathtt{x}_i \xrightarrow{s_i^2} \mathtt{x}_i^2 \cdots \xrightarrow{s_i^{m_i}} \mathtt{x}''_i$ in $\mathtt{G}_i$ with $s_i^h \in \Sigma_{s_i} \ \forall h = \{2, \ldots, m_i\}$ corresponds to a path $P_{\Sigma_{\bar{o}_i}} = P^1_{\Sigma''_i} \xrightarrow{s_i^2} P^2_{\Sigma''_i} \cdots \xrightarrow{s_i^{m_i}} P^{m_i}_{\Sigma''_i}$ in G_i such that
  - $\Sigma_{\bar{o}_i} = \Sigma_i \setminus \Sigma_{o_i}$ and $\Sigma''_i = \Sigma_{\bar{o}_i} \setminus \Sigma_{s_i}$,
  - $x_i = Start(P_{\Sigma_{\bar{o}_i}})$
  - $x''_i = Targ(P_{\Sigma_{\bar{o}_i}})$
  - there exists a fault label l_i such that $l'_i = l_i \cup \bigcup_{h=1}^{m_i} EvSet(P^h_{\Sigma''_i}, \Sigma_{f_i})$
- there exists a state x'_j such that $(x''_j \xrightarrow{o_{k+1}} x'_j) \in T_j$ and $x'_i = x''_i \ \forall i \neq j$

$\Rightarrow$ From the above it follows that

- $(x, l) \in \mathtt{R}(o_k, \mathtt{x})$ with $x = \prod_{i=1}^n x_i$ and $l = \prod_{i=1}^n l_i$

  $\Rightarrow (x, l) \in \hat{\mathtt{R}}(\hat{\mathtt{x}}_k, o_k)$ (see (7))

  $\Rightarrow (x, l) \in \hat{R}(\hat{x}_k)$ (see induction hypothesis)
- there exists a path $p = P_{\Sigma_u} \xrightarrow{o_{k+1}} x'$ in G such that
  - $EvSeq(P_{\Sigma_u}, \Sigma_s) = EvSeq(\mathtt{P}_{\Sigma_s}, \Sigma_s)$
  - $x = Start(P_{\Sigma_u})$ and
  - $l' = l \cup EvSet(P_{\Sigma_u}, \Sigma_f)$,

$\Rightarrow$ From the above it follows that

$(\hat{x}_k \xrightarrow{o_{k+1}} \hat{x}_{k+1}) \in \hat{T}$ and $(x', l') \in \hat{R}(\hat{x}_{k+1})$ (see Definition 6)

$\Rightarrow \hat{R}(\hat{x}_{k+1}) \supseteq \hat{\mathtt{R}}(o_{k+1}, \hat{\mathtt{x}}_{k+1})$ □ ■

Thus, we have proved that function $\hat{\mathtt{R}}$ of the deterministic diagnosis model retrieves the same diagnosis information as the state labelling function $\hat{R}$ of the diagnoser. By computing the relevant part of the deterministic diagnosis model, that is the part that is consistent with the events observed, we are therefore able to retrieve the correct global diagnosis information. This argumentation does not depend on whether this relevant part is computed off-line as it is the case for the

deterministic diagnosis model, or on-line as it is the case for the diagnosis based on the remaining compiled diagnosis models. Therefore, for all these models, we can describe on-line diagnosis algorithms that are guaranteed to retrieve the correct diagnosis information.

2.5.4 On-line Diagnosis Based on the Compiled Diagnosis Models

We now show how to compute the diagnosis information that is consistent with a sequence of observations. Similarly to the direct diagnosis approach (see Section 2.4.3) we do this by retrieving, on-line, the label of the corresponding deterministic diagnosis model state.

Algorithm 4 incrementally updates the diagnosis information following a new observable event σ, based on any of the compiled diagnosis models $DiagModel$. The input parameter $DiagInfoRef$ refers to the states whose labels contain the current diagnosis information. Initially, it contains the initial state $\mathtt{x}_0$ or $\hat{\mathtt{x}}_0$ depending on the approach (in case the diagnosis is based on the decentralised model, state $\mathtt{x}_0$ needs to be computed first).

To obtain the global diagnosis information, we need to consider the classical label Φ_{compID} of the decentralised diagnosis model $\mathtt{G}_{compID}$ in which σ is defined and the quiet state labels Φ'_j of the remaining models. Hence we need to determine the component in which σ is defined. We do this using function $GetCompID$ (line 3) that returns the corresponding identifier of the component by considering the b^C variables used to encode σ (see Section 2.4.1).

Now, to update the diagnosis information, we first consider all possible interactions among components that could have taken place after the last event σ_l was observed and before the new event σ is observed (lines 4–6). In the nondeterministic and deterministic diagnoser model, these interactions are already precomputed. For the other models, we need to trigger all shared transitions from states in $DiagInfoRef$ and compute their target states. For the decentralised diagnosis model, we additionally need to synchronise the shared event paths starting in states of $DiagInfoRef$ (as shown in Algorithm 2 on page 43). This is all done in function $ComputeSharedTarg$ (line 5).

In order to retrieve the states that refer to the diagnosis information that is also consistent with observing σ, we trigger this transition from the previously computed states (lines 7–13). Note that for the decentralised diagnosis model we

Algorithm 4 *DiagOnline*($DiagModel$, $DiagInfoRef$, σ)

INPUT:
$DiagModel$: symbolic diagnosis model (either $\mathtt{G}_i$, $\mathtt{G}$, $\widetilde{\mathtt{G}}$ or $\hat{\mathtt{G}}$)
$transObs$ are the model's observable transitions ($\mathtt{T}_{o_i}$, $\mathtt{T}_o$, $\widetilde{\mathtt{T}}$ or $\hat{\mathtt{T}}$)
b^{Start} and b^{Targ} are the variables encoding states in $transObs$
$DiagInfoRef$: states whose labels contain the current global diagnosis information
σ : new observation

Initialise
$\mathtt{X}_{Start} \leftarrow DiagInfoRef$
$compID \leftarrow GetCompID(\sigma)$

Trigger shared events
if $DiagModel = \mathtt{G}_i$ or $\mathtt{G}$ **then**
  $\mathtt{X}_{Start} \leftarrow \mathtt{X}_{Start} \vee ComputeSharedTarg(DiagModel, DiagInfoRef)$
end if

Trigger new observation σ
if $DiagModel = \mathtt{G}_i$ **then**
  $\mathtt{X}_{Targ} \leftarrow SwapVar(\sigma \wedge \mathtt{X}_{Start} \wedge T_{o_{compID}}, b^{\mathtt{X}}_{compID}, b^{\mathtt{X}'}_{compID})$
  $\mathtt{X}_{Targ} \leftarrow ExtractVar(\mathtt{X}_{Targ}, b^{\mathtt{X}}_{compID})$
else
  $\mathtt{X}_{Targ} \leftarrow ExtractVar(\sigma \wedge \mathtt{X}_{Start} \wedge transObs, b^{Targ})$
  $\mathtt{X}_{Targ} \leftarrow SwapVar(\mathtt{X}_{Targ}, b^{Start}, b^{Targ})$
end if
$newDiagInfoRef \leftarrow \mathtt{X}_{Targ}$

Look up diagnosis information
if $DiagModel = \hat{\mathtt{G}}$ **then**
  $\mathtt{X}_{Targ} \leftarrow ExtractVar(\Gamma \wedge \mathtt{X}_{Targ}, b^{\mathtt{X}})$
end if
$diagInfo \leftarrow AbstractVar(\mathtt{X}_{Targ} \wedge \Phi_{compID}, b^{\mathtt{X}}_{compID})$
for all components $j \neq compID$ **do**
  $diagInfo \leftarrow AbstractVar(diagInfo \wedge \Phi'_j, b^{\mathtt{X}}_j)$
end for
OUTPUT:
states $newDiagInfoRef$ whose labels contain the new global diagnosis, diagnosis information $diagInfo$

need to consider only the local observable transition and swap local state variables of the model $compID$ in which σ is defined (line 8).

Finally, the diagnosis information is obtained by combining the labels of the states computed above (lines 18–21). For the deterministic diagnosis model, however, this first requires the computation of the global diagnosis states (line 16).

2.5.5 Experimental Comparison of On-Line Diagnosis Algorithms

We have tested this algorithm on the same case study and scenarios we used to evaluate the direct diagnosis algorithms (see Section 2.4.3). Figure 2.27 shows the diagnosis times obtained with the various compiled models. The fastest approach is the one based on the nondeterministic diagnosis model. Here, the diagnosis time is only composed of the triggering of observable transitions and the retrieval of diagnosis information. Compared to the deterministic model, the former takes only half the time, because the states of the nondeterministic model from which the diagnosis information is derived are obtained explicitly (without the need of using Γ (see line 16 of Algorithm 4)). What the triggering of observable transitions concerns (lines 7–14) , so this is not much slower than in the deterministic model. This results from two facts: first the efficiency of triggering transitions depends on the size of the BDDs representing them and $\widetilde{\mathrm{T}}$ is much smaller than $\hat{\mathrm{T}}$, and second triggering transition sets is symbolically very efficient. Overall the on-line diagnosis based on the nondeterministic diagnosis model is therefore the fastest diagnosis method.

To consider the time/space complexity tradeoff we now also look at the space requirements of the compiled diagnosis approaches (see Figure 2.26 on page 71). For instance, the decentralised diagnosis models are 8 times smaller than the nondeterministic model, yet only 3.5 times slower. Compared to the component based diagnosis the diagnosis times are five times faster, due to a significant decrease in precomputation time. Indeed, the compiled approach avoids the costly update of fault labels; the precomputation is limited to the on-line synchronisation and abstraction of shared events. Furthermore, these operations are more efficient since the BDDs that are manipulated are defined over fewer variables (b_i^{X} instead of over $b_i^X \cup b_i^F$). For similar reasons, the centralised and nondeterministic models also outperform the corresponding direct models.

As far as the diagnoser and the deterministic diagnosis model are concerned,

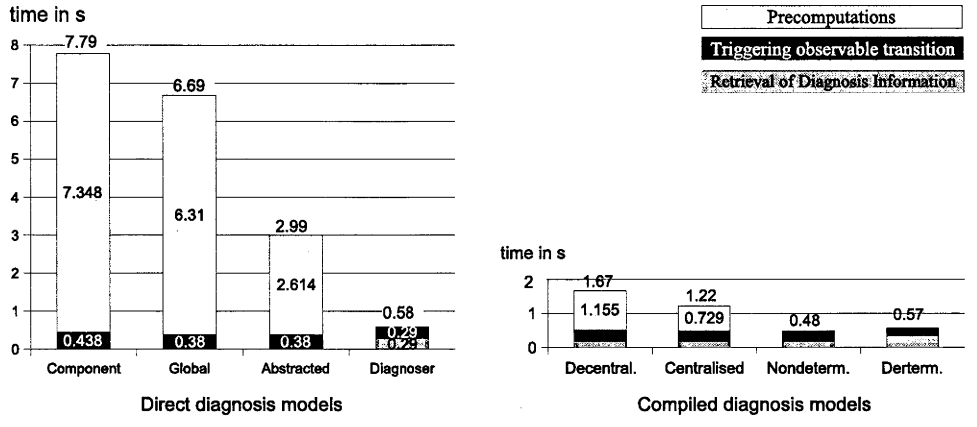


Figure 2.27: Average diagnosis times for the different symbolic computation steps of all direct (on the left) and compiled (on the right) diagnosis methods, based on 100 scenarios consisting each of a sequence of 10000 observations.

we can state that they neither differ in their size nor in the time taken by the corresponding diagnosis algorithms. However, these algorithms are not competitive since they are neither as space efficient nor as time efficient as the diagnosis based on the nondeterministic diagnosis model.

For the other three diagnosis models, the diagnosis time increases only linearly in the number of observations (see Figure 2.28). This is essential, since the diagnosis information has to be updated continuously with the flow of observations.

In conclusion, we can state that on-line diagnosis based on the decentralised, centralised or nondeterministic models is significantly faster than based on the corresponding direct models. Since the latter two compiled models additionally require less memory than their counterpart in the direct approach, it follows that the global and abstracted models are not competitive. Due to their very small size the component models remain an alternative.

2.6 Related Work

In the past, off-the-shelf symbolic model-checkers have been used for fault diagnosis [Cordier & Largouët, 2001] and for checking diagnosability [Cimatti, Pecheur, & Cavada, 2003]. In the work of Cordier and Largouët 2001, the model checker is the diagnoser, so to speak. It checks whether traces of the system exist which satisfy a temporal logic formula stating what the observations are and expressing constraints between their time of occurrence. While this approach is very attractive

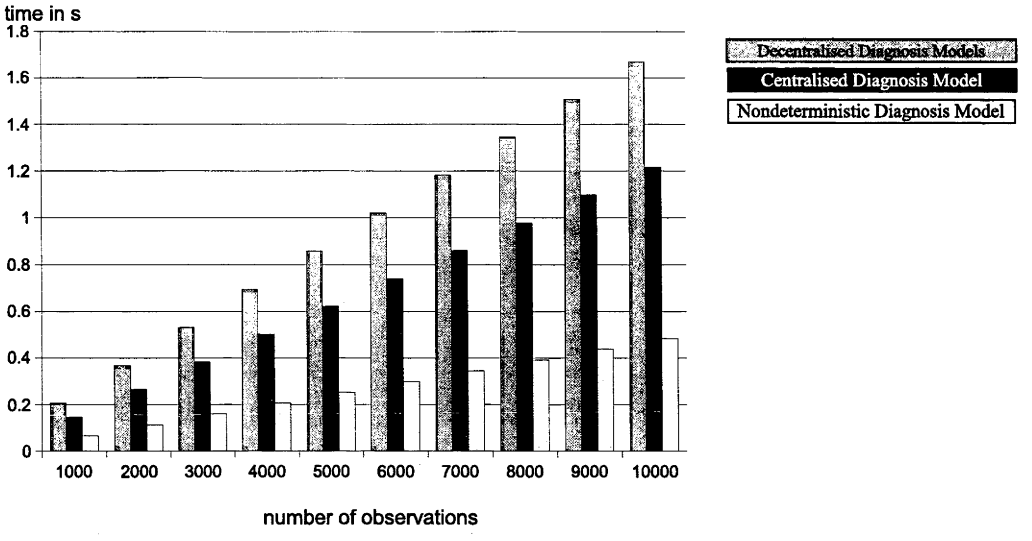


Figure 2.28: Development of the diagnosis times for the decentralised, centralised and nondeterministic diagnosis model.

for off-line diagnosis, it is much less suited to on-line diagnosis because symbolic model-checkers do not provide facilities for incremental computation.

Static systems have previously been successfully diagnosed symbolically using BDDs [Torta & Torasso, 2004] and Decomposition Negation Normal Forms [Darwiche, 1998]. The latter of which are not very suitable for diagnosing dynamic systems since the main operation, namely the triggering of transitions, requires here exponential time in the size of the representation rather than polynomial time as for BDDs.

From a symbolic point of view, the work by Marchand and Rozé 2002 is the closest to ours. They present a theoretical framework for modelling a system in terms of polynomial equations which are generalisations of Boolean formulae, and define a form of nondeterministic diagnoser within this framework. The framework is state-based, meaning that observations and faults are directly associated to states (by means of polynomial equations). The nondeterministic diagnoser is obtained by gathering the states of the global model into observational/fault equivalence classes. This nondeterministic diagnoser is quite different from ours and still requires computations of non-trivial complexity to be performed on-line. Furthermore, our event-based framework lends itself to a straightforward extension to intermittent faults along the lines suggested by Contant *et al.* 2002, while extending a state-based framework in this direction is more difficult. However, it would be interesting to implement the approach by Marchand and Rozé (their

paper does not report any implementation), and compare the results to ours.

Very recently, Xue and colleagues 2005 have shown how BDDs can help to diagnose distributed discrete-event systems modelled as Petri nets. In their approach, which is limited to two components whose interactions can be observed, BDDs are used to encode the diagnosis information. However, the model itself is represented in an nonsymbolic way and the computation of diagnosis information is also nonsymbolic. Hence a continual conversion from nonsymbolic to symbolic representation is required.

In the area of model-based planning the symbolic approaches introduced in [Bertoli *et al.*, 2001], [Albore & Bertoli, 2006] and [Jensen & Veloso, 2000] are the closest to ours. In the first one the authors show how to efficiently construct plans under partial observability. This requires the consideration of belief states that are similar to the state information contained in the diagnoser states. The second work then reasons over these symbolic belief states to compute assumption-based plans. The plans are said to be safe if their execution guarantees that it can always be observed whether the assumptions are met or not. Safe planning can thus also be seen as enforcing a specific form of diagnosability by construction. Finally, the third approach defines a planning domain description language for non-deterministic multi-agent universal planning. The language is defined such that it allows an efficient BDD encoding. Our work applies similar techniques to a different problem, and, unlike these three approaches, focuses on using the techniques in decentralised computations.

Several nonsymbolic decentralised methods for on-line diagnosis have been introduced to deal with the space complexity of large scale systems like [García *et al.*, 2005] or [Pencolé & Cordier, 2005] described in Section 1.3.2. The closest to our decentralised diagnosis approach is the work of Debouk and colleagues 2000 who have proposed a framework consisting of a set of diagnosers, that each explain the observations from one site as described on page 9. The authors present a general protocol that computes the correct diagnosis information for any system, as well as two more efficient protocols which are only correct under certain assumptions on the system. In the general case, the diagnosis information is retrieved on the basis of extended diagnoser state labels and of the unobservable reach of these diagnoser states. The unobservable reach includes all states that can be reached from states of the diagnoser state label by triggering only global transition sequences that are unobservable for the particular site and whose last event is observable by another site. Thus the unobservable reach is similar to our quiet state labels except

that we also consider transition sequences ending with a fault event. Furthermore, since our quiet state labels are computed based on local models, we only consider unobservable transition sequences that do not contain shared events. Note that the approach of Debouk and colleagues does not require to consider interactions between components on-line, since all site diagnosers are computed based on the global model and all diagnoser states contain global diagnosis information. Hence this approach does not scale to large systems for which the global model cannot be computed.

The continuous diagnosis approach described on page 10 is similar to our diagnosis method based on the abstracted model. While the continuous method computes the diagnosis information based on the snapshot and the historic diagnostic set, we retrieve the new diagnosis information by first triggering all fault transitions starting in one of the system states consistent with the old diagnosis information. The labels of these fault transitions contain the new faults, which are exactly those included in the snapshot diagnostic set. In the continuous diagnosis approach, the base model, the monitor, is computed on-line directly from the component model, which differs from the way we approach diagnosis based on our abstracted model. Due to the large monitor size, old monitor states are removed such that only the part of the monitor that was relevant to the computation of the last q diagnosis information is kept in memory. The efficiency of this approach depends on the extent to which the monitor part that is stored in memory can be used to compute the new diagnosis information. However, since the approach does not precompute any diagnosis information locally, we believe that even in the best case, the diagnosis times will only be as good as those we achieve with our direct abstracted model. Furthermore, the size of the relevant monitor part is likely to be significantly bigger than the size of our decentralised diagnosis models, which allow for a faster on-line diagnosis.

2.7 Summary

In this chapter, we have shown that using symbolic techniques by means of BDDs significantly improves the diagnosis of discrete-event systems. Our two approaches indicate not only that the symbolic encoding of an existing diagnosis approach, the well-know efficient diagnoser approach by Sampath, leads to a considerable decrease in space complexity but also that the design of a compiled symbolic framework that systematically exploits the advantages of BDDs can lead to a significant reduction

in diagnosis time.

In order to determine which diagnosis approach is best suited for a given application with specific time and space requirements we have presented eight symbolic diagnosis methods and have analysed their time/space tradeoff. Our experiments clearly indicate the superiority of four of these methods. For large applications where space is critical the diagnosis is best based on the component or decentralised models. On the other hand, smaller applications can be more efficiently diagnosed using the centralised or nondeterministic model.

Although BDDs are very suitable to handle large sets of data, our experiments have also shown that the set size impacts on the BDD size and thus on the efficiency of all our symbolic on-line diagnosis approaches. This suggests that systems allowing a more precise diagnosis, that is those for whom the diagnosis algorithm returns fewer diagnosis candidates, can be faster diagnosed than those where the diagnosis result consists of large numbers of possible explanations. The next chapter addresses this issue by computing a minimum-cost solution for making the whole system diagnosable. This then assists a systems designer in reducing the number of explanations.

Chapter 3

Scalable Diagnosability Checking

3.1 Introduction

The on-line diagnosis approaches described in the previous chapter determine all faults that *could* have occurred. However, for many applications one rather wishes to know what faults have *definitely* occurred. Computing the latter in general requires *diagnosability* of the system, that is, the guarantee that the occurrence of a fault can be detected with certainty after a finite number of subsequent observations [Sampath *et al.*, 1995]. Consequently, diagnosability analyses should be performed on the system before any diagnostic reasoning. The diagnosability results then help in choosing the type of diagnostic algorithm that can be performed and provide some information of how to change the system to make it more diagnosable.

In this chapter, we propose a formal framework for checking diagnosability on event-driven systems which is mainly motivated by two facts. On the one hand, checking diagnosability means determining the existence of two behaviours in the system that are not *distinguishable*. However, in realistic systems, there is a combinatorial explosion of the search space that does not permit the practical use of classical and centralised diagnosability checking methods [Sampath *et al.*, 1995] like the twin plant method [Jiang *et al.*, 2001; Yoo & Lafortune, 2002]. On the other hand, in the case of a nondiagnosable system, verifying its nondiagnosability may not be sufficient; the diagnosability analysis should also provide the reasons why the system is not diagnosable. Then a systems designer can make the appropriate changes to improve the diagnostic reasoning, for instance, by adding sensors.

Our proposal makes several contributions to the diagnosability problem. The first one is the definition of a new theoretical framework where the classical diagnos-

ability problem is described as a distributed search problem. Instead of searching for indistinguishable behaviours in a single FSM, often called the global twin plant, we propose to distribute the search based on local twin plants. Specifically, we exploit the modularity of the system by organising the system components into a special tree structure, known as a *jointree*, where each node of the tree is assigned a subset of the local twin plants. Once the jointree is constructed we need only synchronise the twin plants in each jointree node, and all further computation takes the form of message passing along the edges of the jointree. Using the jointree properties we show that after two messages per edge, the FSMs at all nodes are collectively consistent. This allows us to decide diagnosability by considering these FSMs in sequence instead of the large global twin plant.

We describe how messages, which are themselves FSMs, are computed and how diagnosability information can be propagated along with the messages. Furthermore, we employ a systematic iterative procedure so that only a subset of the jointree is considered at a time and the loop terminates as soon as the current subset is sufficient for deciding diagnosability. Finally, we identify a condition under which the size of the messages can be reduced.

We also consider the practical use of the algorithm. Since the diagnosability analysis problem is complex, a complete analysis may not be possible due to lack of computational resources. Our distributed search therefore ensures that the computation is scalable in the sense that it is able to provide an approximate but exhaustive solution to the diagnosability problem when it cannot run to completion due to limited computational resources. Then we return the set of all problems that could possibly cause the nondiagnosability of a fault.

Finally, we identify those system behaviours that require modification to restore diagnosability. Since a system may admit several possibilities to remove nondiagnosable behaviour, a mechanism to rank the modifications is desirable. Our approach employs a ranking approach based on cost estimation to isolate behaviours in easily accessible components whose modification removes not only the diagnosability problem but can also be cheaply performed. Here, it is assumed that cost estimates are available that reflect important characteristics of proposed system modifications, such as accessibility of subsystems or number of affected subsystems. Furthermore, we improve upon previous approaches to solving the diagnosability problem by exploiting cost estimation to prune models of subsystems to gain computational efficiency.

This chapter is organised as follows. In Section 3.2 we define the diagnosability

problem for discrete-event systems and present a modified version of the twin plant method allowing its efficient use based on decentralised models, rather than on centralised ones. Section 3.2.3 gives an introduction to jointrees which we use in Section 3.3 to solve the diagnosability problem. Section 3.4 describes techniques to further increase the efficiency of this approach and in Section 3.5 we compare our work with related approaches. Finally, we extend our algorithm in Section 3.6 to compute the "best" system modifications that restore diagnosability.

3.2 Background

In this section we review the definition of diagnosability and the twin plant approach to diagnosability checking, and give a short introduction to jointrees. Given the new concepts we present in this chapter we also use a different running example, namely the one shown in Figure 3.1.

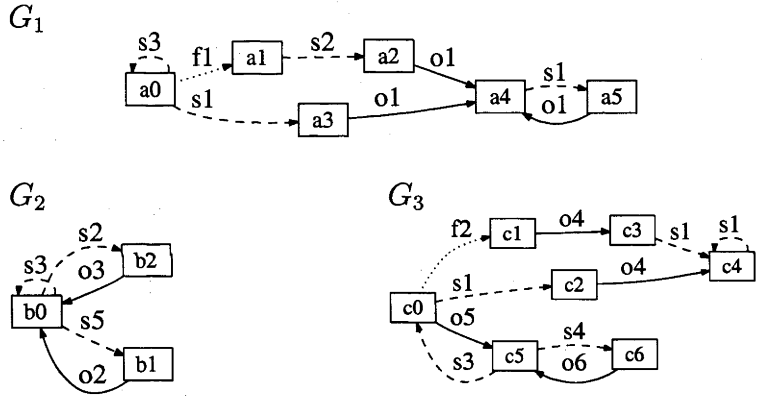


Figure 3.1: Three components of a system modelled as FSMs. Solid, dashed, and dotted lines denote observable, shared, and fault transitions, respectively.

3.2.1 Diagnosability of a Fault in Discrete-Event Systems

A fault $F \in \Sigma_f$ of the system is *diagnosable* iff its (unobservable) occurrence can always be deduced after finite delay [Sampath *et al.*, 1995]. In other words, a fault is not diagnosable if there exist two infinite paths from the initial state which contain the same infinite sequence of observable events but exactly one of which contains the fault.

More formally, let p_F denote a path starting from the initial state of the system and ending with the occurrence of a fault F in a state x_F, let s_F denote a finite path starting from x_F, and let $obs(p)$ denote the sequence of observable events in

a path p. As in [Sampath *et al.*, 1995], we assume that (i) the system is *live* (there is a transition from every state), and (ii) the observable behaviour of the system is *live* ($obs(p)$ is infinite for any infinite path p of the system). We have:

Definition 19 (Diagnosability) *F is diagnosable iff*

$$\exists d \in \mathbb{N}, \forall p_F s_F, |obs(s_F)| > d \Rightarrow$$

$$(\forall p, obs(p) = obs(p_F s_F) \Rightarrow F \text{ occurs in } p).$$

If a fault is diagnosable then a diagnostic algorithm can diagnose its occurrence with certainty based on a finite sequence of observations. Diagnosability checking thus requires the search for two infinite paths p and p', i.e. paths containing a cycle, with $obs(p) = obs(p')$ such that F is in p but not in p'. The pair (p, p') is called a *critical pair* [Cimatti, Pecheur, & Cavada, 2003]. From here on we will write *path* to mean a path that starts from the initial state of the system.

3.2.2 Twin Plant Approach for Diagnosability Checking

The idea of the twin plant method is to build a FSM that compares every pair of paths (p, p') in the system that are equivalent to the observer ($obs(p) = obs(p')$), and apply Definition 19 to determine diagnosability [Jiang *et al.*, 2001].

For the sake of clarity in the rest of the chapter, we now present the twin plant method in a new way, based on the decentralised model instead of the global model. We start with the *interactive diagnoser* [Pencolé, 2005], which gives the set of faults that can possibly have occurred for each sequence of observable and shared events[1].

Definition 20 *The interactive diagnoser of a component G_i is the nondeterministic finite state machine $\widetilde{G}_i = \langle \widetilde{X}_i, \widetilde{\Sigma}_i, \widetilde{x}_{0_i}, \widetilde{T}_i \rangle$ where*

- *$\widetilde{X}_i$ is the set of states ($\widetilde{X}_i \subseteq X_i \times \mathcal{F}$ with $\mathcal{F} \subseteq 2^{\Sigma_{f_i}}$),*
- *$\widetilde{\Sigma}_i$ is the set of events ($\widetilde{\Sigma}_i = \Sigma_{o_i} \cup \Sigma_{s_i}$),*
- *$\widetilde{x}_{0_i} = (x_{0_i}, \emptyset)$ is the initial state, and*
- *$\widetilde{T}_i \subseteq \widetilde{X}_i \times \widetilde{\Sigma}_i \times \widetilde{X}_i$ is the transition set $(x, \mathcal{F}) \xrightarrow{\sigma} (x', \mathcal{F}')$ such that there exists a transition sequence $x \xrightarrow{\sigma_1} x_1 \cdots \xrightarrow{\sigma_m} x_m \xrightarrow{\sigma} x'$ in G_i with $\Sigma'_i = \{\sigma_1, \ldots, \sigma_m\} \subseteq \Sigma_{f_i} \cup \Sigma_{u_i}$ and $\mathcal{F}' = \mathcal{F} \cup (\Sigma'_i \cap \Sigma_{f_i})$.*

[1] To avoid complex notation we reuse several symbols of the previous chapter like $\widetilde{\ }$ and $\hat{\ }$. This shall not cause confusion.

Thus the interactive diagnoser is very similar to our decentralised diagnosis model (see Definition 11 on page 62). The main difference is that the former is nondeterministic while the latter is deterministic. Figure 3.2 (top) depicts the interactive diagnoser for component G_1 shown in Figure 3.1. Following the transitions $s2, o1$ from the initial state of the diagnoser, for example, we arrive at state $(a4, \{f1\})$, meaning that the system contains a path to state $a4$ on which the sequence of observable and shared events is exactly $s2, o1$ and the set of faults is exactly $\{f1\}$.

A local twin plant is obtained by synchronising two interactive diagnosers. The synchronisation operation, denoted $Sync(M_1, \ldots, M_n)$, is the classical synchronisation operation on the n finite state machines based on their common events. The result M of the synchronisation is obtained as the Cartesian product $M_1 \times \ldots \times M_n$ restricted with the following rule (see also the definition of the global model transitions on page 31):

> From any state $(x_1, \ldots, x_n)$, the event e can occur if for all machines M_j where e can occur, there exists in M_j a transition $x_j \xrightarrow{e} x'_j$.

The *local twin plant* is then constructed by synchronising two instances $\widetilde{G}_i^l$ (left) and $\widetilde{G}_i^r$ (right) of the same interactive diagnoser based on the observable events $\Sigma_{o_i} = \Sigma_{o_i}^l = \Sigma_{o_i}^r$. Since only observable behaviours are compared, the shared events must be distinguished between the two instances: in $\widetilde{G}_i^l$ (resp. $\widetilde{G}_i^r$), any shared event $\sigma \in \Sigma_{s_i}$ from $\widetilde{G}_i$ is renamed l:$\sigma \in \Sigma_{s_i}^l$ (resp. r:$\sigma \in \Sigma_{s_i}^r$).

Definition 21 (Local twin plant) *The local twin plant of G_i is the finite state machine*

$$\hat{G}_i = Sync(\widetilde{G}_i^l, \widetilde{G}_i^r).$$

Figure 3.2 (bottom) depicts part of the twin plant for component G_1 in Figure 3.1. The top labels $x0, \ldots, x13$ of the states are their identifiers to which we will refer in subsequent figures. State labels are composed of a state in the left interactive diagnoser (middle label) and one in the right interactive diagnoser (bottom label). Each state of the twin plant is a pair $\hat{x} = ((x^l, \mathcal{F}^l), (x^r, \mathcal{F}^r))$ that represents two possible diagnoses given the same sequence of observable events. If some fault F belongs to $\mathcal{F}^l \cup \mathcal{F}^r$ but not to $\mathcal{F}^l \cap \mathcal{F}^r$, then the occurrence of F cannot be deduced in this state. In this case, the state $\hat{x}$ is called *F-nondiagnosable*; otherwise it is called *F-diagnosable*. In Figure 3.2 the oval nodes represent f1-nondiagnosable states.

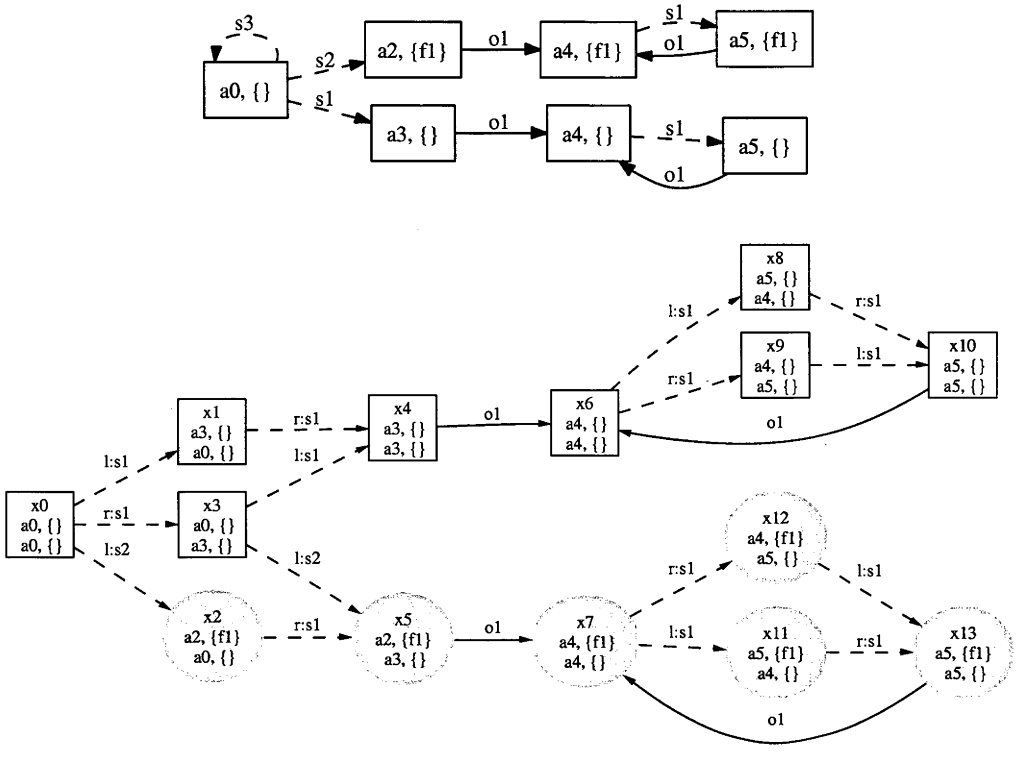


Figure 3.2: Interactive diagnoser (top) and part of a twin plant (bottom).

In the following, ω represents any set of components $\mathbb{G}_\omega = \{G_{j_1}, \ldots, G_{j_{|\omega|}}\}$ of the system with $|\omega| \geq 1$. The *ω-coupled twin plant* is the twin plant of ω obtained by synchronisation of the local twin plants, that is by computing $\hat{G}_\omega = Sync(\hat{G}_{j_1}, \ldots, \hat{G}_{j_{|\omega|}})$.

Clearly, by extension, a state $\hat{x} = (\hat{x}_{j_1}, \ldots, \hat{x}_{j_{|\omega|}})$ is *F-nondiagnosable* if any state $\hat{x}_{j_m}$ is *F-nondiagnosable* in $\hat{G}_{j_m}$. This results from the fact that a fault can only occur in one component.

We now show the relation of ω-coupled twin plants and the global twin plant using the *Sync* operator. For the sake of clarity we explicitly state the common events $\mathcal{C}$ that are shared by more than one FSM using the notation $Sync(\{M_1, \ldots, M_n\}, \mathcal{C})$. Since the synchronisation operation *Sync* is commutative and associative, and $\Sigma^l_{s_i}$, $\Sigma^r_{s_i}$ and Σ_{o_i} are disjoint sets by definition, it then follows:

$$
\begin{aligned}
\hat{G}_\omega &= Sync\Big(\{\hat{G}_{j_1},\ldots,\hat{G}_{j_{|\omega|}}\},\bigcup_{i=j_1}^{j_{|\omega|}}(\Sigma^l_{s_i}\cup\Sigma^r_{s_i})\Big)\\
&= Sync\Big(\Big\{Sync(\{\widetilde{G}^l_{j_1},\ldots,\widetilde{G}^l_{j_{|\omega|}}\},\bigcup_{i=j_1}^{j_{|\omega|}}\Sigma^l_{s_i}),\\
&\qquad Sync(\{\widetilde{G}^r_{j_1},\ldots,\widetilde{G}^r_{j_{|\omega|}}\},\bigcup_{i=j_1}^{j_{|\omega|}}\Sigma^r_{s_i})\Big\},\bigcup_{i=j_1}^{j_{|\omega|}}\Sigma_{o_i}\Big)\Big)\\
&= Sync\Big(\{\widetilde{G}^l_\omega,\widetilde{G}^r_\omega\},\bigcup_{i=j_1}^{j_{|\omega|}}\Sigma_{o_i}\Big). \qquad (3.1)
\end{aligned}
$$

In other words, any ω-coupled twin plant can be also obtained by first synchronising the interactive diagnosers over the set of shared events, to obtain two instances of the diagnoser of ω, followed by the synchronisation of the two diagnoser instances over the set of observable events.

Consequently, $\omega = \mathbb{G} = \{G_1,\ldots,G_n\}$ is the ω-coupled twin plant that represents the *global twin plant* GTP of the system where all paths of G with the same observable behaviour are compared. Hence the following fundamental result which follows directly from a similar result presented in [Jiang *et al.*, 2001] regarding the GTP.

Theorem 3.2.1 *F is diagnosable in G iff, in the $\mathbb{G}$-coupled twin plant, there is no path p with a cycle containing at least one observable event and one F-nondiagnosable state.*

Such a path p represents a critical pair (p_1, p_2), and is called a *critical path.* The twin plant method searches for such a path in the GTP. In this chapter, we propose a new algorithm that avoids building the global twin plant, which is impractical for systems with a large number of states. Instead, local twin plants are built for components G_i of the system. Since the existence of a critical path in a local twin plant does not imply nondiagnosability of the global system, we need to propagate information between local twin plants, which we accomplish by message passing on a *jointree.* This will then allow us to search local twin plants for a critical path rather than the GTP.

3.2.3 Jointrees

Jointrees have been a classical tool in probabilistic reasoning and constraint processing [Shenoy & Shafer, 1986; Dechter, 2003] and correspond to *tree decompositions* [Robertson & Seymour, 1986]. For our purposes, a jointree is a tree whose nodes are labelled with sets of events satisfying two special properties:

Definition 22 (Jointree) *Given a set of FSMs $M_1, \ldots, M_n$ defined over events $\Sigma_1, \ldots, \Sigma_n$ respectively, a jointree is a tree where each node is labelled with a subset of $\Sigma = \bigcup_i \Sigma_i$ such that*

- *every event of Σ_i is contained in at least one node, and*
- *if an event is shared by two distinct nodes, then it also occurs in every node on the path connecting the nodes.*

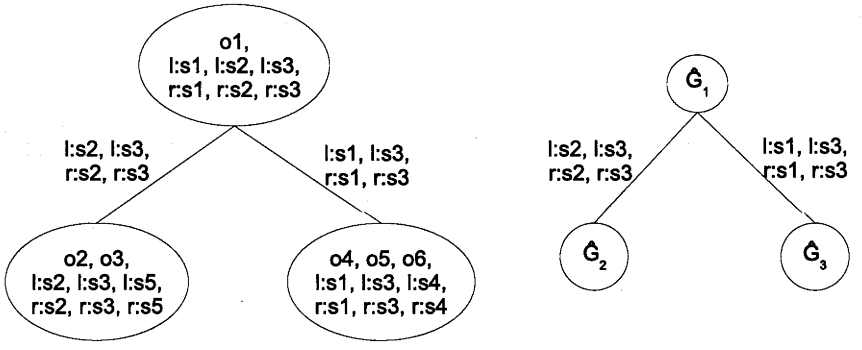


Figure 3.3: Jointree (left) and assignment of local twin plants to jointree nodes (right).

Figure 3.3 (left) depicts a jointree for the three local twin plants for the system in Figure 3.1. The intersection of two neighbouring nodes, that is their common event set, is called a *separator*, which is shown on every edge in Figure 3.3.

Once a jointree is constructed each FSM G_i is assigned to a node that contains all of its events Σ_i. Figure 3.3 (right) depicts such an assignment. Note that in general each node can have multiple FSMs assigned to it.

3.3 A Jointree Algorithm for Diagnosability

The synchronisation of all twin plants would allow us to solve the diagnosability problem. However, for large systems this can easily be impractical. Therefore we

only synchronise the twin plants in each node and then pass messages to achieve consistency. Afterwards a system is diagnosable iff no jointree node has a critical path. We now show how we can achieve consistency among a set of FSMs organised in a jointree and how we can equip every jointree node with diagnosability information such that we can decide whether a path is critical or not.

Jointrees admit a generic message passing method that achieves consistency among the nodes [Dechter, 2003]. In our case this translates into a method that achieves consistency of all FSMs labelling the jointree nodes. The messages passed on will themselves be FSMs. In this section we describe how these messages can be computed and passed and how the diagnosability information can be propagated correctly as part of the messages. This will then also enable us to present an iterative diagnosability algorithm at the end of this section.

3.3.1 Establishing Consistency

While FSMs assigned to the same tree node are synchronised directly to obtain a local picture of the system behaviour, messages must be exchanged to achieve consistency between nodes.

Definition 23 (Global Consistency; Completeness) *A FSM M_i with events Σ_i is* globally consistent *with respect to FSMs $M_1, \ldots, M_n$ iff for every path p_i in M_i there exists a path p in the synchronised product $Sync(M_1, \ldots, M_n)$ that has with respect to Σ_i the same event sequence as p_i (i.e. $EvSeq(p, \Sigma_i) = EvSeq(p_i, \Sigma_i)$)*[2]*. A FSM M_i^c is* complete *iff it contains all globally consistent paths of M_i.*

Each edge in a jointree partitions the tree into two subtrees, and a message sent over an edge represents a summary of the collective behaviour permitted by one side of the partition. A major advantage of this method is that this summary needs only to mention events given by the separator labelling the edge; the jointree construction ensures that this equals the intersection of the two sets of events across the partition.

A message can be computed by projecting a FSM onto a subset of its events.

Definition 24 (Projection) *The* projection $\Pi_{\Sigma'}(M) = \langle X', \Sigma', x_0, T' \rangle$ *of a FSM M on events $\Sigma' \subseteq \Sigma$ is obtained from M by first contracting all transitions not labelled by an event in Σ' and then removing all states (except the initial state*

[2]See Definition of *EvSeq* on page 32.

x_0) that are not a target of any transition in the new set of transitions T'. More formally, T' is given as follows:

$$T' = \Big\{ \; x \xrightarrow{\sigma'} x' \mid x, x' \in X' \text{ and } \sigma' \in \Sigma' \text{ and } \exists\, x \xrightarrow{\sigma_1} x_1 \cdots \xrightarrow{\sigma_k} x_k \xrightarrow{\sigma'} x' \text{ in } M \text{ such that } \sigma_i \notin \Sigma' \;\; \forall i = 1, \ldots, k \Big\}.$$

Figure 3.4 shows the result of projecting part of the twin plants $\hat{G}_2$ and $\hat{G}_3$ on events {l:s2, r:s2} and {l:s1, r:s1} respectively. As this example shows, compared to the original FSM, its projection might only lead to a minor size decrease. Section 3.4.1 shows how the message size can be further decreased.

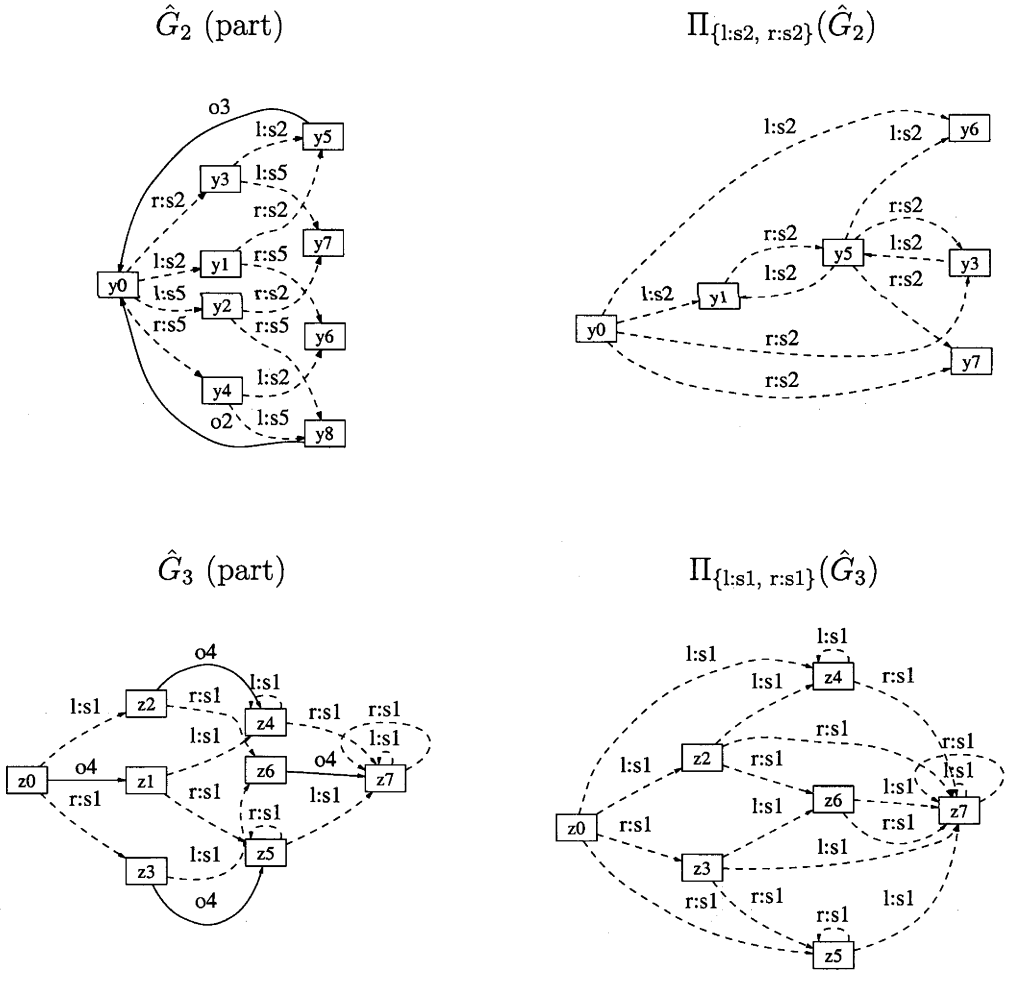


Figure 3.4: Projection of twin plant parts $\hat{G}_2$ and $\hat{G}_3$.

3.3.2 Message Passing

We now describe the message passing assuming that the FSMs in every node have been synchronised into a single FSM. To achieve consistency among the FSMs, each node of the jointree will in principle require a summary of the behaviour permitted by FSMs residing in the rest of the tree. Given the jointree properties, all these summaries can be computed in only two passes over the jointree, one inward pass, in which the root "pulls" messages toward it from the rest of the tree and one outward pass, in which the root "pushes" messages away from it toward the leaves. Once all these messages have been sent, every FSM is updated based on all the messages it receives resulting in a globally consistent FSM.

The process starts by designating any node of the tree as root. Then, in the first, inward pass, beginning with the leaves each node sends a message to its (unique) neighbour in the direction of the root. To compute this message, its FSM is synchronised with all messages it receives from its other neighbours (leaves do not have "other neighbour" and hence skip this step). The message it sends is then the projection of this FSM onto the separator between itself and the receiver of the message.

In the second, outward pass, each node (except the root) receives a message from its (unique) neighbour in the direction of the root. Again this message is computed by synchronising its FSM with all messages it received from its other neighbours and by projecting the resulting FSM onto the separator between itself and the receiver of the message.

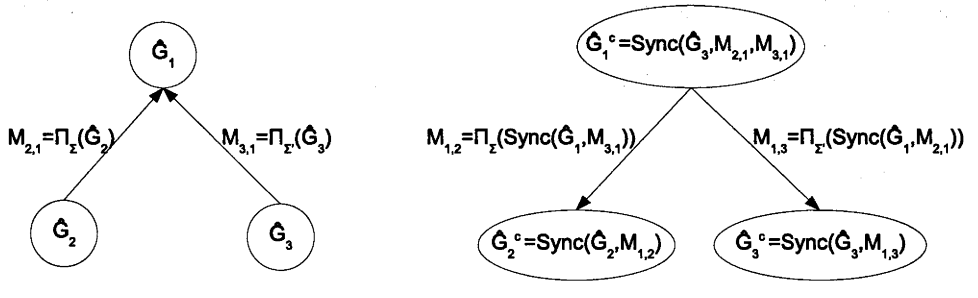


Figure 3.5: Inward (left) and outward (right) message propagation using jointrees, where $\Sigma = \{l{:}s2, r{:}s2, l{:}s3, r{:}s3\}$ and $\Sigma' = \{l{:}s1, r{:}s1, l{:}s3, r{:}s3\}$.

Finally, each node updates its own FSM by synchronising it with messages from all its neighbours. Then every FSM M_i^c of a jointree node represents exactly the behaviour that is complete and globally possible (see proof for Theorem 3.3.1). Figure 3.5 illustrates the inward and outward propagation steps performed on the jointree of Figure 3.3 (right), resulting in the FSMs $\hat{G}_1^c$, $\hat{G}_2^c$ and $\hat{G}_3^c$. Although these

FSMs no longer conform to our previous definition of twin plants, we will continue to refer to them as such. We do this because they provide sufficient information to decide diagnosability, if the diagnosability information is propagated correctly (see next subsection).

To state the benefits of the jointree propagation we define the equivalence relation for FSMs. Two FSMs M and M' are equivalent (written $M \simeq M'$) iff they admit the same set of event sequences, or more formally:

$$\begin{aligned} & \exists \tau = x_0 \xrightarrow{\sigma_1} x_1 \cdots \xrightarrow{\sigma_k} x_k \text{ in } M \\ \leftrightarrow \quad & \exists \tau' = x'_0 \xrightarrow{\sigma_1} x'_1 \cdots \xrightarrow{\sigma_k} x'_k \text{ in } M' \end{aligned} \tag{3.2}$$

This gives us the following theorem:

Theorem 3.3.1 *Every FSM M_i^c labelling a jointree node is complete and consistent with respect to all other FSMs $M_1, \ldots, M_n$ of the tree once it is synchronised with all messages it received, i.e. $M_i^c \simeq \Pi_{\Sigma_i}(Sync(M_1, \ldots, M_n))$ holds for all FSMs.*

Proof: Let $\Sigma_1, \ldots, \Sigma_n$ be the event sets of the n FSMs $M_1, \ldots, M_n$ labelling the jointree nodes. We now prove the theorem by induction showing (i) that the synchronisation of a FSM M_i with all its inward messages results in a FSM that is consistent with all descendants $M_1, \ldots, M_{i-1}$ of M_i, that is, it is equivalent to the FSM $M'_i \simeq \Pi_{\Sigma_i}(Sync(M_1, \ldots, M_i))$. Next we show (ii) that every message $\vec{M}_{i,j}$ is equivalent to the FSM $\Pi_{\Sigma_{i,j}}(Sync(M_1, \ldots, M_i))$, where $M_1, \ldots, M_i$ is the set of all FSMs on the i-side of the jointree partition. This allows us then to show (iii) that the synchronisation of a FSM M_i with all the messages it receives results in a FSM that is consistent with all nodes $M_1, \ldots, M_n$, that is, it is equivalent to the FSM $\Pi_{\Sigma_i}(Sync(M_1, \ldots, M_n))$.

We prove the inward case by induction on the depth of the tree rooted at M_i. The base case is straightforward. A leaf is consistent with itself, has no descendants and therefore does not receive any inward messages. In the induction step we show that any given FSM M_i whose children all satisfy property (i) will also satisfy this property.

Without loss of generality we assume that $M_1, \ldots, M_{i-1}$ are the descendants of M_i which are all ordered and that $M'_{j_1}, \ldots, M'_{j_k}$ ($j_k = i - 1$) are its children. Further let $\Sigma_{j_1,i}, \ldots, \Sigma_{j_k,i}$ be the events labelling the edge of a child M_{j_h} to its parent M_i. Resulting from property (i) we have $M'_{j_h} \simeq \Pi_{\Sigma_{j_h}}(Sync(M_{j_{h-1}+1}, \ldots, M_{j_h}))$ for every child M'_{j_h} where $M_1 = M_{j_0+1}$. If M_i is synchronised with all the inward messages it received it represents the FSM:

$$
\begin{aligned}
&Sync(\vec{M}_{j_1,i}, \ldots, \vec{M}_{j_k,i}, M_i) \\
&\simeq Sync\Big(\Pi_{\Sigma_{j_1,i}}(Sync(M_1, \ldots, M_{j_1})), \ldots, \Pi_{\Sigma_{j_k,i}}(Sync(M_{j_{k-1}+1}, \ldots, M_{j_k})), M_i\Big) \\
&= Sync\Big(\Pi_{\Sigma_i}(Sync(M_1, \ldots, M_{j_1})), \ldots, \Pi_{\Sigma_i}(Sync(M_{j_{k-1}}, \ldots, M_{j_k})), M_i\Big) \\
&\quad \text{since the jointree properties guarantee that } (\Sigma_{j_{h-1}+1} \cup \ldots \cup \Sigma_{j_h}) \cap \Sigma_i \subseteq \Sigma_{j_h,i} \\
&\quad \text{for all edge labels } \Sigma_{j_h,i} \text{ and } \Sigma_1 = \Sigma_{j_0+1} \\
&\simeq Sync\Big(\Pi_{\Sigma_i}(Sync(M_1, \ldots, M_{j_k})), M_i\Big) \\
&\quad \text{since the jointree properties guarantee that no two children of } M_i \\
&\quad \text{share events not defined in } \Sigma_i \\
&\simeq \Pi_{\Sigma_i}\Big(Sync(M_1, \ldots, M_{j_k}, M_i)\Big) \\
&\quad \text{since } \Sigma_i \text{ is the event set of } M_i \text{ and therefore } M_i = \Pi_{\Sigma_i}(M_i) \\
&= \Pi_{\Sigma_i}\Big(Sync(M_1, \ldots, M_i)\Big) \qquad \text{since } M_{j_k} = M_{i-1}
\end{aligned}
$$

This concludes the proof of part (i). The proof of part (ii) follows directly from property (i). The inward message $\vec{M}_{i,j}$ sent from M_i to M_j is the FSM:

$$
\begin{aligned}
&\Pi_{\Sigma_{i,j}}(Sync(\vec{M}_{j_1,i}, \ldots, \vec{M}_{j_k,i}, M_i)) \\
&\simeq \Pi_{\Sigma_{i,j}}(\Pi_{\Sigma_i}(Sync(M_1, \ldots, M_i))) \text{ since property (i) holds for } M_i \\
&= \Pi_{\Sigma_{i,j}}(Sync(M_1, \ldots, M_i)) \\
&\quad \text{since the jointree properties guarantee that } \Sigma_{i,j} \subseteq \Sigma_i
\end{aligned}
$$

Since the jointree is symmetric, it follows that the outward message $\vec{M}_{j,i}$ is equivalent to the FSM $\Pi_{\Sigma_{i,j}}(Sync(M_{i+1}, \ldots, M_n))$. We now prove part (iii) by induction on the distance of the node from the root M_n. As the base case we consider the root M_n. Since all nodes other than M_n are descendants of M_n and therefore all messages it receives are inward messages and since it satisfies property (i) we have $M'_n \simeq \Pi_{\Sigma_n}(Sync(M_1, \ldots, M_n)) = M_n^c$. Hence the root also satisfies property (iii). We now show that this property also holds for any other node M_i once it is synchronised with all inward messages $\vec{M}_{j_1,i}, \ldots, \vec{M}_{j_k,i}$ and the outward message $\vec{M}_{j,i}$ it receives. The resulting FSM has the following form:

$$Sync(M_i, \vec{M}_{j_1,i}, \ldots, \vec{M}_{j_k,i}, \vec{M}_{j,i})$$

$$= Sync(\Pi_{\Sigma_i}(Sync(M_i, \vec{M}_{j_1,i}, \ldots, \vec{M}_{j_k,i}), \Pi_{\Sigma_i}(\vec{M}_{j,i}))$$

since the jointree properties guarantee that $\Sigma_i \cup \Sigma_{j_1,i} \ldots \cup \Sigma_{j_k,i} \cup \Sigma_{j,i} \subseteq \Sigma_i$

$$\simeq Sync(\Pi_{\Sigma_i}(Sync(M_1, \ldots, M_i)), \Pi_{\Sigma_i}(\vec{M}_{j,i})) \text{ since property (i) holds for } M_i$$

$$\simeq Sync(\Pi_{\Sigma_i}(Sync(M_1, \ldots, M_i)), \Pi_{\Sigma_i}(\Pi_{\Sigma_{j,i}}(Sync(M_{i+1}, \ldots, M_n)))$$

since property (ii) holds for $\vec{M}_{j,i}$

$$= Sync(\Pi_{\Sigma_i}(Sync(M_1, \ldots, M_i)), \Pi_{\Sigma_i}(Sync(M_{i+1}, \ldots, M_n))$$

since the jointree properties guarantee that $\Sigma_{j,i} \subseteq \Sigma_i$

$$= \Pi_{\Sigma_i}(Sync(Sync(M_1, \ldots, M_i), Sync(M_{i+1}, \ldots, M_n)))$$

$$\simeq \Pi_{\Sigma_i}(Sync(M_1, \ldots, M_n)) \qquad \square$$

Thus every jointree node is indeed consistent with all other nodes once it is synchronised with all the messages it received. In particular this means that for every path p in $\hat{G}'_i = \Pi_{\Sigma_i}(\hat{G}_1, \ldots, \hat{G}_n)$ there is also an equivalent path p_i in $\hat{G}^c_i$, i.e. p_i is defined over the same event sequence as p, and vice versa. Now, for deciding diagnosability this simple equivalence is not sufficient. In addition we need to ensure that for every *critical* path p in G'_i there is also an equivalent *critical* path p_i in G^c_i. This requires the propagation of diagnosability information.

3.3.3 Propagation of Diagnosability Information

In the rest of the section we will assume that (i) the twin plants for components have been assigned to appropriate jointree nodes and synchronised within each node, (ii) G_F is the component defining the fault F whose diagnosability is to be checked, and (iii) the node containing the twin plant $\hat{G}_F$ is chosen as root. For the sake of readability we will use the notation $\hat{G}_1, \ldots, \hat{G}_n$ (instead of $\hat{G}_{\omega_1}, \ldots, \hat{G}_{\omega_n}$) to refer to the n jointree nodes. Each of these nodes may be composed of a set of local twin plants.

Now, the root can already be examined for critical paths after the inward propagation phase for two reasons: first, the synchronisation of the root with all its incoming messages results in a globally consistent twin plant, and second, since the fault F appears in the root, the FSM already contains diagnosability informa-

tion, that is, the classification of states into diagnosable and nondiagnosable ones. If the root does not contain a nondiagnosable state, F is known to be diagnosable. Otherwise, the outward propagation phase must be carried out to determine whether another jointree node has a critical path.

Once propagation is complete, every state of a twin plant comprises a tuple $(\hat{x}_1, \ldots, \hat{x}_n)$. In particular, each state contains a state (labelled diagnosable or nondiagnosable) from $\hat{G}_F$ that has been received and synchronised with the local FSM as part of the messages pushed from the root in the outward propagation phase. To ensure diagnosability information is preserved, we must ensure that no path to a nondiagnosable state is lost in this process.

Recall that the projection operation applied to compute the outward message removes all states that are no longer a target of a transition labelled by a separator event in Σ. This can lead to the removal of nondiagnosable states, resulting in the incomplete propagation of diagnosability information. Consider for instance the twin plant $\hat{G}_u$ shown in Figure 3.6 (left). When computing the message $\mathcal{P}_u$ we remove the nondiagnosable state $u1$. This results in the consistent twin plant $\hat{G}_v^c$ which does not contain any critical paths although it should contain one as $\hat{G}_v^{c'}$ indicates.

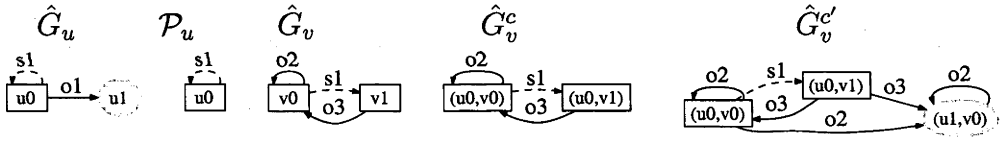


Figure 3.6: Twin plants $\hat{G}_u$, $\mathcal{P}_u = \Pi_{\{s1\}}(\hat{G}_u)$, $\hat{G}_v$, $\hat{G}_v^c = Sync(\Pi_{\{s1\}}(\hat{G}_u), \hat{G}_v)$, and $\hat{G}_v^{c'} = \Pi_{\Sigma_v}(Sync(\hat{G}_u, \hat{G}_v))$ (from left to right).

We therefore need to ensure that every message $\mathcal{M} = M_{\hat{G} \xrightarrow{\Sigma_{sep}} \hat{G}'}$ passed on from $\hat{G}$ to $\hat{G}'$ via the separator events Σ_{sep} will lead to a consistent twin plant $\hat{G}'^c$ that has a critical path iff $\Pi_{\Sigma_{sep}}(Sync(\hat{G}, \hat{G}'))$ has one.

To achieve this it is necessary to annotate every diagnosable state $\hat{x}$ in a message to capture whether it has a *nondiagnosable local future*, that is, whether there is a transition sequence starting in $\hat{x}$ and leading to a nondiagnosable state $\hat{x}_k$ such that none of the transition events is kept in the projection:

Definition 25 (Nondiagnosable Local Future) *Let $\hat{G}$ and $\hat{G}'$ be two FSMs associated with adjacent nodes in a jointree connected by an edge labelled Σ_{sep}, and let $\hat{x}_k$ denote a nondiagnosable state in $\hat{G}$. Then, a diagnosable state $\hat{x} \in \hat{G}$ has a*

nondiagnosable local future *iff there exists a transition sequence*

$$\tau = \hat{x} \xrightarrow{\sigma_1} \hat{x}_1 \cdots \xrightarrow{\sigma_k} \hat{x}_k$$

in $\hat{G}$ *such that none of the events* $\sigma_1, \ldots, \sigma_k$ *are in* Σ_{sep}.

We capture this information by adding additional nondiagnosable subgraphs to the FSM $\Pi_{\Sigma_{sep}}(\hat{G})$ obtained by projection of $\hat{G}$: for every diagnosable state $\hat{x} \in \hat{G}$ that has a nondiagnosable local future w.r.t. Σ_{sep}, a nondiagnosable *extended* terminal state $ext(\hat{x})$ and a terminal transition $\hat{x} \xrightarrow{ext} ext(\hat{x})$ are added to ensure that a critical path is not lost in the projection.

Figure 3.7 illustrates the example of passing such a message from $\hat{G}_1$ (see Figure 3.2) to $\hat{G}_3$ (the part shown in Figure 3.4) to check the diagnosability of fault f1. Here the projection $\mathcal{P} = \Pi_{\{l:s1,r:s1\}}(\hat{G}_1)$ has the two diagnosable states $x0$ and $x3$, which both satisfy the above condition (see paths $x0 \xrightarrow{l:s2} x2$ and $x3 \xrightarrow{l:s2} x5$ in $\hat{G}_1$ in Figure 3.2). Thus the message $M_{\hat{G}_1 \xrightarrow{\{l:s1,r:s1\}} \hat{G}_3}$ contains two terminal transitions. On the other hand the message sent from $\hat{G}_1$ to $\hat{G}_2$ does not contain any extended states since there does not exist such a sequence τ satisfying Definition 25 for the only diagnosable state $x0$.

Note that there is no need to introduce artificial states for a *nondiagnosable* state $\hat{x}'$. This results from the fact that all states reachable from $\hat{x}'$ via transitions labelled by events not kept in the projection can only be part of a nondiagnosable cycle if there is also a nondiagnosable cycle with state $\hat{x}'$ (according to the synchronisation operation). Hence nondiagnosability can be verified correctly based only on the latter. This allows us to state the main result of this section:

Theorem 3.3.2 *Fault F is diagnosable in G iff after both passes of jointree propagation with diagnosability information, no FSM in a jointree node has a critical path.*

Proof: We prove the theorem by showing (i) that there exists a critical path in $\hat{G}'_i = \Pi_{\Sigma_i}(Sync(\hat{G}_1, \ldots, \hat{G}_n))$ iff there exists a critical path in $\hat{G}^c_i$ and (ii) that there exists a critical path in $GTP = Sync(\hat{G}_1, \ldots, \hat{G}_n)$ iff there exists a jointree node $\hat{G}^c_i$ with a critical path.

To prove part (i) we first show that $\hat{G}'_i$ and $\Pi_{\Sigma_i}(\hat{G}^c_i)$ are guaranteed to be equivalent. Since the only events projected out in $\Pi_{\Sigma_i}(\hat{G}^c_i)$ are the *ext* events and since the extended transitions are only added as terminal transitions the equivalence follows directly from Theorem 3.3.1 on page 96. The fact that the extended transitions

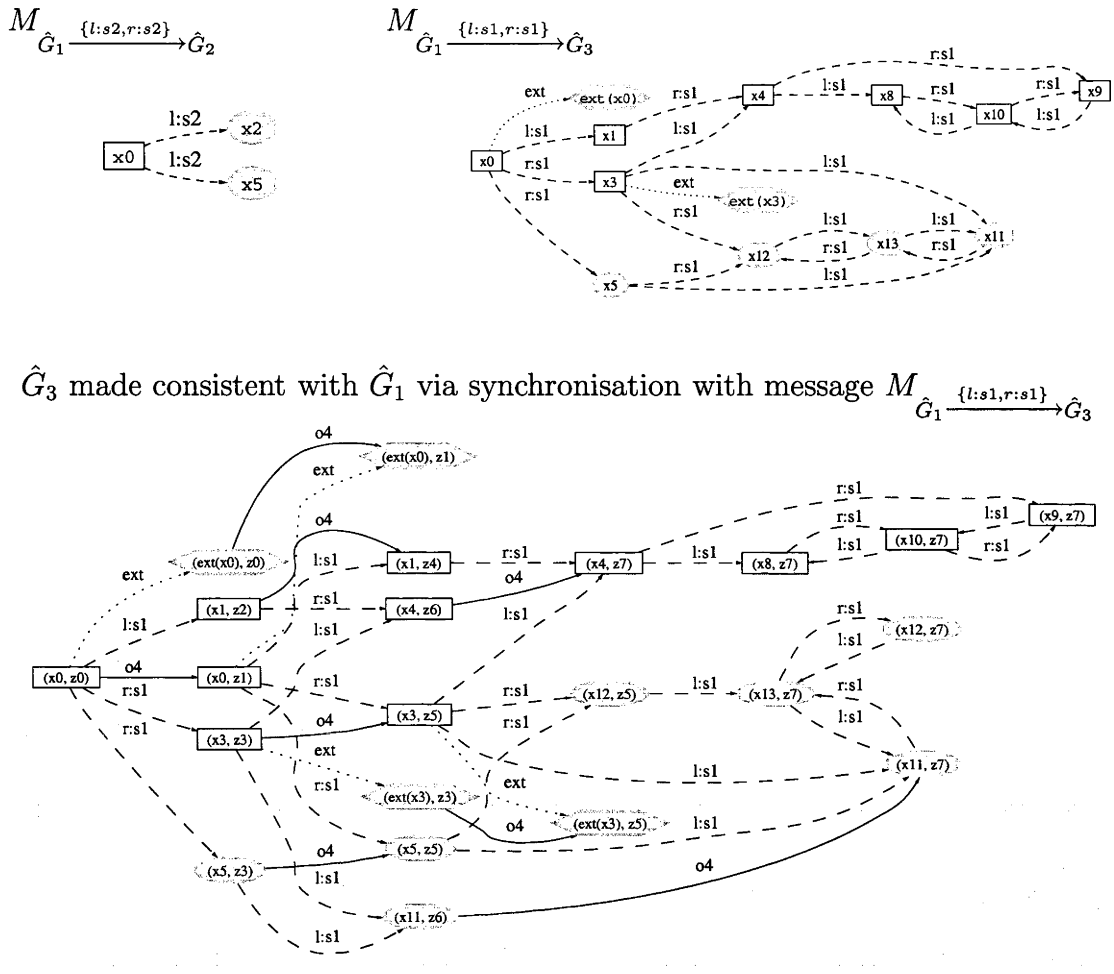


Figure 3.7: Message $M_{\hat{G}_1 \xrightarrow{\{l:s2,r:s2\}} \hat{G}_2}$ on the top left, message $M_{\hat{G}_1 \xrightarrow{\{l:s1,r:s1\}} \hat{G}_3}$ on the top right and its synchronisation with $\hat{G}_3$ on the bottom. Grey states are nondiagnosable and hexagon shaped ones are extended states.

are only added as terminal transitions also implies that there are no cycles with an extended transition in $\hat{G}_i^c$. Hence, $\hat{G}_i'$ and $\Pi_{\Sigma_i}(\hat{G}_i^c)$ are equivalent with respect to their critical paths iff this holds also for $\hat{G}_i'$ and $\hat{G}_i^c$.

Next we show that the equivalence of FSMs implies equivalence with respect to cycles. For every path $x_0 \xrightarrow{\sigma_1} x_1 \cdots \xrightarrow{\sigma_k} x_k$ in $\hat{G}_i'$ with the cycle state x_k and $k = i \in \{0, \ldots, k-1\}$ it follows that there is also a path $x_0' \xrightarrow{\sigma_1} x_1' \cdots \xrightarrow{\sigma_k} x_k'$ in $\hat{G}_i^c$ with the cycle state $x_k' = x_i'$. This results from the fact that $\hat{G}_i'$ and $\hat{G}_i^c$ are equivalent and that every path with a cycle can be extended arbitrarily often, i.e. there is also a path $p = x_0 \xrightarrow{\sigma_1} x_1 \cdots \xrightarrow{\sigma_k} x_k \xrightarrow{\sigma_{i+1}} x_{i+1} \cdots \xrightarrow{\sigma_k} x_k \xrightarrow{\sigma_{i+1}} x_{i+1} \cdots \xrightarrow{\sigma_k} x_k$ in $\hat{G}_i'$. Since the number of states in $\hat{G}_i^c$ is finite it means that states along an arbitrarily long path need to repeat themselves which means that for every path with a cycle in $\hat{G}_i'$ there must also be an equivalent path with a cycle in $\hat{G}_i^c$.

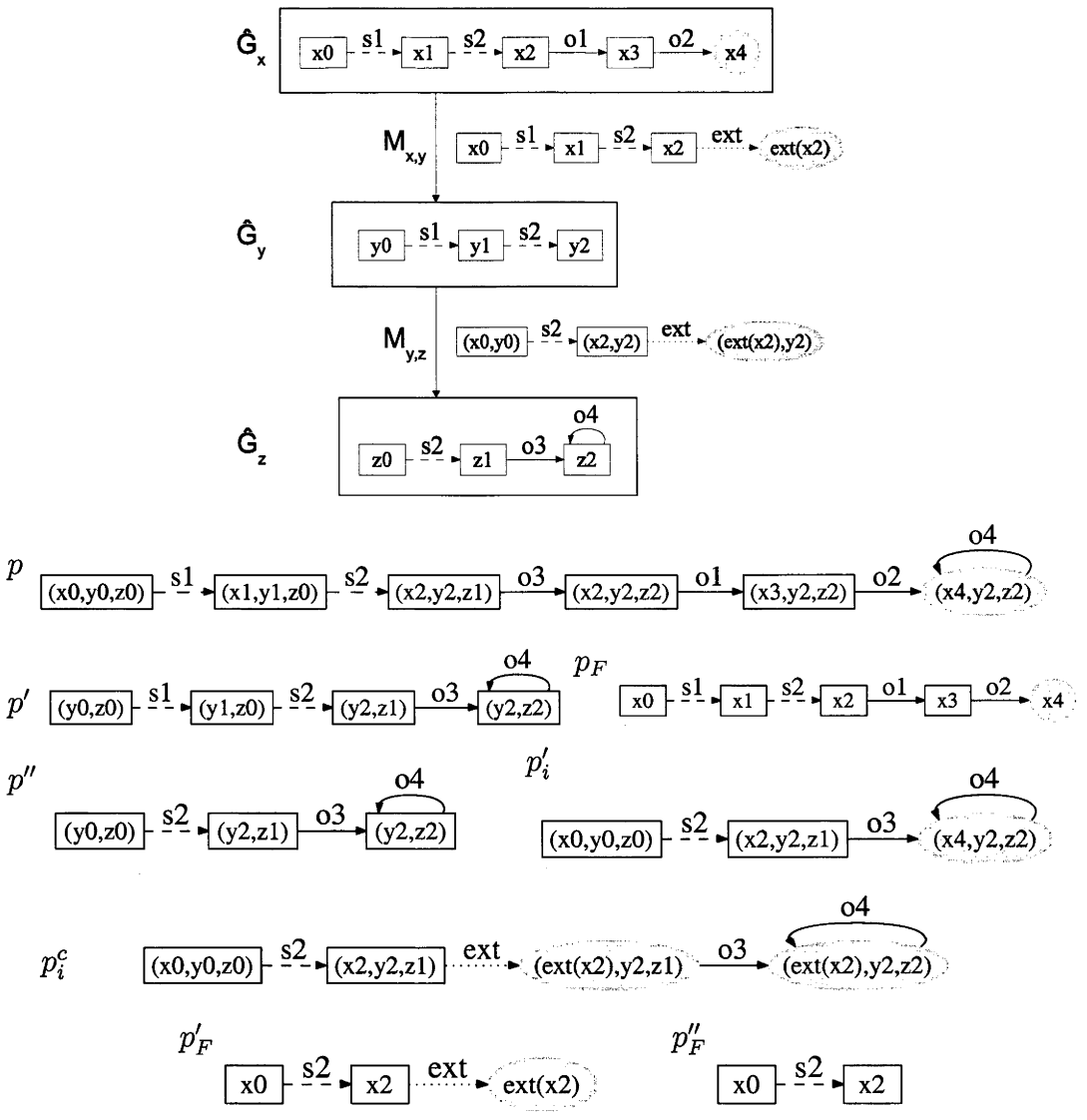


Figure 3.8: Example paths referred to in the proof.

In order for a path to be critical, there must also be a nondiagnosable state in the cycle. We now show that $\hat{G}'_i$ and $\hat{G}^c_i$ are also equivalent with respect to their critical paths. The proof is based on the recursive implication that the existence of one path leads to the existence of another path. To increase the readability we give an example of the paths we introduce in Figure 3.8. On the top of that Figure the jointree is depicted consisting of the three nodes $\hat{\hat{G}}_x$, $\hat{\hat{G}}_y$, and $\hat{\hat{G}}_z$ The two outward messages are also shown.

($\Rightarrow$) We now show that given the critical path p'_i in $\hat{G}'_i$ there must also be an equivalent critical path p_i in $\hat{G}^c_i$. Let p be a corresponding critical path in

the GTP= $Sync(\hat{G}_1, \ldots, \hat{G}_n)$, i.e. $p'_i = \Pi_{\Sigma_i}(p)$[3], that is composed of a path p' in $Sync(\hat{G}_1, \ldots, \hat{G}_{n-1})$ and a path p_F in $\hat{G}_F = \hat{G}_n$ such that p is a path in $Sync(p', p_F)$ (see Figure 3.8). This means that $p'' = \Pi_{\Sigma_i}(p')$ is also a path in $\hat{G}'_i$ equivalent to p'_i, since all events of p'_i are also in p'. This path p'' is diagnosable since none of its states is composed of a state in $\hat{G}_F$. Therefore there is a critical path p^c_i in $\hat{G}^c_i$, if the outward message $\vec{M}_{j,i}$ has a path that is composed of a path p'_F whose target state is nondiagnosable and such that $p''_F = \Pi_{\Sigma_i}(p_F) = \Pi_{\Sigma_i}(p'_F)$. Let $\Sigma_{p_F} \subseteq \Sigma_i$ denote the set of events of p''_F. Since From the jointree properties it then follows that every node along the path from $\hat{G}_F$ to $\hat{G}_i$ is labelled by an event set that contains Σ_{p_F}. Therefore $\vec{M}_{j,i}$ has a path composed of p'_F, if all the other outward messages along the jointree path from $\hat{G}_F$ to $\hat{G}_i$ are composed of such a path. This holds since p_F is a path in $\hat{G}_F$ ending in a nondiagnosable state and defined over the same event sequence as p'_F with respect to Σ_i. Therefore the outward message sent by $\hat{G}_F$ is guaranteed to have such a path p'_F leading to a (possibly extended) nondiagnosable state. Hence $\hat{G}^c_i$ has a critical path.

($\Leftarrow$) We now show that if there is a critical path p^c_i in $\hat{G}^c_i$, there is also one in $\hat{G}'_i$. Let p^c_i be composed of the paths p'_F and p'' in $\Pi_{\Sigma_i}(Sync(\hat{G}_1, \ldots, \hat{G}_{n-1}))$ such that $p'' \simeq p^c_i$ and p'_F is defined over events in $\Sigma_{p_F} \cup \{ext\}$ where $\{ext\} \notin \Sigma_{p_F}$. Resulting from the computation of outward messages this implies that there is a path p_F in $\hat{G}_F$ leading to a nondiagnosable state and such that $\Pi_{\Sigma_i \cap \Sigma_{p_F}}(p_F) = \Pi_{\Sigma_i \cap \Sigma_{p_F}}(p'_F)$. This means that there is a critical path in $Sync(p'', p_F)$ and therefore in $\Pi_{\Sigma_i}(Sync(Sync(\hat{G}_1, \ldots, \hat{G}_{n-1}), \hat{G}_n))$ and hence in $\hat{G}'_i$.

This concludes the proof of part (i). To prove part (ii) we need to show that the projection operation Π_Σ does not remove cycles, if at least one event $\sigma \in \Sigma$ of the cycle is kept in the projection. This results from the fact that the target state x of this event is kept in the projection. Hence, if the cycle is extended twice, i.e. as in the path $x_0 \xrightarrow{\sigma_1} x_1 \cdots \xrightarrow{\sigma_k} x \cdots \xrightarrow{\sigma} x \cdots \xrightarrow{\sigma} x$, its projection contains the state x twice and hence has a cycle. This state x is the same as in the original cycle and hence has the same diagnosability label. It implies that if there is a critical path in the GTP= $Sync(\hat{G}_1, \ldots, \hat{G}_n)$ with the observable event $\sigma \in \Sigma_i$ in the nondiagnosable cycle, then there is also a critical path in $\Pi_{\Sigma_i}(Sync(\hat{G}_1, \ldots, \hat{G}_n))$ and since part (i) holds there is also a critical path in the jointree node $\hat{G}^c_i$.

Now, from the existence of a critical path in a jointree node $\hat{G}^c_i$ it follows that

[3]The projection of paths is analogue to the one of FSMs (see page 94).

there is one in $\Pi_{\Sigma_i}(Sync(\hat{G}_1, \ldots, \hat{G}_n))$ (see part (i)). Since the projection operation does not add states and can only add a transition $x \xrightarrow{\sigma} x'$ if there has previously been a transition sequence starting in x and leading to x' it means that the projection operation does not add cycles. Therefore it follows that there is also a critical path in $Sync(\hat{G}_1, \ldots, \hat{G}_n)$ and hence in the GTP.

□

3.3.4 An Iterative Jointree Algorithm

Rather than propagating messages over the entire jointree, we now describe how we can improve efficiency and scalability by searching for a subset of it that is sufficient to decide diagnosability.

The basic idea is that any critical path p in the global twin plant can be detected by looking only at those twin plants that define events appearing on p, since every other twin plant has no impact on the behaviour represented by p. Our aim is to find a critical path defined over as few events as possible. The search for such a path will be done by iteratively increasing the set of jointree nodes (twin plants) $\hat{\mathbb{G}}$ under consideration, and looking for a critical path defined over only events Σ_{int} that are *internal* to $\hat{\mathbb{G}}$ (i.e., events that do not appear in the rest of the jointree). The detection of such a path establishes nondiagnosability and terminates the search.

Algorithm 5 gives the pseudo-code for this procedure. Our set of jointree nodes $\hat{\mathbb{G}}$ starts out containing just the root (*PickNode* on line 4 always returns the root the first time it is called), as the root is the only initial source of diagnosability information, without which no critical path can be detected in the other twin plants. At each iteration we select a new node that has a neighbour in $\hat{\mathbb{G}}$ (line 4), and add it to $\hat{\mathbb{G}}$ as well as update the set of internal events Σ_{int} (line 5). (We will discuss the node selection heuristic in the next subsection.)

Jointree propagation is then run twice:

1. On $\hat{\mathbb{G}}$ (line 6) to remove inconsistent paths. This can lead to the removal of nondiagnosable states which in turn may cause the root to become diagnosable ($HasNondiagState(root)$ is *false*) and thus verify diagnosability;

2. On $\hat{\mathbb{G}}_{\Sigma_{int}}$ (line 8) which is obtained by removing from all twin plants in $\hat{\mathbb{G}}$ the transitions labelled by events not in Σ_{int} (line 7). This allows to detect if a twin plant $\hat{G} \in \hat{\mathbb{G}}$ has a critical path whose global consistency can be verified by considering only the twin plants in $\hat{\mathbb{G}}$, since it does not contain

Algorithm 5 CheckDiagnosability(jointree: J)

```
1: Ĝ ← ∅                                              nodes in J being considered
2: Σ_int ← ∅                                          events internal to Ĝ
3: while Ĝ ≠ J and HasNondiagState(root) and SufficientMemory(Ĝ) do
4:   v ← PickNode(J, Ĝ)
5:   UpdateSets(v, Ĝ, Σ_int)
6:   Propagate(Ĝ)
7:   Ĝ_Σint ← GetAllPathsOverΣ_int(Ĝ)
8:   Propagate(Ĝ_Σint)
9:   if ExistsTwinPlantWithCritPath(Ĝ_Σint) then
10:    return GetCritPath(Ĝ_Σint)
11:  end if
12: end while
13: if SufficientMemory(Ĝ) then
14:   return "F is diagnosable"
15: else
16:   ω ← set of components included in Ĝ
17:   if ExistsTwinPlantWithCritPath(Ĝ) then
18:     return "ω has a critical path"
19:   else
20:     return "ω has no critical path"
21:   end if
22: end if
```

any event that appears in the rest of the tree. In this case, Algorithm 5 stops and returns the critical path that implies nondiagnosability (line 10).

The algorithm continues until one of the following conditions is satisfied:

- The root node (and hence the entire system) has been shown diagnosable. Note that it is indeed sufficient to check only the root node, since if the root has no nondiagnosable states, none of the messages it propagates and hence no twin plant includes a nondiagnosable state.

- The entire jointree is considered (and hence $J = \hat{\mathbb{G}} = \hat{\mathbb{G}}_{\Sigma_{int}}$), but none of the twin plants contains a critical path. This verifies the diagnosability of the system.

- The available resources have been exhausted (lines 16–21). In this case the maximal subsystem ω for which the existence of critical paths has been decided (but not yet verified against the rest of the system) is returned.

  Any critical path in ω can be interpreted as hint indicating nondiagnosability (of at least the isolated subsystem considered so far). In case critical paths exist in ω, then the larger this subsystem is, naturally, the more likely the

whole system is not diagnosable; otherwise the reverse is true. Such an approximate solution is also useful in that it implies that on-line monitoring of this particular subsystem will not be sufficient to reliably detect faults.

3.3.5 Jointree Node Selection

The heuristic used to select a jointree node to explore next can have a considerable impact on the number of nodes necessary to decide diagnosability. Instead of directly choosing a node, we first consider choosing an event which the new node might bring into Σ_{int}.

Let Σ_p denote the set of shared events appearing on a critical path p in some twin plant $\hat{G} \in \hat{\mathbb{G}}$. A reasonable heuristic is to expand Σ_{int} with some new event in $\Sigma_p \setminus \Sigma_{int}$ in the hope that p may at some point evolve into a new critical path that contains only internal events. To further focus the search, we will only consider events in $\Sigma_p \setminus \Sigma_{int}$ for paths p for which $|\Sigma_p \setminus \Sigma_{int}|$ is minimal.

Among these "eligible" events, we then select one that appears in the fewest nodes outside $\hat{\mathbb{G}}$. The idea here is to minimise the number of nodes that need to be included in $\hat{\mathbb{G}}$ for that event to be internal. After choosing an event, we iteratively add to $\hat{\mathbb{G}}$ the neighbouring nodes containing that event.

3.4 Further Enhancements and Modifications

The efficiency of the algorithm largely depends on the number and size of the messages propagated between jointree nodes. We now show how we can reduce both. The former is done by avoiding the exchange of messages that have already been passed on during a previous iteration and the latter by applying and modifying well-known techniques for the reduction of FSMs. Moreover we show how we can improve scalability by determining critical paths without requiring the synchronisation of all twin plants labelling a single jointree node. Finally we describe how our approach can be extended to decide the diagnosability of multiple faults at once.

3.4.1 Reduction of Message Sizes

The reduction of message sizes requires the distinction of two cases, namely the inward case in which the messages do not contain any diagnosability information

and the outward case where diagnosability information needs to be propagated correctly. While for the inward case we can apply well known reduction procedures for FSMs, we need to define new reduction rules for the outward case.

The Inward Case

There are efficient methods that compute a smaller equivalent FSM (see Condition 3.2 on page 96) from a given one based on merging indistinguishable states, that is, states that have the same past $Past$ or future Fut behaviour [Champarnaud & Coulon, 2004].

$$\begin{aligned} & Past(\hat{x}) = Past(\hat{x}') : \\ & \qquad \exists \tau = \hat{x}_0 \xrightarrow{\sigma_1} \hat{x}_1 \cdots \xrightarrow{\sigma_k} \hat{x} \text{ in } \hat{G} \\ \leftrightarrow \quad & \qquad \exists \tau' = \hat{x}'_0 \xrightarrow{\sigma_1} \hat{x}'_1 \cdots \xrightarrow{\sigma_k} \hat{x}' \text{ in } \hat{G} \end{aligned} \tag{3.3}$$

$$\begin{aligned} & Fut(\hat{x}) = Fut(\hat{x}') : \\ & \qquad \exists \tau = \hat{x} \xrightarrow{\sigma_1} \hat{x}_1 \cdots \xrightarrow{\sigma_k} \hat{x}_k \text{ in } \hat{G} \\ \leftrightarrow \quad & \qquad \exists \tau' = \hat{x}' \xrightarrow{\sigma_1} \hat{x}'_1 \cdots \xrightarrow{\sigma_k} \hat{x}'_k \text{ in } \hat{G} \end{aligned} \tag{3.4}$$

Any pair of states $(\hat{x}, \hat{x}')$ satisfying one of these two conditions can be merged by deleting $\hat{x}'$, redirecting all incoming transitions of $\hat{x}'$ to $\hat{x}$, and repositioning the outgoing transitions of $\hat{x}'$ so that they originate from $\hat{x}$. For example, in the message sent from $\hat{G}_2$ to $\hat{G}_1$ shown on the top left of Figure 3.9 states $y0$ and $y5$ have the same future behaviour and furthermore we have $Past(y1) = Past(y6)$ and $Past(y3) = Past(y7)$. The merge of these three indistinguishable state pairs leads to the FSM shown on the top right of that Figure. In the message sent from $\hat{G}_3$ to $\hat{G}_1$ (see bottom left) all states have the same future behaviour and can therefore be merged.

Note that the reduction of messages is also a key operation in order to obtain small consistent twin plants. For example passing on the two inward messages shown on the left of Figure 3.9 to the original twin plant $\hat{G}_1$ we obtain a FSM with 26 states, whereas passing on the reduced messages to $\hat{G}_1$ we obtain a FSM with only 8 states.

The reduction of a FSM by merging indistinguishable states may not lead to a unique FSM in general. Fortunately there are good merging heuristics available, as well as efficient methods to identify mergeable states. For example, the method of [Champarnaud & Coulon, 2004] runs in $\mathbb{O}(|X| \times |T|)$. Note that the reduction to the smallest equivalent FSM is PSPACE-complete [Meyer & Stockmeyer, 1972],

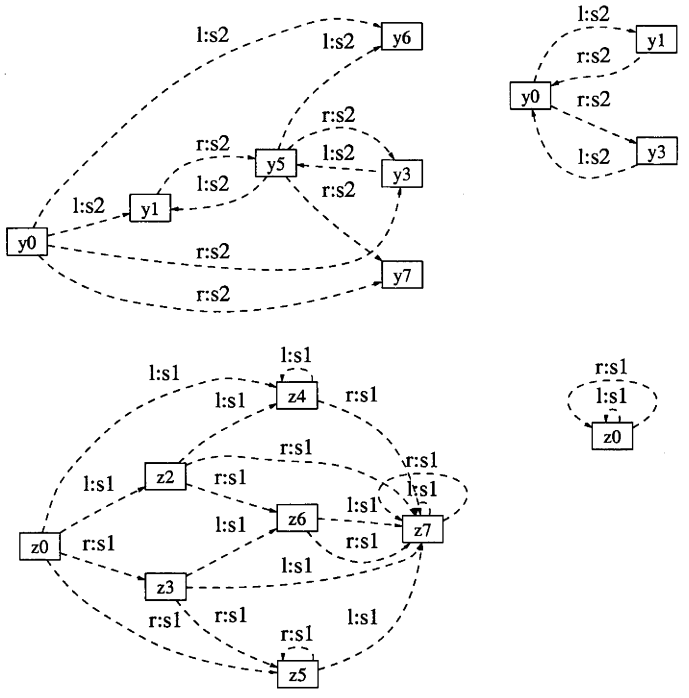


Figure 3.9: Reduced inward messages for our example.

unless the FSM is deterministic, in which case it can be minimised in $\mathbb{O}(m \log m)$ where $m = |X| \times |\Sigma|$ [Hopcroft & Ullman, 1979]. This lower complexity is due to that all states in a deterministic FSM have different pasts, and hence only Fut-indistinguishable states are merged. Since this condition is transitive, the resulting FSM is independent of the order in which states are merged, leading to efficient minimisation.

The merge of states is an essential operation to obtain *reduced* FSMs.

Definition 26 *A FSM G is* reduced *if it satisfies the following conditions:*

- *G has no two states satisfying condition 3.3, and*
- *G has no two states satisfying condition 3.4.*

In the following we will denote with $\mathbb{R}(G)$ a reduced FSM obtained from G by iteratively merging all states that satisfy Condition 3.3 or 3.4 until no more states can be merged.

The Outward Case

On the one hand, the outward case requires a more restrictive merging condition since we also need to deal with the correct propagation of diagnosability informa-

tion. On the other hand, the removal of diagnosable states is less restrictive if they are not part of a transition sequence containing nondiagnosable states.

When reducing the size of a FSM in this case we need to ensure that the critical paths are preserved. This is the case if we prune only *irrelevant* states from the twin plants and if the states we merge satisfy certain conditions.

Relevant Twin Plant States

A state in a twin plant is *relevant* if it can possibly be on a critical path. This is the case if it is on a path whose target state is nondiagnosable. For the purpose of deciding diagnosability all other states are irrelevant and can therefore be removed (see proof of Theorem 3.4.1 on page 112), resulting in the relevant part $rel(\hat{G})$ of a twin plant $\hat{G}$. The relevant twin plant part for our example is shown in Figure 3.10.

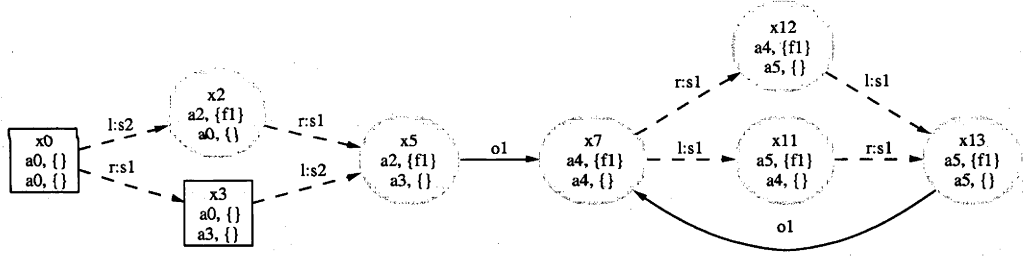


Figure 3.10: Relevant part of the twin plant shown in Figure 3.2 that contains only relevant states.

Condition for state merging

To ensure that state merges in twin plants do not remove all critical paths we define the two merging criteria $Past_D$ and Fut_D as follows:

$$\begin{aligned} & Past_D(\hat{x}) = Past_D(\hat{x}') : \\ & \qquad \exists \tau = \hat{x}_0 \xrightarrow{\sigma_1} \hat{x}_1 \cdots \xrightarrow{\sigma_k} \hat{x} \text{ in } \hat{G} \\ \leftrightarrow \quad & \qquad \exists \tau' = \hat{x}'_0 \xrightarrow{\sigma_1} \hat{x}'_1 \cdots \xrightarrow{\sigma_k} \hat{x}' \text{ in } \hat{G} \end{aligned} \tag{3.5}$$

$$
\begin{aligned}
& Fut_D(\hat{x}) = Fut_D(\hat{x}') : \\
& \qquad (\exists \tau = \hat{x} \xrightarrow{\sigma_1} \hat{x}_1 \cdots \xrightarrow{\sigma_k} \hat{x}_k \text{ in } \hat{G} \\
\leftrightarrow \quad & \qquad \exists \tau' = \hat{x}' \xrightarrow{\sigma_1} \hat{x}'_1 \cdots \xrightarrow{\sigma_k} \hat{x}'_k \text{ in } \hat{G}) \wedge \\
& \qquad \left(\bar{D}(\hat{x}) \leftrightarrow \bar{D}(\hat{x}')\right) \wedge \left(\bar{D}(\hat{x}_1) \leftrightarrow \bar{D}(\hat{x}'_1)\right) \ldots \wedge \left(\bar{D}(\hat{x}_k) \leftrightarrow \bar{D}(\hat{x}'_k)\right)
\end{aligned}
\tag{3.6}
$$

Thus the condition for merging states with the same past behaviour is identical to the one in Section 3.4.1. Also the merge of two states is the same as described before with two additional rules:

- When merging two *Past*-indistinguishable states with different diagnosability label, the state to be deleted is the diagnosable one.
- When merging two diagnosable *Past*-indistinguishable states of which only one has an extended state, the state to be deleted is the one that does not have an extended state.

The first rule originates from the fact that we need to ensure that the critical paths are kept which all have a nondiagnosable state. The second rule is added so that extended states for messages computed based on D-reduced twin plants (see below) can be correctly obtained.

In contrast, Fut_D-indistinguishable states are required to have the same diagnosability label. Suppose $\hat{x}$ is diagnosable, $\hat{x}'$ is nondiagnosable and we would label $mgr(\hat{x}, \hat{x}')$ nondiagnosable. This would result in a path leading to a nondiagnosable state that was not previously in $\hat{G}$. On the other hand, if $mgr(\hat{x}, \hat{x}')$ would be labelled diagnosable it could lead to the removal of a previously existing path with a nondiagnosable state and thus possibly to the removal of a critical path resulting in the wrong diagnosability verification.

The merging strategies for the outward case are the same as described earlier. Again such a reduction may not lead to a unique FSM. Consider for example the projection $\mathcal{G}$ shown at the top left of Figure 3.11. Here we have $Fut_D(x5) = Fut_D(x13)$ whose merging leads to the reduced FSM shown to its right. On the other hand we also have $Past_D(x3) = Past_D(x5)$ in $\mathcal{G}$ whose merging leads to the FSM shown at the bottom left. This FSM is not yet reduced since we have $Fut_D(x5) = Fut_D(x13)$. Merging these states we arrive at the smallest FSM equivalent to $\mathcal{G}$ shown at the bottom right of that same Figure. Note, as in the inward case, there is no guarantee that the iterative merge of states will lead to

the smallest FSM. The problem of computing the smallest one is again PSPACE-complete.

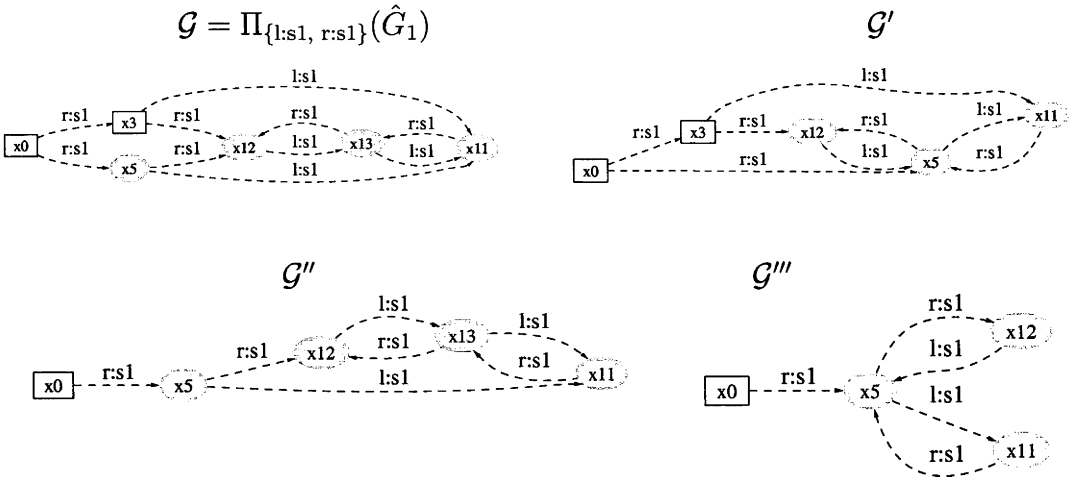


Figure 3.11: A FSM $\mathcal{G}$ and three smaller equivalent representations.

D-reduced FSMs

The merge of states is an essential operation to obtain D-reduced twin plants.

Definition 27 *A FSM $\hat{G}$ is* D-reduced *if it satisfies the following conditions:*

- *$\hat{G}$ contains only relevant states,*
- *$\hat{G}$ has no two states satisfying condition 3.5, and*
- *$\hat{G}$ has no two states satisfying condition 3.6.*

In the following we will denote with $\mathbb{R}_D(\hat{G})$ a D-reduced twin plant obtained from $\hat{G}$ by executing below steps in sequence:

1. Removal of all states from $\hat{G}$ that are not relevant;
2. Computation of $\mathcal{P} = \Pi_{\hat{\Sigma}}(\hat{G})$; and
3. Computation of $\mathbb{R}_D(\hat{G})$ obtained from iteratively merging all states in $\mathcal{P}$ that satisfy Condition 3.5 or 3.6 until no more states can be merged.

Figure 3.12 illustrates the reduction of the twin plant $\hat{G}$. Here $\mathcal{P}$ contains only two indistinguishable states namely $(m1, q3)$ and $(ext(m1), q3)$ which have the same past. To show that the reduction operation is indeed valid we prove the following theorem:

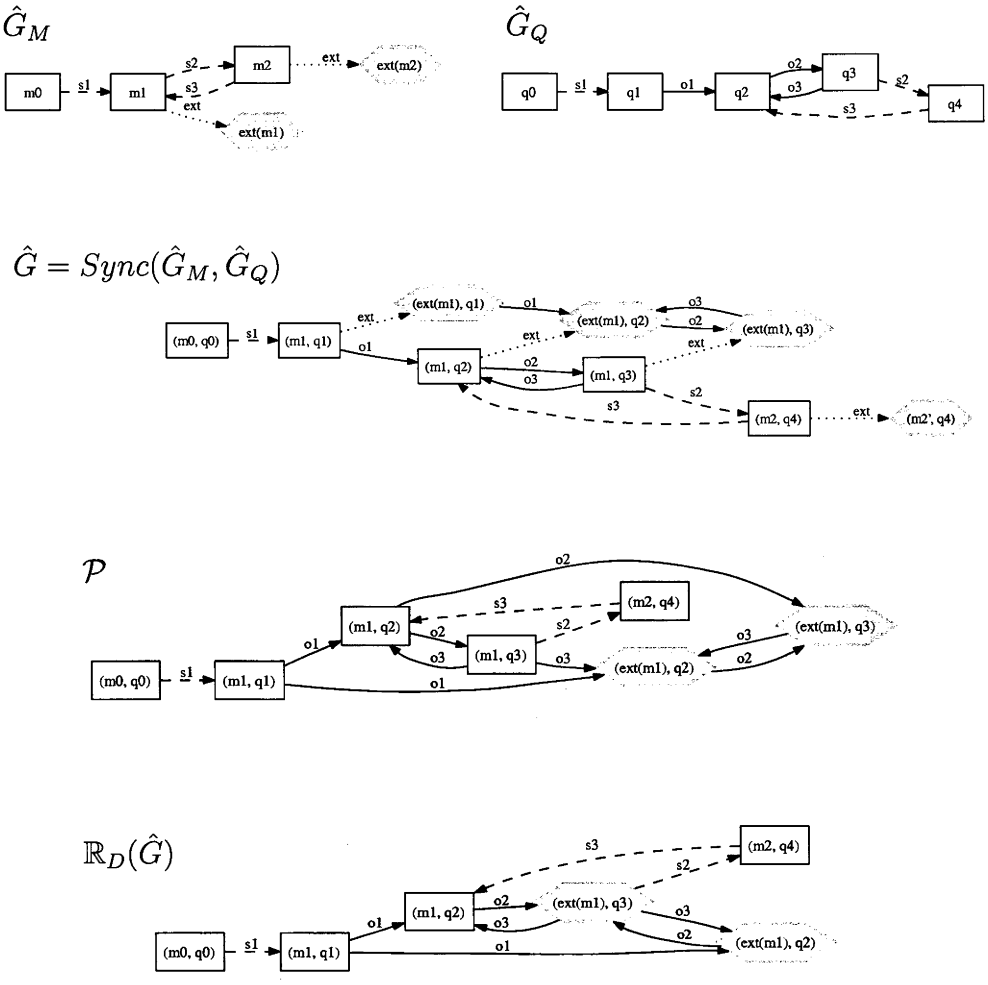


Figure 3.12: Reduction of $\hat{G}$

Theorem 3.4.1 *A twin plant $\hat{G}$ has a critical path iff every twin plant $\mathbb{R}_D(\hat{G})$ has a critical path.*

Proof: We prove this Theorem by showing that none of the three steps to compute $\mathbb{R}_D(\hat{G})$ (see page 111) can lead to the removal or introduction of a critical path.

(1) We show the correctness of removing irrelevant states by contradiction.

Without loss of generality, let $p = \hat{x}_0 \xrightarrow{\sigma_1} \hat{x}_1 \cdots \xrightarrow{\sigma_k} \hat{x}_k$ be a critical path in $\hat{G}$ where $\hat{x}_k \in \{\hat{x}_0, \ldots, \hat{x}_{k-1}\}$ is a nondiagnosable state in a cycle. Suppose now that there is no critical path in $rel(\hat{G})$, the relevant part of $\hat{G}$. Since state $\hat{x}_k$ is nondiagnosable, all states of p are on a path whose target state is nondiagnosable and are therefore relevant. Thus p cannot be removed by the removal of irrelevant states.

Since $rel(\hat{G})$ is obtained from $\hat{G}$ by removing states and transitions every

path in $rel(\hat{G})$ must be in $\hat{G}$. Hence in particular any critical path in $rel(\hat{G})$ must be in $\hat{G}$.

(2) Computing the projection $\mathcal{P} = \Pi_{\hat{\Sigma}}(\hat{G})$ cannot lead to the introduction of a critical path since projection does not introduce states or cycles. Furthermore, the projection operation cannot lead to the removal of a critical path, since the only event that is abstracted is *ext* and since extended transitions are added as terminal transitions and are therefore never part of a cycle.

(3) Suppose the merge of a state $\hat{x}_i \in \{\hat{x}_0, \ldots, \hat{x}_k\}$ on the critical path p (see (1)) with a state $\hat{x}'_i$ led to the removal of all critical paths in the resulting twin plant $mgr(\hat{G})$. Therefore the path $p' = \hat{x}_0 \xrightarrow{\sigma_1} \hat{x}_1 \cdots \xrightarrow{\sigma_{i-1}} \hat{x}_{i-1} \xrightarrow{\sigma_i} \hat{x}'_i \cdots \xrightarrow{\sigma_k} \hat{x}_k$ in $mgr(\hat{G})$ is not critical (p' is in $mgr(\hat{G})$ since all incoming and outgoing transitions of $\hat{x}_i$ were redirected to $\hat{x}'_i$ in the process of the state merge). This means that the cycle of p' no longer has a nondiagnosable state, which implies that state $\hat{x}'_i$ is diagnosable. Thus the merge of a diagnosable and a nondiagnosable state led to the removal of the nondiagnosable state. This contradicts all merging conditions (see Section 3.4.1).

Suppose now that there is no critical path in $\hat{G}$ but one in $mgr(\hat{G})$. The merging conditions ensure that for every path $p = \hat{x}_0 \xrightarrow{\sigma_1} \hat{x}_1 \cdots \xrightarrow{\sigma_k} \hat{x}_k$ resulting from a merge of states, there exists a corresponding path $p' = \hat{x}_0 \xrightarrow{\sigma_1} \hat{x}'_1 \cdots \xrightarrow{\sigma_k} \hat{x}'_k$ in $\hat{G}$ such that the pairwise corresponding states have the same diagnosability label (i.e. $\bar{D}(\hat{x}_i) \leftrightarrow \bar{D}(\hat{x}'_i)$ for all $\hat{x}_i \in \{\hat{x}_1, \ldots, \hat{x}_k\}$). Thus, if $\hat{G}$ has no critical path there cannot be one in $\mathbb{R}_D(\hat{G})$. □

Hence we can indeed reduce the size of FSMs while still being able to decide diagnosability. In order to ensure the correct propagation of diagnosability information it is again necessary to add the extended states and transitions after the reduction (see page 99).

3.4.2 Reduction of Message Propagations

We now describe the conditions under which we can reduce the number of messages being propagated. As described in our core algorithm on page 105 the propagation is repeated every time a new node v is added to set $\hat{\mathbb{G}}$. The question we now consider is to what extend the results of the previous propagation on set $\hat{\mathbb{G}}_{\bar{v}} = \hat{\mathbb{G}} \setminus \{v\}$ can be utilised.

Due to the jointree node selection heuristic (see Section 3.3.5) v is always a leaf of the subtree $\hat{\mathbb{G}}$ and hence the inward propagation starts here. Now if the parent $Par(v)$ of v, which is already guaranteed to be consistent with $\hat{\mathbb{G}}_{\bar{v}}$, is also consistent with v, we need not change its behaviour. This implies that the messages now exchanged within nodes of $\hat{\mathbb{G}}_{\bar{v}}$ are exactly the same as the ones exchanged in the previous propagation. Hence in that case $\hat{\mathbb{G}}$ is guaranteed to be consistent once v has received the outward message from its parent.

This reasoning can be generalised and adopted also to the outward propagation as shown in Algorithms 6 and 7. The first one describes the inward propagation that is repeated until either a node v' is reached that was already consistent with v (i.e. $v' \simeq \Pi_{\hat{\Sigma}_{v'}}(Sync(v, v'))$) or until the root is reached (line 5). This means that the number of inward messages propagated cannot exceed the depth of v, that is, the number of edges on the path from the root to v. Hence, even in the worst case, the number can be exponentially smaller than the number of messages generally exchanged during the inward propagation.

Algorithm 6 Propagation(jointree: J, set of consistent nodes: $\hat{\mathbb{G}}_{\bar{v}}$, node $v \notin \hat{\mathbb{G}}_{\bar{v}}$)

1: $\hat{\mathbb{G}} \leftarrow \hat{\mathbb{G}}_{\bar{v}} \cup \{v\}$ — nodes in J being considered
2: $\hat{\mathbb{G}}_{ch} \leftarrow \emptyset$ — nodes changed during inward propagation
3: $v' \leftarrow par(v)$ — $par(v)$ returns the parent of v
4: $\widetilde{v}' \leftarrow InProp(v, v')$
5: **while** v' and $\widetilde{v}'$ are not equivalent and are not the root **do**
6: $\quad \hat{\mathbb{G}}_{ch} \leftarrow \hat{\mathbb{G}}_{ch} \cup \{\widetilde{v}'\}$
7: $\quad v' \leftarrow par(\widetilde{v}')$
8: $\quad \widetilde{v}' \leftarrow InProp(\widetilde{v}', v')$
9: **end while**
10: $OutPropProc(J, v', \hat{\mathbb{G}}_{ch})$

Function $InProp(j, j')$ returns the twin plant for j' once it has received the message from its child j.

In order to also reduce the number of the messages exchanged during the outward propagation we memorise all nodes $\hat{\mathbb{G}}_{ch}$ that have changed during the inward propagation. Now starting with the last node v' considered during the inward propagation we perform the outward propagation recursively as described in Algorithm 7. Here we only propagate messages outwardly from those nodes that have either changed during the inward or outward propagation (line 4) which again reduces the number of messages that need to be exchanged.

Algorithm 7 OutPropProc(jointree: J, jointree node v', $\hat{\mathbb{G}}_{ch}$)

1: **if** p is not a leaf of J **then**
2: **for all** successors v'_s of v' **do**
3: $\widetilde{v}'_s \leftarrow OutProp(v', v'_s)$
4: **if** $v'_s \in \hat{\mathbb{G}}_{ch}$ or $\widetilde{v}'_s$ is not equivalent to v'_s **then**
5: $OutPropProc(J, \widetilde{v}'_s, \hat{\mathbb{G}}_{ch})$
6: **end if**
7: **end for**
8: **end if**

Function $OutProp(j, j')$ returns the twin plant for j' once it has received the message from its parent j.

3.4.3 Improvement of Scalability

So far all our jointree algorithms are based on the assumption that all twin plants associated to a single jointree node have been synchronised (see first assumption on page 98). Even if this is feasible, the space required to represent all these synchronised twin plants can limit our ability to perform the iterative jointree algorithm. We therefore drop this assumption and consider now a jointree in which each node is labelled with a set of twin plants each referring to exactly one component.

After a node v is added to the set of considered nodes $\hat{\mathbb{G}}$ and the internal events Σ_{int} are updated (see lines 4–5 in Algorithm 5 on page 105) we synchronise only those twin plants of v that have an event in Σ_{int}. While this approach is not sufficient to guarantee the consistency of all twin plants labelling nodes in $\hat{\mathbb{G}}$ (the propagation on line 6 can no longer be performed), it is sufficient for checking the consistency of a path entirely composed of events in Σ_{int} and for achieving consistency of twin plants in $\hat{\mathbb{G}}_{\Sigma_{int}}$. Thus, the partial synchronisation of twin plants labelling a single jointree node allows faster detection of a critical path and thus faster verification of nondiagnosability.

Note that for the stepwise synchronisation of twin plants we can also adopt the same heuristics for selecting a jointree node as described in Section 3.3.5. Instead of considering the critical paths of the synchronised twin plants, we now look at the whole set of twin plants labelling a node.

3.4.4 Diagnosability of a System

So far we have shown how we can verify the diagnosability of a single fault. Now we consider the diagnosability verification for a whole system.

Definition 28 *A system G is diagnosable iff all its faults $F \in \Sigma_f$ are diagnosable.*

We can modify our algorithm to consider all faults at once. For the jointree this means that diagnosability information can be found in more than one node, which has the following implications:

- We need to decide which of these nodes we should select as root of the jointree.

  This can again be done using the heuristics described in Section 3.3.5.

- We can only merge states satisfying equations 3.3-3.6.

  Equivalence is now also required, since the diagnosable behaviour that is irrelevant for the diagnosability test of the faults occurring in the considered twin plant can be essential to detecting the consistency of critical paths in other twin plants.

- We need to perform an inward and outward propagation without propagating diagnosability information, i.e. without introducing extended states.

  This results from the fact that diagnosability information can only be propagated based on consistent twin plants and after the inward propagation only the root is guaranteed to be consistent.

- Afterwards we need to perform another inward and outward propagation, this time with the purpose of propagating diagnosability information.

  Now we require a double propagation, since diagnosability information is *exchanged* among all the jointree nodes rather than propagated from a single node.

Compared to the single-fault approach we have the overhead of additional propagations (now 4 instead of 2), as well as an increase in complexity due to an increase in the size of the twin plants. The space complexity of the interactive diagnoser is $\mathbb{O}(|X_i| \times 2^{|\Sigma_{f_i}|})$ and hence that of the twin plant is $\mathbb{O}(|X_i|^2 \times 2^{2|\Sigma_{f_i}|})$ [Jiang *et al.*, 2001]. Clearly, if the number of faults is high we now face a significant increase in space complexity. Thus it is generally better to check the diagnosability individually for each fault by running our one fault algorithm $|\Sigma_f|$ times.

3.5 Relation to Previous Work

In this section we show in detail how the approach of [Pencolé, 2004], the closest to ours, can be simulated with jointrees, and how it relates to our algorithm. Specifically we show that it amounts to a restricted way of jointree construction, together with a particular way of achieving consistency, which does not take advantage of the propagation of messages with bounded event sets. Towards the end of the section we also discuss other related work.

The *ClassDecent* approach of [Pencolé, 2004] also solves the diagnosability problem in a decentralised way. The approach is based on the assumption that the observable behaviour of every component is live, that is, that there is no component with a cycle containing only unobservable events. This is more restrictive than the assumption in [Sampath *et al.*, 1995] which we adopted namely that the observable behaviour of the system (but not necessarily that of individual components) is required to be live (see page 88). The more restrictive assumption of the *ClassDecent* approach implies that it is sufficient to only search for a critical path in the twin plant $\hat{G}_F$ containing the fault. To compare the efficiency of our approach to this one we also restrict the search for a critical path to $\hat{G}_F$. This means that we only need to compute consistent paths of the jointree root which can be done by the straightforward inward propagation.

Starting with the twin plant $\hat{G} = \hat{G}_F$ the approach *ClassDecent* proceeds iteratively by selecting a twin plant $\hat{G}_i$ from the not yet considered ones in $\hat{\mathbb{G}}_{out}$ (line 2 of the pseudocode below), synchronising it with $\hat{G}$ (line 4) and removing from it (by projection) the shared events Σ_{int} internal to $\hat{G}$ (line 5). This process is repeated until either a critical path defined only over the observable events is found[4] in which case nondiagnosability is established or until $\hat{G}$ has no critical path or $\hat{\mathbb{G}}_{out}$ has no connected twin plant (see below) which both verifies diagnosability (line 1).

1: **while** $HasCritPath(\hat{G})$ and $HasNoCritPathOver\Sigma_o(\hat{G})$ and $Has\hat{G}consTP(\hat{\mathbb{G}}_{out})$ **do**
2: $\quad\hat{G}_i \leftarrow PickTwinPlant(\hat{\mathbb{G}}_{out})$
3: $\quad UpdateSet(\hat{\mathbb{G}}_{out}, \Sigma_{int})$
4: $\quad\hat{G} \leftarrow Sync(\hat{G}, \hat{G}_i)$
5: $\quad\hat{G} \leftarrow \Pi_{\hat{\Sigma}\setminus\Sigma_{int}}(\hat{G})$
6: **end while**

[4]Since the internal events have been removed from $\hat{G}$ such a path corresponds to a critical path in the global twin plant defined only over observable and internal events.

Thus, the approach requires essentially two operations during each iteration: projection and synchronisation. Recall that these are exactly the same steps we need to perform during the inward propagation. Indeed the *ClassDecent* approach can be thought of as a special way of using jointrees. Such a jointree $\mathcal{J}$ can be constructed based on the connectivity of twin plants with respect to $\hat{G}_F$.

Definition 29 (α-connectivity $Con(\alpha, \gamma)$) *The set of twin plants connected to the twin plant γ by distance α is recursively defined as follows.*

$Con(0, \gamma) = \{\gamma\}$

$Con(\alpha, \gamma) = \{\gamma_1 \mid \exists \gamma_2 \in Con(\alpha - 1, \gamma)$ *such that*
γ_1 *and* γ_2 *share at least one event and*
$\gamma_1 \notin Con(\beta, \gamma)$ *for all* $\beta < \alpha\}$.

Twin plants γ_1 and γ_2 are *connected* if $\gamma_1 \in Con(1, \gamma_2)$. The set $transCon(\gamma)$ denotes the set of twin plants whose behaviour can possibly influence that of γ ($transCon(\gamma) = \bigcup_{\alpha=0}^{n-1} Con(\alpha, \gamma)$) where n is the number of components in the system. Now the approach only proceeds as long as there is a twin plant in $\hat{\mathbb{G}}_{out}$ that is connected to $\hat{G}$. Therefore $\hat{G}$ is at most composed of the synchronisation of all k elements in $transCon(\gamma)$. This is similar to our approach where we also do not need to consider any further twin plants to check the global consistency of critical paths in the root twin plant.

To create the jointree $\mathcal{J}$ with k nodes, one for each twin plant in $transCon(\gamma)$, we can use the same twin plant selection heuristics as function *PickTwinPlant*. Let the order in which the latter picks the twin plants be $\hat{G}_1, \ldots, \hat{G}_k$, that is, in the i^{th} loop iteration $\hat{G}$ is synchronised with $\hat{G}_i$. In the same order we now also add the corresponding nodes to $\mathcal{J}$. Initially $\mathcal{J}$ consists only of the root $\hat{G}_F = \hat{G}_1$. Now in the i^{th} step node $\hat{G}_i$ is added by linking it to an existing node $\hat{G}_j \in Con(1, \hat{G}_i)$ with minimal distance to $\hat{G}_F$.

Figure 3.13 shows the resulting tree for an example in which the four nodes $Con(1, \hat{G}_F) = \{\hat{G}_2, \hat{G}_4, \hat{G}_6, \hat{G}_7\}$ are connected to $\hat{G}_F$. Further we have $Con(2, \hat{G}_F) = \{\hat{G}_3, \hat{G}_5, \hat{G}_8, \hat{G}_9\}$ and $Con(3, \hat{G}_F) = \{\hat{G}_{10}, \hat{G}_{11}, \hat{G}_{12}\}$. Now, for instance, if the node for $\hat{G}_5$ is added it could either be linked to $\hat{G}_2$ or $\hat{G}_3$ to which it is both connected via event k. Since the distance of $\hat{G}_2$ to $\hat{G}_F$ is smaller (1 instead of 2 for $\hat{G}_3$), the new node is linked to $\hat{G}_2$.

However, this tree does not yet satisfy the jointree properties (see Definition 22 on page 92): we need to ensure that every path between two nodes containing the same event σ is only composed of nodes that also contain σ. In our example twin plants $\hat{G}_3$ and $\hat{G}_9$ are labelled with event r but none of the nodes on their

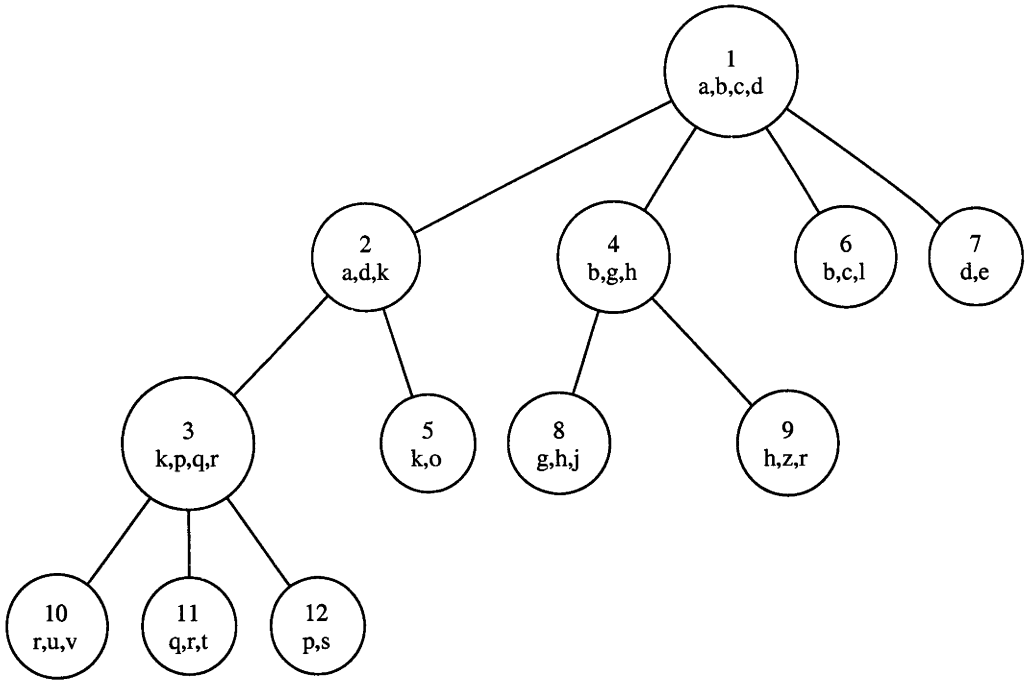


Figure 3.13: Tree in which the top labels of each node $1, \ldots, 12$ correspond to the twin plants $\hat{G}_1, \ldots, \hat{G}_{12}$ and the bottom labels of each node refer to the events defined in it.

connecting path $\hat{G}_3 - \hat{G}_2 - \hat{G}_1 - \hat{G}_4 - \hat{G}_9$. We therefore have to add event r to nodes $\hat{G}_2$, $\hat{G}_1$ and $\hat{G}_4$, after which above property is satisfied. Now we can also add the separator events to every edge between two nodes. These events are simply the intersection of the event sets of these two nodes. Figure 3.14 illustrates the jointree for our example.

The *ClassDecent* approach now amounts to a particular way of achieving consistency on the constructed jointree $\mathcal{J}$. Instead of verifying the consistency among jointree nodes via message propagation, this approach synchronises all considered twin plants $\hat{\mathbb{G}}$, which corresponds to a merge of jointree nodes. To merge two jointree nodes $\hat{G}_i$ and $\hat{G}_j$, their event sets Σ_i and Σ_j are added and all internal events Σ_{int} removed (i.e. the projection $\Pi_{(\Sigma_i \cup \Sigma_j) \setminus \Sigma_{int}}(Sync(\hat{G}_i, \hat{G}_j))$ is computed as in line 5 of the Algorithm shown on page 117). The set Σ_{int} is composed of those events that were only labelling the former edges between the now merged nodes. For instance, after the merging in the first iteration of nodes $\hat{G}_1$ and $\hat{G}_2$ of the jointree shown in Figure 3.14 the event set of the new root is composed of b, c, d, k, r. Since event a did not appear anywhere else on the jointree it was removed. Figure 3.15 depicts the jointree after the fourth iteration of the *ClassDecent* algorithm, i.e. after the twin plants $\hat{G}_1, \hat{G}_2, \hat{G}_3$ and $\hat{G}_4$ have been synchronised and their internal events removed (by projection).

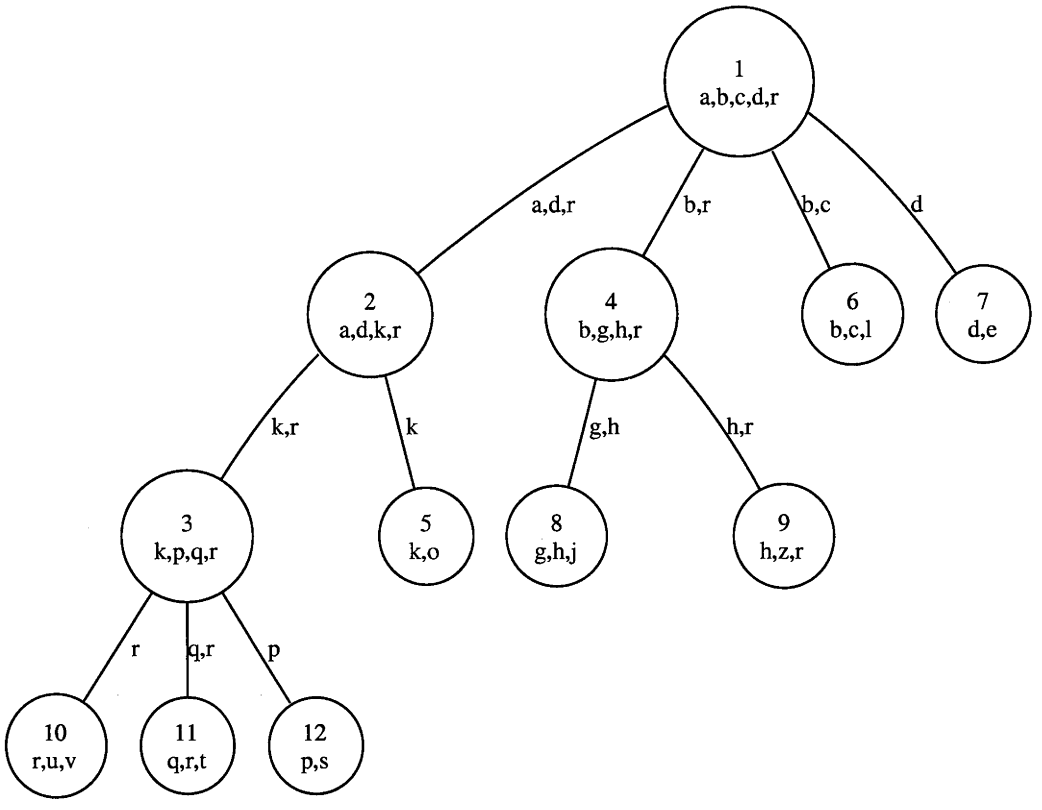


Figure 3.14: Jointree for the tree shown in Figure 3.13.

Recall that our approach does not change the jointree structure during the diagnosability verification. In particular this means that the size of the event sets of the considered twin plants $\hat{\mathbb{G}} = \hat{G}_1, \ldots, \hat{G}_q$ is bounded. Thus also when using the same jointree as the *ClassDecent* procedure, in which every jointree node is labelled by exactly one twin plant, we are guaranteed that every twin plant $\hat{G}_i$ has at most $w+1$ events, where w corresponds to the width of the jointree. In contrast, the *ClassDecent* procedure always considers a twin plant with $|\Sigma_{o_1} \cup \Sigma_{o_2} \ldots \cup \Sigma_{o_q}|$ observable and $|\Sigma_{sep_1} \cup \Sigma_{sep_2} \ldots \cup \Sigma_{sep_m}|$ shared events, where Σ_{sep_i} are the separator events connecting a twin plant in $\hat{\mathbb{G}}$ with one in $\{\hat{G}_{j_1}, \ldots, \hat{G}_{j_m}\} \subseteq \hat{\mathbb{G}}_{out}$.

It is exactly this bound on event size that likely allows our approach to perform the synchronisation and projection operations more efficiently in general, even without considering the additional features such as the reduction of message size. This results from the complexity of searching for a critical path in a twin plant which depends on the number of transitions and thus on the number of events and states $\mathbb{O}(|\Sigma| \times |\hat{X}|^2)$. Note that due to the projection operation that is performed in both approaches the number of events has also a direct impact on the number of states, since only those states are retained in the resulting twin plant that are a target state of a transition labelled by an event in the projection.

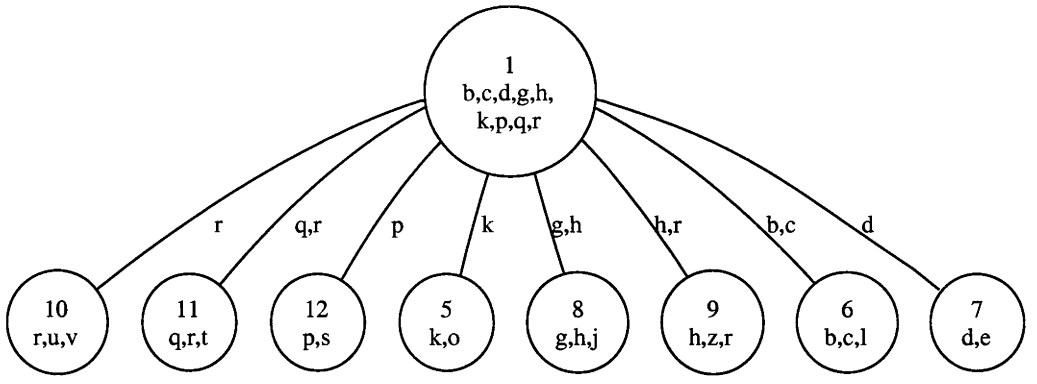


Figure 3.15: Jointree of Figure 3.14 after the merge of nodes $1-4$.

We now discuss other related work. The diagnosability problem of discrete-event systems was introduced in [Sampath *et al.*, 1995] where the authors solved it by considering a deterministic diagnoser for the global system and a part of the global model. The main drawback of this method is its exponential space complexity in the number of system states resulting from the diagnoser whose size is exponential in the number of states in the global model (determination) and therefore doubly exponential in the number of system components.

Jiang *et al.* (2001) and Yoo & Lafortune (2002) then propose new algorithms which are only polynomial in the number of states in G and which introduce the twin plant method. The question of efficiency is raised in [Cimatti, Pecheur, & Cavada, 2003] where the authors propose to use symbolic model-checking to test a restrictive diagnosability property by taking advantages of efficient model-checking tools. But, still the diagnosability problem is seen as a test on a system whose size is exponential in the number of components, even when encoded by means of binary decision diagrams as in [Cimatti, Pecheur, & Cavada, 2003]. Some of the most recent works decide either diagnosability or nondiagnosability but not both. The work by [Rintanen & Grastien, 2007] shows how to search for critical paths using SAT thus verifying nondiagnosability. On the other hand, our previous decentralised approach can only verify diagnosability [Schumann & Pencolé, 2007].

3.6 Assisting in the Design of Diagnosable Systems

We have shown how we can efficiently decide diagnosability based on a jointree and how we retrieve a critical path in case the system is nondiagnosable. However the ultimate aim is not just to decide diagnosability, but rather to develop diagnosable systems. If the system is not diagnosable additional sensors are required to distinguish the ambiguous system behaviours. Several approaches deal with the problem of selecting sensor placements to ensure the diagnosability of a system. However, the problem of computing an optimal sensor set with minimal size has a complexity exponential in the number of possible sensor placements [Yoo & Lafortune, 2001]. Existing sensor placement algorithms are based on the global representation of the system model, which may not be computable for large systems.

In this section we describe how our distributed approach can be extended to identify those system behaviours that require modification to restore diagnosability. Since a system may admit several possibilities to remove nondiagnosable behaviour we use a ranking approach based on cost estimation to isolate behaviours in easily accessible components whose modification removes not only the diagnosability problem but can also be cheaply performed. Our approach thus aims at assisting a human system designer to restore diagnosability. Note that we do not consider *how* to modify the system.

3.6.1 Computation of Critical Paths

The prerequisite of any such analysis is that we can compute indeed the whole set of nondiagnosability causes, that is, *all* critical paths. Currently our approach terminates as soon as a critical path is found (line 10 of Algorithm 5 on page 105). Now the question is when can we be certain that we have found all critical paths, not only among those in twin plants $\hat{\mathbb{G}}$ already considered, but indeed among all the twin plants of the entire jointree.

Surely, if we consider the whole jointree and perform the inward propagation starting in all the leaves followed by the complete outward propagation we obtain all critical paths. The problem with such an approach, however, is that it is not scalable. Thus in case the memory resources are not sufficient to perform the entire propagation we would not even be able to estimate to what extend we have covered the nondiagnosability causes.

In a first step it is therefore crucial to detect which paths could *possibly* be critical in order to focus our computational resources on the identification of which of these paths are indeed critical. Recall that our aim is to detect all critical paths which is a subset of all possibly critical ones. The latter need to be determined in all twin plants of the entire jointree. An efficient approach to detect such paths is the one based on *possibly nondiagnosable states.* All other states are certainly diagnosable.

Definition 30 (Possibly nondiagnosable states) *The set of* possibly nondiagnosable states $\mathbb{P}(\hat{G}_i)$ *of a local twin plant* $\hat{G}_i$ *is determined as follows.*

1. $\mathbb{P}(\hat{G}_i) = \{x \in \hat{X}_i \mid x \text{ is nondiagnosable}\}$ *if* $G_i \in Con(0,\ G_F)$ *(i.e.* $G_i = G_F$*)*
2. $\mathbb{P}(\hat{G}_i) = \hat{Y}_i$ *if* $G_i \in Con(\alpha,\ G_F)$ *with* $\alpha \in \mathbb{N}^+$ *and for all states* $\hat{y} \in \hat{Y}_i$ *and all connected twin plants* $G_j \in Con(\alpha - 1,\ G_F)$ *there exists a state* $(\hat{y}_i, \hat{x}_j)$ *in the twin plant* $Sync(\hat{G}_i, \hat{G}_j)$ *such that* $\hat{x}_j$ *is possibly nondiagnosable.*
3. $\mathbb{P}(\hat{G}_i) = \hat{X}_i$ *if* $G_i \notin transCon(\hat{G}_F)$ *(see page 118)*

Possibly nondiagnosable states $\text{PNS}(\hat{G}_i)$ for a twin plant $\hat{G}_i \in transCon(\hat{G}_F)$ are computed on the basis of the connected twin plants whose distance to $\hat{G}_F$ is smaller by 1. For the example of computing these states for $\hat{G}_5$ shown in Figure 3.16 it means that they are obtained by considering twin plants $\hat{G}_2$ and $\hat{G}_3$ in sequence. For each $\hat{G}_i \in \{\hat{G}_2, \hat{G}_3\}$ of them we compute $\hat{G}_{i,5} = Sync(\hat{G}_5, \hat{G}_i)$ to obtain all possibly nondiagnosable states $\hat{x} \in \text{PNS}(\hat{G}_5)$ for which there exists a state $(\hat{x}, \hat{y})$ in $\hat{G}_{i,5}$ where $\hat{y} \in \text{PNS}(\hat{G}_i)$. A state $\hat{x}$ in $\hat{G}_5$ is only possibly nondiagnosable, iff there exists a corresponding possibly nondiagnosable state in $\hat{G}_2$ and in $\hat{G}_3$ (i.e. there are two states $\hat{y} \in \text{PNS}(\hat{G}_2)$ and $\hat{z} \in \text{PNS}(\hat{G}_3)$ such that $(\hat{x}, \hat{y})$ is a state in $\hat{G}_{2,5}$ and $(\hat{x}, \hat{z})$ is one in $\hat{G}_{3,5}$).

The "propagation" of possibly nondiagnosable states among connected twin plants has some similarity with the propagation of nondiagnosable states during the outward pass of the jointree propagation (see Section 3.3.3). However, since now the propagation is neither performed on globally consistent twin plants, nor on all connected twin plants at once (i.e. there is usually more than one twin plant from which information about possibly nondiagnosable states is received), we do not have the guarantee that these states are indeed nondiagnosable.

Due to the differences of propagating possibly nondiagnosable and nondiagnosable states a state $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_{|\omega|})$ in a ω-coupled twin plant $\hat{G}_\omega$ is only *possibly nondiagnosable* iff $\forall i \in \{1, \ldots, |\omega|\}$ we have $\hat{x}_i \in \mathbb{P}(\hat{G}_i)$. This is in contrast to

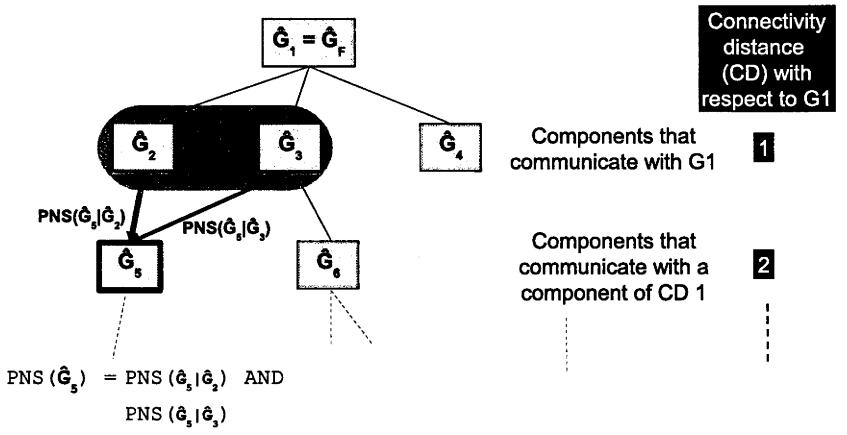


Figure 3.16: Scheme for computing nondiagnosable states. $\mathrm{PNS}(\hat{G}_i|\hat{G}_j)$ denotes the states of $\hat{G}_i$ that are possibly nondiagnosable with respect to $\hat{G}_j$. $\mathrm{PNS}(\hat{G}_5)$ are the possibly nondiagnosable states in $\hat{G}_5$.

the definition of nondiagnosable states that only need to be composed of a single nondiagnosable state (see page 90).

Any observable cycle with a possibly nondiagnosable state we will henceforth refer to as *possibly nondiagnosable cycle* and any path with such a cycle we call a *possibly critical path.* Then in order to show that all critical paths belong indeed to the set of possibly critical ones we prove that the following Theorem holds:

Theorem 3.6.1 *A state $\hat{x}$ in the GTP is nondiagnosable iff it is possibly nondiagnosable.*

Proof:

($\Rightarrow$) Suppose there exists a state $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_n)$ in the GTP such that $\hat{x}$ is nondiagnosable and a state $\hat{x}_i$ such that $\hat{x}_i \notin \mathbb{P}(\hat{G}_i)$. Condition 3 of Definition 30 verifies that all states $\hat{x}_h$ from $(\hat{x}_1, \ldots, \hat{x}_n)$ such that $\hat{G}_h \notin transCon(\hat{G}_F)$ are possibly nondiagnosable. It follows that $\hat{G}_i$ is in $transCon(\hat{G}_F)$. State $\hat{x}$ is nondiagnosable therefore the state $\hat{x}_F$ from $\hat{G}_F$ also contained in the n-tuple $(\hat{x}_1, \ldots, \hat{x}_n)$ is nondiagnosable. It follows that $\hat{x}_F \in \mathbb{P}(\hat{G}_F)$ (see condition 1 of def. 30), so $\hat{G}_i$ is in $transCon(\hat{G}_F) \setminus \{\hat{G}_F\}$. Therefore, there exists an $\alpha > 1$ such that $\hat{G}_i \in Con(\alpha,\ \hat{G}_F)$.

Condition 2 of Definition 30 ensures that there is a twin plant $\hat{G}_j \in Con(\alpha - 1,\ \hat{G}_F)$ in which all states $\hat{x}'_j$, for which $(\hat{x}'_j, \hat{x}_i)$ is a state in $Sync(\{\hat{G}_j, \hat{G}_i\})$, do not belong to $\mathbb{P}(\hat{G}_j)$. Since GTP results from the synchronisation of $\hat{G}_j$ and $\hat{G}_i$, it means that $\hat{x}_i \notin \mathbb{P}(\hat{G}_i)$ implies that $\hat{x}_j \notin \mathbb{P}(\hat{G}_j)$ where $\hat{x}_j$ denotes the state from $\hat{G}_j$ contained in $\hat{x}$.

The previous reasoning led from the existence of a state $\hat{x}_i = \hat{y}_\alpha$ in a twin plant $\hat{G}_i = \hat{H}_\alpha \in Con(\alpha,\ \hat{G}_F)$ with $\hat{y}_\alpha \notin \mathbb{P}(\hat{H}_\alpha)$ to the existence of a state $\hat{x}_j = \hat{y}_{\alpha-1}$ in a twin plant $\hat{G}_j = \hat{H}_{\alpha-1} \in Con(\alpha-1,\ \hat{G}_F)$ with $\hat{y}_{\alpha-1} \notin \mathbb{P}(\hat{G}_{\alpha-1})$. By recursively applying $\alpha - 1$ times the same reasoning, it follows that there exists a twin plant $\hat{H}_0$ belonging to $Con(0,\ \hat{G}_F)$ and a state $\hat{y}_0$ from $\hat{H}_0$ belonging to the n-tuple $(\hat{x}_1, \ldots, \hat{x}_n)$ such that $\hat{y}_0 \notin \mathbb{P}(\hat{H}_0)$. Since $\hat{G}_F$ is the only element in $Con(0,\ \hat{G}_F)$ (see def. 29) it means that $\hat{H}_0$ is actually $\hat{G}_F$ and $\hat{y}_0 = \hat{x}_F$. It finally follows that $\hat{x}_F \notin \mathbb{P}(\hat{G}_F)$, which is a contradiction.

($\Leftarrow$) Suppose there exists a possibly nondiagnosable state $\hat{x}$ containing the local state $\hat{x}_F \in \hat{G}_F$ such that $\hat{x}$ is diagnosable. This implies that $\hat{x}_F \notin \mathbb{P}(\hat{G}_F)$. Therefore $\hat{x} \notin \mathbb{P}(GTP)$ (see def. 30) which contradicts the assumption. □

Given the set of all possibly critical paths we now describe how we can use the available computational resources to identify those that are indeed critical. Algorithm 8 presents this procedure. In contrast to our original Algorithm on page 105 we now continue the computation (line 3) as long as there is sufficient memory and

- there exists either a possibly critical path in a twin plant that has not yet been considered, or

- there exists a critical path p in $\hat{\mathbb{G}}$ that is not yet guaranteed to be globally consistent (i.e. p belongs to the set $CritPath(\hat{\mathbb{G}})$ but not to the set P_{crit}^{cons}).

During each iteration we perform almost the same operations as before except that we now only check for critical paths defined over events in Σ_{int} if the first termination criteria is not satisfied. Otherwise the detection of critical paths is useless, since it can neither lead to a termination of the algorithm nor to a more efficient performance of future operations.

Upon termination the algorithm returns all critical paths $P_{\hat{\mathbb{G}}}$ in $\hat{\mathbb{G}}$ and all possibly critical paths $P_{\hat{\mathbb{G}}_{out}}$ in $\hat{\mathbb{G}}_{out}$. In case the procedure did not stop due to a lack of computational resources the latter set will be empty and the former contains only those paths that are consistent with the whole system and indeed critical. Recall that Theorem 3.6.1 guarantees that $P_{crit}^{cons} \subseteq P_{\hat{\mathbb{G}}} \cup P_{\hat{\mathbb{G}}_{out}}$. Hence the algorithm returns all critical paths.

Algorithm 8 ComputeAllCritPath(jointree: J)

1: $\hat{\mathbb{G}} \leftarrow \emptyset$ — nodes in J being considered
   $\hat{\mathbb{G}}_{out} \leftarrow J \setminus \hat{\mathbb{G}}$ — remaining nodes
2: $\Sigma_{int} \leftarrow \emptyset$ — events internal to $\hat{\mathbb{G}}$
   $P_{crit}^{cons} \leftarrow \emptyset$ — globally consistent critical paths
3: **while** ($ExistPosCritPath(\hat{\mathbb{G}}_{out})$ or $P_{crit}^{cons} \subset CritPath(\hat{\mathbb{G}})$) and $SufficientMemory(\hat{\mathbb{G}})$ **do**
4: $v \leftarrow PickNode(J, \hat{\mathbb{G}})$
5: $UpdateSets(v, \hat{\mathbb{G}}, \hat{\mathbb{G}}_{out}, \Sigma_{int})$
6: $Propagate(\hat{\mathbb{G}})$
7: **if** $ExistPosCritPath(\hat{\mathbb{G}}_{out})$ is false **then**
8: $\hat{\mathbb{G}}_{\Sigma_{int}} \leftarrow GetAllPathsOver\Sigma_{int}(\hat{\mathbb{G}})$
9: $Propagate(\hat{\mathbb{G}}_{\Sigma_{int}})$
10: $P_{crit}^{cons} \leftarrow GetCritPath(\hat{\mathbb{G}}_{\Sigma int})$
11: **end if**
12: **end while**
13: **return** $CritPath(\hat{\mathbb{G}}) \cup PossCritPath(\hat{\mathbb{G}}_{out})$

3.6.2 Dependencies among Critical Paths

Clearly, if we would consider all critical paths in all twin plants independently and remove them by making the required changes to the system behaviour we would obtain a diagnosable system. However, in general, the latter can be achieved with fewer modifications due to the dependencies among critical paths. These result from the fact that a single change to the component behaviour can remove more than one critical path.

We now show how we can detect dependencies among critical paths by labelling each transition such that the contribution of individual system behaviours to the synchronised twin plants can be obtained. Every twin plant transition is labelled with the set of component transition identifiers $\mathcal{T}$ that is composed of all those transitions whose removal would lead to the removal of the twin plant transition. This approach requires the following steps:

1. assign to every component transition (except the fault transitions) a unique identifier label,
2. propagate the identifier label to the corresponding interactive diagnoser $\widetilde{G}_i$,
3. propagate the identifier label to the corresponding twin plant $\hat{G}_i$,
4. propagate the identifier label to the messages sent by $\hat{G}_i$, and

5. propagate the identifier label to any twin plant $\hat{G}_j \neq \hat{G}_i$ receiving above message.

1. Fault transitions do not have a transition identifier, since they describe the component behaviour in case of the occurrence of a fault. An example for the assignment of transition identifiers is given in the top graph G_1^{lab} of Figure 3.17.

2. Every transition t of the interactive diagnoser is an abstraction of one or more component transitions $T_i' \subseteq T_i$. It is labelled with the union of all identifiers in T_i'. This results from the fact that changes to any of these component transitions would remove t. Formally there is a transition $(x, \mathcal{F}) \xrightarrow{\mathcal{T}-\sigma} (x', \mathcal{F}')$ in $\widetilde{G}_i^{lab}$ iff there is a transition sequence $x \xrightarrow{\mathcal{T}_1-\sigma_1} x_1 \cdots \xrightarrow{\mathcal{T}_m-\sigma_m} x_m \xrightarrow{\mathcal{T}_{m+1}-\sigma} x'$ in G_i with $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cdots \cup \mathcal{T}_{m+1}, \{\sigma_1, \ldots, \sigma_m\} \subseteq \Sigma_{f_i} \cup \Sigma_{u_i}$, and $\mathcal{F}' = \mathcal{F} \cup (\{\sigma_1, \ldots, \sigma_m\} \cap \Sigma_f)$. Diagnoser $\widetilde{G}_1^{lab}$ in Figure 3.17 shows an example with this labelling.

3. In the twin plant every shared transition corresponds to exactly one transition in the interactive diagnoser, and every observable transition refers to two transitions (one from the left and one from the right diagnoser). For shared transitions, the labelling is kept. For observable transitions the identifier labels are obtained from the union of the two corresponding diagnoser transition labels. Figure 3.17 shows the labelled twin plant $\hat{G}_1^{lab}$. Formally, there is a transition $(x^l, x^r) \xrightarrow{\mathcal{T}-\sigma} (x^{l'}, x^{r'})$ in $\hat{G}_i^{lab}$ iff there is

- either a transition $x^l \xrightarrow{\mathcal{T}^l-\sigma} x^{l'}$ in $\widetilde{G}_i^{l\ lab}$ and a transition $x^r \xrightarrow{\mathcal{T}^r-\sigma} x^{r'}$ in $\widetilde{G}_i^{r\ lab}$ with $\mathcal{T} = \mathcal{T}^l \cup \mathcal{T}^r$ and $\sigma \in \Sigma_o$,
- or a transition $x \xrightarrow{\mathcal{T}-\sigma} x'$ in $\widetilde{G}_i^{lab}$ with $\sigma \in \Sigma_s$.

4. Every transition t in a message corresponds to a set of transition sequences $\mathbb{T}^{seq} = \{T_1^{seq}, \ldots, T_k^{seq}\}$ in a twin plant $\hat{G}$ (see transition definition for projections on page 94 and Definition 25 on page 99). Now, since we require that the removal of any component transition is sufficient to remove the twin plant transition t, the identifier of t can only include those component transitions whose removal would remove every transition sequence in $\mathbb{T}^{seq}$. Let L_i denote the set of transition identifiers labelling the transition sequence T_i^{seq}. Then t is labelled by the intersection of all these labels ($label(t) = L_1 \cap L_2 \ldots \cap L_k$). Figure 3.17 depicts an example of a labelled message. For instance, here the transition $x3 \xrightarrow{l:s1} x11$ corresponds to the single transition sequence $T_1^{seq} = x3 \xrightarrow{\{t2\}-l:s2} x5 \xrightarrow{\{t3,t5\}-o1} x7 \xrightarrow{\{t6\}-l:s1} x11$. Therefore it is labelled with $L_1 = \{t2, t3, t5, t6\}$, the union of all transition labels of T_1^{seq}.

5. Since a message M is received by a twin plant $\hat{G}'$ via synchronisation, the label propagation is similar to case (3). Thus every transition in $Sync(M, \hat{G}')$ corresponds to the union of the labels referring to the set of transitions it represents. The bottom graph of Figure 3.17 shows a part of such a synchronised twin plant where for simplicity reasons the labelling of $\hat{G}_3$ is not taken into account.

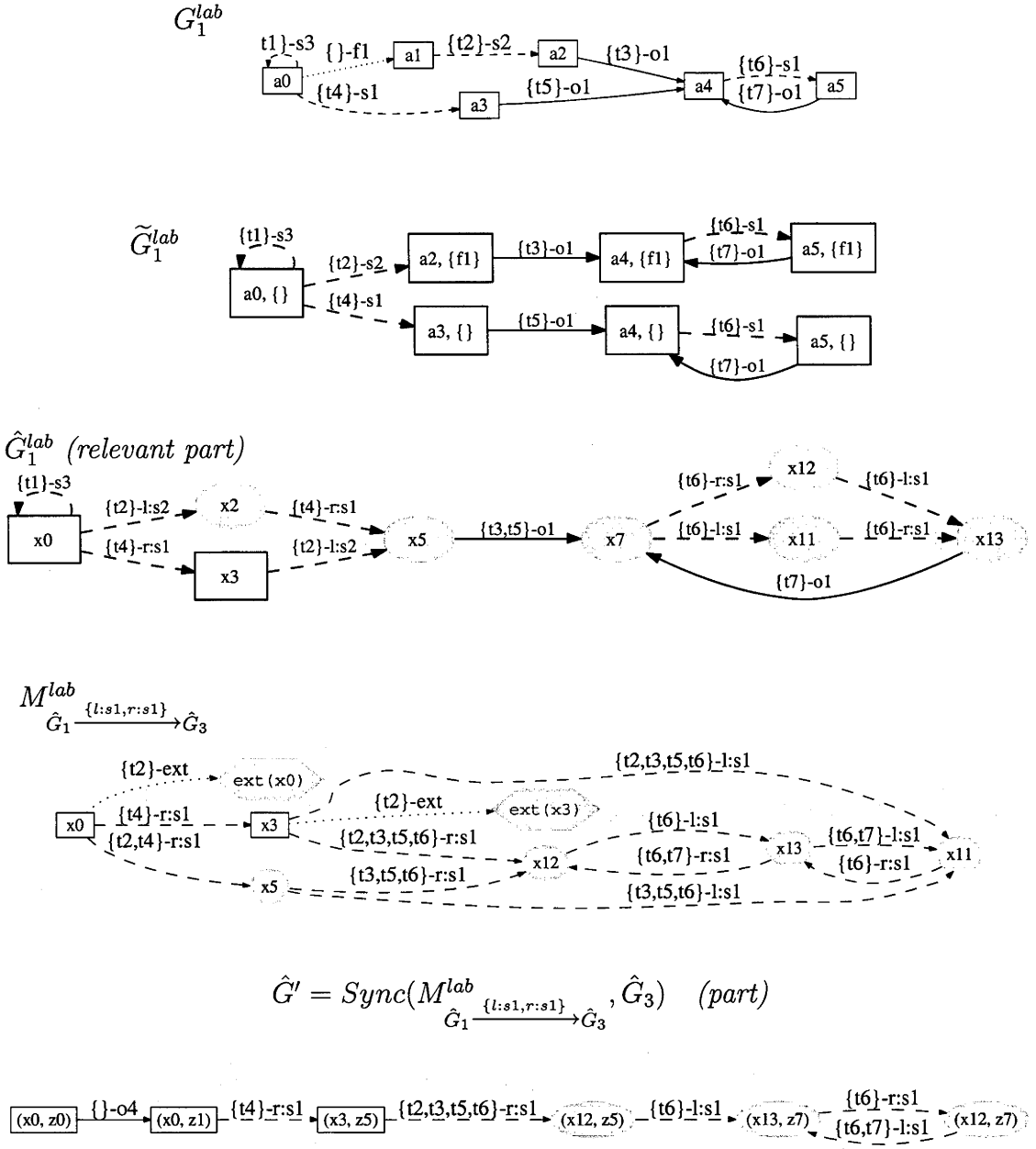


Figure 3.17: Illustration of the propagation of the component's transition identifiers (top) to a neighbouring twin plant (bottom).

Based on these transition labellings we can now decide whether two critical paths are *dependent*.

Theorem 3.6.2 *Two paths p and p' are* dependent, *iff there is a component transition labelling at least one transition in both p and p'.*

For instance, the critical paths in $\hat{G}_1^{lab}$ and $\hat{G}'$ shown in Figure 3.17 are dependent. They all include either transition $x13 \xrightarrow{\{t7\}-o1} x7$ or transition $(x12, z7) \xrightarrow{\{t6,t7\}-l:s1} (x13, z7)$. Both of these are only possible, if the behaviour represented by component transition $t7$ is not modified.

3.6.3 Optimal Removal of Nondiagnosability Causes

The dependencies among critical paths can now be exploited to obtain a diagnosable system by requiring fewer modifications to the behaviour of the components. Indeed this can be achieved by removing any *transition path cover.*

Definition 31 (Transition path cover $\mathbb{T}(\mathbb{P}_{crit})$) *A transition identifier set $\mathbb{T}$ is a path cover, denoted $\mathbb{T}(\mathbb{P}_{crit})$, for a set of critical paths $\mathbb{P}_{crit}$ iff every path in $\mathbb{P}_{crit}$ is labelled by at least one transition identifier in $\mathbb{T}$.*

An interesting problem here is to find a transition cover for a path set $\mathbb{P}$ with minimal size. This problem is similar to the NP-hard Hitting Set optimisation problem, which is defined as follows: Given a set of subsets $A = A_1, \ldots, A_m$ of the universal set $U = 1, \ldots, z$, the goal is to determine the smallest set $A' \subseteq U$ such that $\forall A_i : A' \cap A_i \neq \emptyset$. This problem is dual to the Set Cover problem which can only be solved by an $\mathbb{O}(log\ z)$-approximation algorithm [Arora & Lund, 1997].

In our case, the universal set is composed of all component transition identifiers and each subset A_i corresponds to the set of all transition identifiers labelling a single critical path. Thus, even if the modification costs for all transitions are the same it is very difficult to find a path cover with minimal size.

However, in practice the modification costs mc can vary greatly depending on the transitions to change. The aim is therefore not to find a transition-minimal path cover, but one that requires the least costs. To determine such a path cover we require a priori information on the costs associated with the modification of each component behaviour, in particular with each component transition. In the following, cost estimates are represented as numeric value $mc \geq 0 \in \mathbb{R} \cup \{\infty\}$. For transitions that can not be changed, for example, those in inaccessible components or behaviours determined by factors outside the scope of the system, the mc is set to ∞.

Definition 32 (Optimal transition cover $\mathbb{T}_{opt}$) *Let $\mathcal{T}$ denote the set of all transition covers for $\mathbb{P}_{crit}$. A transition cover $\mathbb{T}_{opt} \in \mathcal{T}$ is called* optimal *iff for all $\mathbb{T} \neq \mathbb{T}_{opt}$ in $\mathcal{T}$ the following holds: $mc(\mathbb{T}_{opt}) \leq mc(\mathbb{T})$ where $mc(T)$ denotes the costs required to modify all transitions in T.*

The costs for modifying a set of transitions $\mathbb{T}$ is obtained as the accumulated costs of modifying all individual transitions in $\mathbb{T}$. Therefore, the problem of computing the optimal transition set is analogous to the *Weighted Minimal Hitting Set* optimisation problem, which can be solved efficiently by the approximation algorithm presented in [Cincotti, Cutello, & Pappalardo, 2003]. Here, weights correspond to the modification costs. Using such an algorithm we can determine the best transition cover based on any path set returned by Algorithm 8.

3.6.4 Cost-Driven Computation of Nondiagnosability Causes

In addition to suggest possible behavioural modifications, the transition path covers may also be used to further increase the efficiency of Algorithm 8. Now it is no longer necessary to check whether all possibly critical paths are indeed consistent and hence critical. Rather we determine in each iteration i the critical paths P_{crit}^{cons} whose consistency can be verified based on the twin plants $\mathbb{G}$ considered so far (lines 8–10). For them we compute a transition cover $\mathbb{T}^i = GetTransCover(P_{crit}^{cons})$. Now a new node v is only added if in the outward message passed on from an element in $\mathbb{G}$ to v there is at least one path to a nondiagnosable state that is not labelled by a transition in $\mathbb{T}^i$. This results from the fact that otherwise every critical path in v or one of its children would necessarily be covered by $\mathbb{T}^i$ and hence the consistency check for these paths is not required. Therefore we can also terminate the search for critical paths if we cannot add a new node to $\mathbb{G}$ that would still require the verification of critical paths. In general, the cost-driven computation of nondiagnosability causes is therefore faster because the consideration of all jointree nodes with a possibly critical path can be avoided.

3.6.5 Extension to Multiple Faults

As stated in Section 3.4.4 a system is only diagnosable if all its faults are diagnosable. Therefore we need to find a transition cover for all critical paths of not just one fault, but all the faults. Again there are two possibilities to determine them. We can either consider all faults individually or compute the larger twin plants that

allow the consideration of these faults at once. The advantages and disadvantages of these methods are the same as stated in Section 3.4.4.

3.6.6 Summary

We have shown how we can extend the diagnosability approach to compute a subset of the nondiagnosability causes whose modification would be sufficient to obtain a diagnosable system. This method can also be used when the costs of the modifications need to be considered and those nondiagnosability causes need to be returned whose modification requires the lowest costs. The extension became only possible due to our novel scalable diagnosability approach presented in this chapter.

Chapter 4

Conclusion

We now summarise the main contributions of this thesis and outline some directions for future work. For a comparison to related work we refer the reader to sections 2.6 and 3.5.

4.1 Thesis Contributions

Diagnosing discrete-event systems poses the problem of determining the set of all possible faults that are consistent with a sequence of observations. When applied to large scale working systems this task has to be done on-line in a timely manner. In this case the diagnosis result is updated continuously when new events are observed. Its efficiency depends on the number of diagnosis candidates and/or on the extent to which the system description has been compiled off-line.

In this thesis we have described several contributions to increase the efficiency of the on-line diagnosis while taking into account the different time and space requirements of applications. The latter led us to present a spectrum of diagnosis approaches which differ in the amount of model compilation performed off-line. The underlying models range from the small component models that do not incorporate any compilation, to the diagnoser model in which the diagnosis information is compiled for the entire observable behaviour of the system.

In order to determine which diagnosis approach is best suited for a given application with specific time and space requirements we analysed the time/space tradeoff for all our diagnosis methods. For large applications where space is critical the diagnosis is best based on the decentralised models. On the other hand, smaller applications can be more efficiently diagnosed using our nondeterministic model.

To handle the generally large number of diagnosis candidates all our approaches are implemented symbolically using BDDs. This allows us to perform the diagnostic reasoning at once for the whole set of diagnosis candidates rather than considering each consistent system state and fault individually as required in an enumerative approach. To determine the advantage of the symbolic implementation over an enumerative one we have implemented four of our diagnosis methods across our spectrum also in an enumerative way.

Here only the on-line use of the symbolic diagnoser incurs a small time overhead. In all other cases the run-time of the symbolic approach is significantly better, and so are the space requirements of the larger models. Therefore, an enumerative approach is mainly useful for very small applications for which the computation and storage of the large diagnoser is feasible.

We have not only shown how we can exploit the advantages of BDDs for efficiently implementing diagnosis algorithms but also how we can use their properties for determining which computations are better performed off-line (those that are slow and hardly increase the model size) and which ones should be performed on-line (those that are fast and increase the model size). These considerations led us to define four models of our spectrum where the local fault information is compiled off-line. This decision is derived from the fact that the symbolic update of faults is very slow but its synchronisation, to obtain the fault information for the whole system, is very fast. Our experimental results have shown that these diagnosis approaches led to much faster diagnosis while requiring less space (except for the decentralised model that was larger than the component model).

Although BDDs can efficiently handle large sets of diagnosis candidates their size still has a significant impact on the diagnosis time. This motivated us to study how we can assist a system designer in designing systems where the diagnosis information can be determined (more) precisely, that is, systems that are (more) diagnosable. We have therefore presented a new algorithm to solve the diagnosability problem and have shown how we can extend it to assist a system designer.

Our algorithm addresses the fundamental bottleneck of the classical diagnosability approach, which requires the computation of the global twin plant method. Instead we consider local twin plants for subsystems which we make globally consistent by message passing on a jointree. We have presented a complete framework for computing and propagating such messages based on finite state machines, and presented new conditions for reducing message size without losing diagnosability

information. Our approach is scalable in the sense that if the problem is too complex to be solved with available computational resources, it can still provide an approximate analysis of the diagnosability of the system. We have also shown how we can extend our framework to identify all nondiagnosability causes regardless of the available computational resources. However, in general, with more computational resources, one can detect more inconsistencies in critical paths, avoiding the return of paths that are diagnosable. Finally we have shown how to identify component behaviours and transitions that, if modified, render a system diagnosable, while requiring minimal costs. This information can then be used by a systems designer to perform the according changes and restore diagnosability.

4.2 Directions for Future Work

The perspectives of this thesis are numerous. We now plan to extend our decentralised diagnosis approach by representing subsystems not only as decentralised diagnosis models but also as centralised and nondeterministic diagnosis models. By defining the latter models also at the subsystem level, we aim at increasing the on-line efficiency of diagnosis algorithms for large discrete-event systems.

It would then be very interesting to conduct an exhaustive experimental analysis to determine the best diagnosis approach for a given application. Such a decision might not only be based on the available computational resources but also on the number of observations, faults, components and their interactions, since these parameters might impact the efficiency of the diagnosis approach and thus the computational resources required to perform it.

Importantly, our work is largely orthogonal to the state of the art, and likely to benefit other approaches as well. It would be interesting, for example, to extend our framework to stochastic systems and compute probability distributions on diagnoses, using for instance algebraic decision diagrams which are generalisation of BDDs to real-valued functions over the booleans.

Finally, integrating diagnosis and planning for repair or reconfiguration actions is one of the most significant challenges faced by the field of model-based diagnosis [Console & Dressler, 1999]. Given the recent success of planning techniques based on symbolic model-checking, we believe that our framework will prove a good basis for addressing this challenge.

Concerning the diagnosability part, we aim at extending our framework to provide optimal design recommendations not only based on the modification costs

but also based on its gain in diagnosis time. To achieve this purpose we propose to implement this approach using BDDs and to incorporate it directly into our symbolic diagnosis framework. Then we could automatically determine the impact of possible modifications on the diagnosis time via simulation. A systems supervisor can then decide for each generated modification suggestion whether he wants to follow it given the expected gain for the on-line diagnosis. Alternatively we can also imagine automating this process based on a formal description of the supervisor specifying the exact conditions under which a modification is to be made. With such a framework we aim at providing design recommendations for improving the diagnosability of health monitoring system for aircraft maintenance [Ghelam *et al.*, 2006] or to provide an assistance to the design of composite Web Services [Yan *et al.*, 2005].

Bibliography

[Aghasaryan *et al.*, 1997] Aghasaryan, A.; Boubour, R.; Fabre, E.; Jard, C.; and Benveniste, A. 1997. A petri net approach to fault detection and diagnosis in distributed systems. Technical Report 1117, Irisa.

[Akers, 1978] Akers, S. B. 1978.x Binary decision diagrams. *IEEE Transactions on Computers* C-27(6).

[Albore & Bertoli, 2006] Albore, A.; and Bertoli, P. 2006. Safe LTL assumption-based planning. In *International Conference on Planning and Scheduling (ICAPS'06)*, 193–202.

[Ardissono *et al.*, 2005] Ardissono, L.; Console, L.; Goy, A.; Petrone, G.; Picardi, C.; Segnan, M.; and Dupré, D. T. 2005. Enhancing web services with diagnostic capabilities. In *3rd IEEE European Conference on Web Services (ECOWS'05)*.

[Arnold, 1987] Arnold, A. 1987. Transition systems and concurrent processes. In *Mathematical problems in Computation theory*, volume 21. Banach Center.

[Arora & Lund, 1997] Arora, S., and Lund, C. 1997. *Approximation algorithms for NP-hard problems.* PWS Publishing Co.

[Baroni *et al.*, 1999] Baroni, P.; Lamperti, G.; Pogliano, P.; and Zanella, M. 1999. Diagnosis of large active systems. *Artificial Intelligence* 110(1):135–183.

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

[Bertoli *et al.*, 2001] Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in nondeterministic domains under partial observability via symbolic model checking. In *IJCAI*, 473–478.

[Bertoli *et al.*, 2002] Bertoli, P.; Cimatti, A.; Slaney, J.; and Thiébaux, S. 2002. Solving power supply restoration problems with planning via symbolic model-checking. In *In Proceedings of the 15th European Conference on Artificial Intelligence*, 576–580.

[Bryant, 1986] Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* C-35(8):677–691.

[Bryant, 1991] Bryant, R. E. 1991. On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Transactions on Computers* 40(2):205–213.

[Burch *et al.*, 1990] Burch, J. R.; Clarke, E. M.; McMillan, K. L.; Dill, D. L.; and Hwang, L. J. 1990. Symbolic Model Checking: 10^{20} States and Beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, 1–33. IEEE Computer Society Press.

[Cassandras & Lafortune, 1999] Cassandras, C., and Lafortune, S. 1999. *Introduction to discrete event systems.* Kluwer Academic Publishers.

[Champarnaud & Coulon, 2004] Champarnaud, J.-M., and Coulon, F. 2004. NFA reduction algorithms by means of regular inequalities. *Theoretical Computer Science* 327(3):241–253.

[Cimatti, Pecheur, & Cavada, 2003] Cimatti, A.; Pecheur, C.; and Cavada, R. 2003. Formal verification of diagnosability via symbolic model checking. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence IJCAI'03*, 363–369.

[Cincotti, Cutello, & Pappalardo, 2003] Cincotti, A.; Cutello, V.; and Pappalardo, F. 2003. An Ant-Algorithm for the Weighted Minimum Hitting Set Problem. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium (IEEE SIS'03)*, 1–6.

[Console & Dressler, 1999] Console, L., and Dressler, O. 1999. Model-based diagnosis in the real world: lessons learned and challenges remaining. In *Proc. IJCAI-99.*

[Console, Picardi, & Ribaudo, 2002] Console, L.; Picardi, C.; and Ribaudo, M. 2002. Process algebras for systems diagnosis. *Artificial Intelligence* 142(1):19–51.

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

[Cordier & Dousson, 2000] Cordier, M.-O., and Dousson, C. 2000. Alarm driven monitoring based on chronicles. In *Proceedings of Safeprocess'2000*, 286–291.

[Cordier & Grastien, 2007] Cordier, M.-O., and Grastien, A. 2007. Exploiting independence in a decentralised and incremental approach of diagnosise. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 292–297.

[Cordier & Largouët, 2001] Cordier, M.-O., and Largouët, C. 2001. Using model-checking techniques for diagnosing discrete-event systems. In *Proceedings of the Twelfth International Workshop on Principles of Diagnosis (DX-01)*, 39–46.

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

[Darwiche, 1998] Darwiche, A. 1998. Model-based diagnosis using structured system descriptions. *JAIR* 8:165–222.

[de Kleer & Williams, 1987] de Kleer, J., and Williams, B. C. 1987. Diagnosing multiple faults. *Artificial Intelligence* 32(1):97–130.

[Debouk, Lafortune, & Teneketzis, 2000] Debouk, R.; Lafortune, S.; and Teneketzis, D. 2000. Coordinated decentralized protocols for failure diagnosis of discrete event systems. *Journal of Discrete Event Dynamical Systems: Theory and Application* 10(1–2):33–86.

[Dechter, 2003] Dechter, R. 2003. *Constraint Processing.* Morgan Kaufmann.

[Fabre, Benveniste, & Jard, 2002] Fabre, E.; Benveniste, A.; and Jard, C. 2002. Distributed diagnosis for large discrete event dynamic systems. In *Proceedings of the IFAC world congress*, 237–256.

[Fortune, Hopcroft, & Schmidt, 1978] Fortune, S.; Hopcroft, J.; and Schmidt, E. 1978. The complexity of equivalence and containment for free single variable program schemes. In *ICALP'78 - Automata Languages and Programming*, volume 62, 227–240. Springer-Verlag.

[Friedman & Supowit, 1990] Friedman, S., and Supowit, K. 1990. Finding the optimal variable ordering for binary decision diagrams. *IEEE Transactions on Computers* C-39(5):710–713.

[García *et al.*, 2005] García, E.; Correcher, A.; Morant, F.; E.Quiles; and Blasco, R. 2005. Modular fault diagnosis based on discrete event systems. In *Proceedings Discrete Event Dynamic Systems*, volume 15, 237–256.

[Ghelam *et al.*, 2006] Ghelam, S.; Simeu-Abazi, Z.; Derain, J.-P.; Feuillebois, C.; Vallet, S.; and Glade, M. 2006. Integration of health monitoring in the avionics maintenance systems. In *6th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, 1519–1524.

[Grastien, Cordier, & Largouët, 2004] Grastien, A.; Cordier, M.-O.; and Largouët, C. 2004. Extending decentralized discrete-event approach to diagnose reconfigurable systems. In *International Workshop on Principles of Diagnosis (DX-04).*

[Grastien *et al.*, 2007] A. Grastien, Anbulagan, J. R., and Kelareva, E. 2007. Diagnosis of discrete-event systems using satisfiability algorithms. In *American National Conference on Artificial Intelligence (AAAI-07).*

[Grosclaude, 2004] Grosclaude, I. 2004. Model-based monitoring of software components. In *International Workshop on Principles of Diagnosis (DX-04).*

[Hillston, 1996] Hillston, J. 1996. A compositional approach to performance modelling. *Cambridge University Press.*

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

[Iwasaki, 1997] Iwasaki, Y. 1997. Real-world applications of qualitative reasoning. IEEE Expert, 12(3):16–21.

[Jensen & Veloso, 2000] Jensen, R., and Veloso, M. 2000. OBDD-based universal planning for multiple synchronized agents in non-deterministic domains. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-00)*, 167–176. AAAI Press.

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

[Kurien & Nayak, 2000] Kurien, J., and Nayak, P. 2000. Back to the future with consistency-based trajectory tracking. In *American National Conference on Artificial Intelligence (AAAI-00).*

[Lamperti & Zanella, 2003a] Lamperti, G., and Zanella, M. 2003a. Continuous diagnosis of discrete-event systems. In *14th International Workshop on Principles of Diagnosis (DX-03)*, 105–111.

[Lamperti & Zanella, 2003b] Lamperti, G., and Zanella, M. 2003b. *Diagnosis of active systems.* Kluwer Academic Publishers.

[Lamperti & Zanella, 2004] Lamperti, G., and Zanella, M. 2004. Diagnosis of discrete-event systems by separation of concerns, knowledge compilation, and reuse. In *16th European Conference on Artificial Intelligence (ECAI-04)*, 838–842.

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

[Lee, 1959] Lee, C. Y. 1959. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal* 38:985–999.

[Marchand & Rozé, 2002] Marchand, H., and Rozé, L. 2002. Diagnostic de pannes sur des systèmes à événements discrets: une approche à base de modèles symboliques. In *13ème Congrès AFRIF-AFIA de Reconnaissances des Formes et Intelligence Artificielle*, 191–200.

[McMillan, 1992] McMillan, K. L. 1992. *Symbolic model checking : An approach to the state explosion problem.* Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh.

[Meyer & Stockmeyer, 1972] Meyer, A. R., and Stockmeyer, L. J. 1972. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th Annual IEEE Symposium on Switching and Automata Theory*, 125–129.

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

[Pencolé, Cordier, & Rozé, 2002] Pencolé, Y.; Cordier, M. O.; and Rozé, L. 2002. A decentralized model-based diagnostic tool for complex systems. *International Journal on Artificial Intelligence Tools* 11(3):327–346.

[Pencolé, 2004] Pencolé, Y. 2004. Diagnosability analysis of distributed discrete event systems. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, 43–47.

[Pencolé, 2005] Pencolé, Y. 2005. Assistance for the design of a diagnosable component-based system. In *Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence.*

[Perrow, 1984] Perrow, C. 1984. *Normal Accidents: Living with High Risk Technologies.* Basic Books.

[Poucet *et al.*, 1987] Poucet, A.; Contini, S.; Petersen, K. E.; and Vestergaard, N. K. 1987. An expert system approach to systems safety and reliability analysis. In Singh, M. G.; Hindi, K. S.; Schmidt, G.; and Tzafestas, S., eds., *Fault Detection and Reliability: Knowledge Based & Other Approaches.* Pergamon Press.

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

[Reiter, 1987] Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32(1):57–95.

[Rintanen & Grastien, 2007] Rintanen, J., and Grastien, A. 2007. Diagnosability testing with satisfiability algorithms. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 532–537.

[Robertson & Seymour, 1986] Robertson, N., and Seymour, P. D. 1986. Graph minors II: Algorithmic aspects of treewidth. *Journal of Algorithms* 7:309–322.

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

[Sampath *et al.*, 1995] Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1995. Diagnosability of discrete event system. *IEEE Transactions on Automatic Control* 40(9):1555–1575.

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

[Scarl, 1994] Scarl, E. 1994. Sensor placement for diagnosability. *Journal Annals of Mathematics and Artificial Intelligence.*

[Scherer & White, 1987] Scherer, W. T., and White, C. C. 1987. A survey of expert systems for equipment maintenance and diagnostics. In Singh, M. G.; Hindi, K. S.; Schmidt, G.; and Tzafestas, S., eds., *Fault Detection and Reliability: Knowledge Based & Other Approaches.* Pergamon Press.

[Schumann & Pencolé, 2006] Schumann, A., and Pencolé, Y. 2006. Efficient on-line failure identification for discrete-event systems. In *Proceedings of Safeprocess'2006.* Beijing, P.R. China: Elsevier Press.

[Schumann & Pencolé, 2007] Schumann, A., and Pencolé, Y. 2007. Scalable diagnosability checking of event-driven systems. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 575–580.

[Shannon, 1938] Shannon, C. E. 1938. A symbolic analysis of relay and switching circuits. *Trans. A.I.E.E.* 57:713–723.

[Shenoy & Shafer, 1986] Shenoy, P. P., and Shafer, G. 1986. Propagating belief functions with local computations. *IEEE Expert* 1(3):43–52.

[Sieling & Wegener, 1993] Sieling, D., and Wegener, I. 1993. Reduction of obdds in linear time. *Information Processing Letters* 48:139–144.

[Somenzi, 2005] Somenzi, F. 2005. CUDD: CU decision diagram package release 2.4.1. In *University of Colorado at Boulder.*

[Struss, 1997] Struss, P. 1997. Fundamentals of model-based diagnosis of dynamic systems. In *Proc. IJCAI-97, Nagoya, Japan*, 480–485.

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

[Sztipanovits & Misra, 1996] Sztipanovits, J., and Misra, A. 1996. Diagnosis of discrete event systems using ordered binary decision diagrams. In *Seventh International Workshop on Principles of Diagnosis.*

[Thiébaux *et al.*, 1996] Thiébaux, S.; Cordier, M.-O.; Jehl, O.; and Krivine, J.-P. 1996. Supply restoration in power distribution systems – a case study in integrating model-based diagnosis and repair planning. In *Conference on Uncertainty in Artificial Intelligence – UAI'96)*, 525–532.

[Torta & Torasso, 2004] Torta, G., and Torasso, P. 2004. The role of obdds in controlling the complexity of model based diagnosis. In *International Workshop on Principles of Diagnosis (DX-04)*, 9–14.

[Travé-Massuyès, Escobet, & Milne, 2001] Travé-Massuyès, L.; Escobet, T.; and Milne, R. 2001. Model-based diagnosability and sensor placement application to a frame 6 gas turbine subsystem. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI'01*, volume 1, 551–556.

[Viswanadham & Johnson, 1988] Viswanadham, N., and Johnson, T. L. 1988. Fault detection and diagnosis of automated manufacturing systems. In *Proc. 27th IEEE Conferencce on Decision and Control*, 2301–2306.

[Wang, Yoo, & Lafortune, 2004] Wang, Y.; Yoo, T.; and Lafortune, S. 2004. New results on decentralized diagnosis of discrete event systems. *Proceedings of 42nd Annual Allerton Conference on Communication, Control, and Computing.*

[Williams & Nayak, 1996] Williams, B., and Nayak, P. 1996. Immobile robots – ai in the new millenium. *AI magazine* 17(3):17–35.

[Xue, Yan, & Zheng, 2005] Xue, F.; Yan, L.; and Zheng, D. 2005. Fault diagnosis of distributed discrete event systems using OBDD. *Informatica* 16(3):431–448.

[Yan *et al.*, 2005] Yan, Y.; Pencolé, Y.; Cordier, M.-O.; and Grastien, A. 2005. Monitoring web service networks in a model-based approach. In *Proceedings of the 3rd European Conference on Web Services (ECOWS05).*

[Yan, 2004] Yan, Y. 2004. Sensor placement and diagnosability analysis at design stage. In *MONET Workshop on Model-Based Systems at 16th European Conference on Artificial Intelligence.*

[Yang, 1999] Yang, B. 1999. Optimizing model checking based on BDD characterization. Technical Report CMU-CS-99-129, School of Computer Science, Carnegie Mellon University.

[Yoo & Lafortune, 2001] Yoo, T., and Lafortune, S. 2001. On the computational complexity of some problems arising inpartially-observed discrete-event systems. *American Control Conference*, volume 1:307–312.

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