SAT-Based Diagnosability and Predictability Analysis in Centralized and Distributed Discrete Event Systems

Hassan Ibrahim

NNT : 2016SACLS539

Thèse de doctorat de l'Université Paris-Saclay préparée à l'Université Paris-Sud

Ecole doctorale n°580
Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique

par

M. Hassan Ibrahim

Analyse à base de SAT de la diagnosticabilité et de la prédictabilité des systèmes à événements discrets centralisés et distribués

Thèse présentée et soutenue à Orsay, le 16 décembre 2016.

Composition du Jury :

M.	Stefan Haar	Directeur de recherche INRIA & ENS Cachan	(Président du jury)
M.	Stéphane Lafortune	Professeur University of Michigan	(Rapporteur)
M.	Lakhdar Saïs	Professeur Université d'Artois	(Rapporteur)
M.	Hervé Marchand	Chargé de recherche INRIA	(Examinateur)
M.	Laurent Simon	Professeur Université de Bordeaux	(co-directeur de thèse)
M.	Philippe Dague	Professeur Université Paris-Sud	(Directeur de thèse)

To Hadil and Jado
To my loving family in Syria
To my Syria ...

Acknowledgements

I want to acknowledge the financial support of this thesis by a scholarship from the Syrian government, in particular from the High Institute of Applied Sciences and Technology (HIAST), and also by the University Paris Sud. I would like to specially thank the responsibles at HIAST for their care of to continue the funding to the maximum possible degree despite the very hard conditions in my beloved Syria.

I would like to thank the reviewers of this thesis, Professors Stéphane Lafortune and Lakhdar Saïs for the valuable time, they have allocated in order to evaluate the quality of this work, despite of their busy schedules. Many thanks also to Professor Stefan Haar and Dr. Hervé Marchand for accepting to be members of my thesis committee.

During this thesis, I had the chance to work with some amazing researchers, from whom I have learned a lot and to whom I would like to acknowledge the results of this thesis. First of all is Professor Philippe Dague, I would like to thank him for his patience, encouragements, guidance and his permanent availability for discussion (even at nights or in the weekends), his passion for science and for work in general is an example that I will never forget. Actually this thesis would not given its fruits without his guidance and his constructive comments during our continuous discussions. I would like to thank also my co-supervisor, Professor Laurent Simon who kindly proposed to supervise this thesis at the beginning and despite of his early move to another university, his comments and feedbacks on my work were always constructive and allowed me to see the work from another perspective to see more and learn more.

I would like to thanks also Alban Grastien and Lina Ye for their interesting discussions and collaboration. Double thanks to Lina for her continuous encouragements and support to keep going on during my hard moments.

I want also to thank all LRI members and especially LaHDAK team members, where I have done this thesis, although my topic was not so compatible with their current research topics but I tried to retain as much as I can from them and their seminars! Thanks also for financing my summer schools and conferences.

Many thanks to my officemates Céline, Camille and Martin for breaking the boredom of the office hours by their friendly conversation, for the knowledge I acquired from them about

France and also for giving me the chance to practice and improve my French with native speakers with very different talking speeds!. I hope that we will keep in touch and that our paths cross again.

Living far from the family in Syria was not so easy, and here is where our friends come to compensate us and to add the flavor of a family to our life. I want to thank from all my heart Wassim and Yara for standing with me and my small family in this last year and especially in these last busy days of my thesis, I wish the best for you and your families!. Thanks also to Haidar, Alessandro for all your support.

I will never forgot to thank my family in Syria, who despite of the hard conditions in Syria did not stop of thinking of us and supporting us in all the ways. My parents, my father-in-law and mother-in-law, my sisters (Maryam and Zeina), my brother (Mohammad) and my cousins (Assil, Awos, Alaa, Rahaf, Ali and Usama), so thank you all again and again!

Finally, all my love and thanks to my wife (Hadil) and my son (Jad) for their endless love and support in all my hard days during these thesis, without you I would not ever arrived to this point. You are the hope of my life.

Abstract

Complex systems are omnipresent in our lives, but subject to failures that it is important to detect or predict. Discrete event system (DES) modeling is a natural way to represent and study such systems formally. Thus a system can be described by a set of states such that its current state is obtained after firing a sequence of events. These events are predefined in a finite set and can be fired spontaneously in the system. Not all these events are observable (measurable) and some of them are considered faulty, thus they model an abnormal change between two system states.

The diagnosis process in DES aims at determining with certainty if the system is currently in a faulty state or in a normal one, i.e., if an abnormal change of a system state has occurred or not. To this end, a system observer has only the sequence of observable events to decide the current status of the system state. However this state might be currently ambiguous (normal or faulty) according to the available observations. Moreover it can be permanently ambiguous! The possibility to disambiguate it using a finite number of observations is called the diagnosability of a faulty event occurrence. The fault is diagnosable if all its occurrences are diagnosable and the system is diagnosable if all its faults are diagnosable. Similarly, the possibility to predict a future occurrence of a fault using its preceding observable events is called the predictability of a faulty event occurrence. Both problems of diagnosability and predictability can be generalized to study the diagnosability or the predictability of a pattern of events, i.e., an extension-closed language represented by a finite state machine.

This thesis considers in its first part the problems of checking event diagnosability, event predictability and pattern diagnosability in centralized and distributed (with observable or unobservable synchronous communication events) discrete event systems, using SAT solvers. Thus we have encoded them as SAT problems, studied incremental SAT variants and provided experimental results that prove the scalability and flexibility of this approach. In the second part, we have introduced the diagnosability planning problem. This problem consists in finding a plan of actions (intentional/designful predefined events) that ensures, when applied on a set of potential current system states (called a current belief state), to drive the system in a diagnosable belief state from which it can be left to run freely (without control actions). This problem can arise after an external intervention on the system, like the application of a repair plan after a fault detection.Thus this approach can ensure the possibility to detect the system further faults. We analyzed this problem, proved its PSpace-completeness and proposed three methods to find the intended plan that we compared on a benchmark created for this purpose.

Table of contents

List of figures

List of tables

Chapter 1

General Introduction

"If you want new ideas, read old books!"— Ivan Petrovitch Pavlov

This thesis is about diagnosability and predictability analysis in Distributed Discrete Event Systems using a propositional logic approach, in particular using SAT solver. So, what is this? Even if you try to look for the word diagnosability in an Oxford dictionary, you will not get any answer and it may take you, in an any online dictionary, to the word diagnosis, which means "the identification of the nature of an illness or other problem by examination of the symptoms", which is usually used in a medical context. Translating the word in French will return, if it returns something, the word "diagnosticabilité" not a more meaningful one even for native French speaker!! If you move to predictability maybe you will get more results but most of them will be irrelevant to our predictability meaning here. If you search for diagnosability or predictability analysis in Distributed Discrete Event Systems you will get tens of the scientific articles that we will browse in the next section and chapters and will relate them to our work in this thesis. Finally adding the SAT solver to it in the google search will often give you the work in [Rintanen & Grastien 2007], which is our starting point in this thesis. However, before getting on our work, you may want to see our compilation of some historical points which gives you the intuition or the message of the work from our point of view. This compilation contains mostly historical events taken from Wikipedia pages! connected to show our intuition why the SAT-based approach may work, even if the usage and the formalism used to get the experimental results are not yet discussed. It is only an intuition and you may even re-read this introduction as a conclusion! Last point here, if you are familiar with all the keywords, you may want to proceed directly to the contributions section and come back to this compilation later.

1.1 The Origin of the Story: about Logic, Control, Information and Diagnosability

Since the beginnings of humanity on earth, human beings did not stop the interactions with the surrounding nature, firstly, each human being was empowered only by his five senses to do this interaction. He measured the different threats around him to be able to take the decision that keeps him safe to survive. Motivated by his innate curiosity/desire to interact, he developed his tools till creating languages, which allow him to communicate and constructed communities that guaranteed his domination against other species on the planet. After inventing writing, his experiences became more available, endurable (accumulative) and reinforced again his power. Although after the invention of languages humans were crowned on the throne of organisms, however some pioneers from this species continued, and are still continuing, giving the humanity its big cultural strides, through studying the natural phenomena affecting this mankind. They were always guided by their imagination and forte desire to understand and to *control* this surroundings for the benefits of mankind, especially that they realized how insignificant human beings are, in terms of energetic resources, in comparison to other species, not really recognizing the power of the amazing tool they created, i.e., the languages. For this purpose they have developed different measures continuously to understand this surroundings before controlling it with the less possible effort. Their new ideas started by reading the very old book given by the nature; through it trying then to emulate beings from their surroundings by developing models, lows and rules to describe and simulate these beings and even the natural phenomena.

Hereafter some examples that are related historically to our story are mentioned. Thus, in order to study dynamics of moving objects Isaac Newton invented the laws of physics. Then the laws of thermodynamics were set and the first realization of the so-called *system* was done by Nicolas Léonard Sadi Carnot in 1824; through organizing some interconnected components such that each component has a functionality that serves to achieve a global mission of the system. Actually it was a body of water vapor that works when heat is applied to it and it was named the working substance in steam engines, thus it can work for a neighbor by pushing it. Rudolf Clausius generalized the picture in 1850 of the working substance by considering the surroundings of the working substance thus it became the system with known limits so we can define its input and output.

In 1854, George Boole introduced Boolean algebra in his book The Laws of Thought which has been fundamental in the development of digital electronics, and is provided in all modern programming languages. It is also used in set theory and statistics. Boolean *logic* is credited with laying the foundations for the *information age*.

Actually at that time laws of thermodynamics were not taken directly for granted and proving them was not devoid of adventure and skepticism. Thus solving the famous Maxwell's daemon paradox, that tried to contradict the second law, took many years. This law implies the existence of a quantity called the entropy of a thermodynamic system. Which is used to measure the disorder of a system.

Thus the daemon appeared able to do the job of filtering/classifying gas particles into two classes at free cost. Falsifying this view was in 1929, when Leo Szilard proposed that doing the job by measuring the gas particles cannot be done at free cost which means that the daemon should have negative entropy from the act of acquiring information which would require an expenditure of energy. This negative entropy can be seen as information. In other words, Maxwell's daemon gets the entropy from the collected information and uses it to decrease the entropy of the "observed" gas. In fact Szilard did not just "save" the second law of thermodynamics but also explained to all humanity how the information can be considered as a form of energy that justifies this kind ability to hold the leadership over this planet!

Later, the work of Szilard was the starting point for Claude Shannon to state the basics of information theory and to Alan Turing to state the computation theory. Thus, Shannon assumed correct computation (encoding, decoding) to get reliable communication and storage over a noisy communication channel, while Turing assumed the correctness of storage and communication to get the computation which is realized using Von Neumann architecture of an electronic digital computer and opened the way for the emergence of computability, functionalism and artificial intelligence.

Algorithmic information theory was introduced in the 1960s by Ray Solomonoff and later was developed independently by Andrey Kolmogorov in 1965 and Gregory Chaitin around 1966 as the subfield of information theory and computer science. It is the information theory of individual objects, using computer science, and concerns itself with the relationship between computation, information, and randomness. The information content or complexity of an object can be measured by the length of its shortest description.

Nowadays humans are controlling systems that they have created everywhere and from all sizes. Thus, the origin of the story is that we want to check if we can control a system at a given price (represented by observations cost) and the cost of resources to do the checking process (represented by the efficiency of our method). In other words, to identify a system situation (or diagnosis). Actually a system is much more than a diagnosis, but diagnosing aims at answering a specific question about the system, it is like projecting all the acquired information about the system on one of two classes. Diagnosability, which as we said above can be seen as the ability to diagnose, is the problem of ensuring that there is at most two classes of diagnoses. It is classifying possibilities into two classes. The ambiguity is always

the third class that raises difficulty and observations are the way to organize the picture for a clearer vision about the system.

Discrete Event System (DES) model is a natural projection of the normal human thinking of the dynamics of a system. Thus we tend to differentiate several stations of a system behavior, let us call them states, and some spontaneously events that take the system's state to its successor state. In discrete event systems, the output of the diagnosis can be the set of behaviors that explain the observations, where faulty events will be discovered in the results. Diagnosability is the ability to have a precise diagnosis, i.e., given a set of observations paths, have exactly one label of diagnosis, either faulty or correct, consistent with them.

Nowadays systems are omnipresent and they are more and more complex and distributed, the need to control these systems is essential as they are always subject to faults. Thus one wants to know if a system is doing its intended mission (so it is in a normal state otherwise in a faulty one). In order to represent such states in the system, many formalisms can be adopted to model the system like discrete, continuous or hybrid. We adopt here the one which is the closest to the human way of thinking, as we mentioned above, i.e., the discrete one. However, the size of these systems and their distributed nature do not allow them to be fully observable, as such assumption would require very high costs of sensors and measurement process. For this, an abstraction over the system view is applied. However, this lack of information can prevent or complicate diagnosing a fault, i.e., identifying the current system status after having acquired some observation. The problem of the diagnosability of faults means here the ability to detect them after a finite number of observations that proceed their occurrence.

The first introduction to the notion of diagnosability in discrete event systems was by [Sampath *et al.* 1995]. The authors introduced an approach to test this property by constructing a deterministic diagnoser. However, in the general case, this approach is exponential in the number of states of the system, which makes it impractical.

In order to overcome this limitation the work in [Jiang *et al.* 2001] introduced the *Twin Plant* approach, which uses a special structure called Twin Plant. This approach turns the diagnosability problem into a search for a path with a cycle in a finite automaton, and this reduces its complexity to be polynomial of degree 4 in the number of states (and exponential in the number of faults, but processing each fault separately makes its linear in the number of faults).

Let us mention here that the two previous works were interested in centralized systems with simple faults modeled as distinguished events. The first studies about (surveillance or supervision) patterns were introduced in [Jéron *et al.* 2006] and [Genc & Lafortune 2006a] which generalize the simple fault event in a centralized DES to handle sequences of events

considered together as a fault, or multiple occurrences of the same fault or of different faults, or more generally any given behavior to be monitored.

The first work that addressed diagnosability analysis in Distributed DES (DDES) was [Pencolé 2004]. A DDES is modeled as a set of communicating Finite State Machines (FSM). Each FSM has its own events set, synchronous communication events being the only ones shared by at least two different FSM. Thus it also depends on the construction of local Twin Plants then synchronizing them incrementally and using some abstraction to avoid constructing the global Twin Plant of the system. The work by [Ye & Dague 2010] has optimized the construction of local Twin Plants. The generalization to patterns in DDES was introduced by [Ye *et al.* 2010].

After the reduction of diagnosability problem to a path finding problem by [Jiang *et al.* 2001], it became transferable to a satisfiability problem like it is the case for planning problems [Kautz & Selman 1992]. This was done by [Rintanen & Grastien 2007] which formulated the diagnosability problem (in its Twin Plant version) into a SAT problem, assuming a centralized DES with simple fault events. The authors represented the studied transition system by a succinct representation (cf. section 2.4.2). In fact, the main difficulty in diagnosability algorithms is how to reduce the amount of information that must be acquired to retrieve the diagnosability decision which in turn is related to the difficulty of states number explosion. SAT solvers, which are very powerful tools for checking efficiently the satisfiability of propositional/Boolean logical formulas, are now ubiquitous in artificial intelligence and they can deal with such problems. Such propositional formulas consist of n different Boolean variables participating partially or totally in each of m different clauses, each clause being a disjunction of its participating variables. They help to break the diagnosability question into smaller questions to check if the system description is not violating specific constraints. SAT solvers exploit the statistical approaches and the logical reasoning powers. Their heuristics allow them to compile the tested formula into a simpler description and to introduce randomness in its browsing which can reduce the complexity of deciding if it is satisfiable or not.

Actually the results presented in [Rintanen & Grastien 2007] show a good scalability in comparison with the Twin Plant approaches which the authors said to be impractical for systems with number of states larger than 10000 (actually, the literature contains almost no benchmark or experimental results of the Twin Plant approach and does not propose code availability, so real comparison is difficult). And as we mentioned above this approach considered only centralized DES. This motivated us to consider this approach our starting point, and try to add communication events to consider the distributed DES case and to study the effect of employing incremental SAT mode in both centralized and distributed cases through providing the experimental results of our study, which will be presented in Chapter 3.

From another side, we noticed that many similar problems like predictability analysis which consists in verifying the existence of an observable sequence that reveals with certainty that the fault will occur (see Chapter 4) and pattern diagnosability (see Chapter 5) can be encoded using similar techniques without passing by sophisticated structures used in the literature to deal with such cases.

Our contribution can be informally summarized by saying that we can answer efficiently each of these "ability to know" problems about large systems by "speaking logically" with a SAT Solver, which will intelligently summarize our language into a "small" number of well ordered dependent questions to deduce the answer. This is instead of building a sophisticated structure which requires a larger amount of information to be constructed and then checking a necessary and sufficient condition to decide the problem.

1.2 Contributions

The first part of this thesis is based on an existing approach to check diagnosability in centralized discrete event systems using SAT solvers; this approach is recalled in chapter 2 and our contributions in this part are presented in chapters 3,4,5. Thus, we reviewed the following problems in the literature and provided for each one a propositional logic formalism that is implemented and tested using the SAT solvers technology:

- Diagnosability of simple faulty events in distributed discrete event systems. We extend in chapter 3 the existing formalism in order to consider the synchronous communication events and we show how our encoding is smooth enough such that changes in this encoding are just by adding synchronization formulas without changing the internal encoding of each component. This encoding pushes the synchronization cost to the SAT solver and can consider observable and unobservable communications simultaneously. The flexibility will be exploited for dealing with other similar problems. We show how scalable this approach is through experimental results, which show clearly in particular the efficiency to prove non-diagnosability. However, proving diagnosability in SAT has not the same efficiency, the same concern (SAT vs. UNSAT result) appears in many problems that are reduced to SAT. We propose an approach that mitigates this problem using the incremental mode of SAT solvers, hence we modify the problem encoding to adopt this mode then we test this encoding on an artificial benchmark in centralized and distributed DES and we show its benefits on performance and scalability.

- Predictability of event occurrence in centralized and distributed DES. We exploit in chapter 4 the flexibility of adding and removing constraints in order to model the

predictability problem as a SAT one. This property is stronger than diagnosability and it consists in verifying the existence of a prefix that precedes the event f under consideration such that it reveals with certainty that f will occur each time after it has been seen. We test our encoding and show the scalability of this approach. Then we propose an encoding making use of incremental SAT to exploit the learned constraints about the system behavior, this can help in a more efficient verification which is also automated.

- Pattern Diagnosability. We propose in chapter 5 an encoding for the problem of pattern diagnosability, which generalizes the diagnosability of a specific event to consider any formal extension-closed rational language of events. Actually we have implemented this part but not yet tested it.

The second part of this thesis is presented in Chapter 6 where we introduce the problem of diagnosability planning in controllable DES. This problem, not yet considered in the literature to the best of our knowledge, consists in finding a plan of actions (taken from given elementary control actions) that, when applied on a given belief state (a set of potential current states), drives the system into a *diagnosable* belief state in order to be able to let the system run from its new initial belief state freely, i.e., without any control but however with guaranteed diagnosability. This problem may appear after a repair plan which leaves the system in a set of potential states from where the diagnosability of the system has not been yet ensured. It is also interesting, when possible, to replace the active diagnosability by diagnosability planning as a monitoring approach in order to reduce the diagnosis interactions with a running system that may add potential cost and affect the availability of the awaited services from the system, e.g., in monitoring power networks. In this part our contributions are the following:

- Formalizing the diagnosability planning problem.
- Analyzing and proving its complexity class.
- Proposing three methods to solve the problem efficiently.
- Implementing and testing these methods on an artificial benchmark that we created for this purpose.

1.3 Thesis Organization

This thesis consists of seven chapters; five main chapters in addition to this introduction chapter and the conclusion and future works chapter. The second chapter reviews the state

of the art mainly about the diagnosability problem in centralized and distributed DES, in particular it reviews the Labeled Transition System formalism used in the literature to study this problem showing the main approaches. Then we recall in some detail the starting point of this thesis which is the work of [Rintanen & Grastien 2007] which introduced the succinct transition systems that we used to encode the different cases studied here. The third chapter presents our first contribution which is the extension of the existing approach to consider communication events in the formalism then to adopt the encoding to be processed by an incremental SAT solver. In the fourth chapter we consider the predictability property and we encode it using the succinct transition systems into a SAT problem in both centralized and distributed structures. The fifth chapter is devoted to present our encoding of the pattern diagnosability problem as a SAT problem. The second part of the thesis is presented in the sixth chapter where we introduce the problem of diagnosability planning in controllable DES, this part is not strictly disjoint from the first part as we will use the Twin Plant structure recalled in chapter 2, however we do not use SAT technology in this part even if it is mentioned in the future work sections in the last chapter where we conclude.

Chapter 2

Preliminaries

In this chapter we provide the preliminaries required to follow this thesis, that will be cited in the next chapters whenever needed. It contains the formal definitions for the system model that we adapted in our study, like labeled transitions systems in centralized and distributed versions. We recall also the diagnosability problem definition and review how it is addressed in the literature. The basics to understand Satisfiability problem will follow, then we recall the succinct transitions systems in their centralized version and show how they have been used to encode the diagnosability problem in SAT.

2.1 Introduction to Fault Diagnosis

Diagnosis task is mainly using the available observations to explain the difference between the expected behavior of a system and its real behavior which may contain some faults. Systems are around us everywhere in our life, and as they are always subject to faults, therefore fault diagnosis is an important domain that will be always needed to ensure the safe availability of these systems. Examples of diagnosis application can be found in domains such as transportation from vehicles to spacecrafts, infrastructure like water or power networks and in software and hardware testing in addition to many complex systems. This wide appearance motivated from many years a lot of works to study the automatic approaches to do system fault diagnosis [Bavishi & Chong 1994, Sampath *et al.* 1996, Sampath *et al.* 1998, Ushio *et al.* 1998, Debouk *et al.* 2000, Grastien *et al.* 2007]. They all try to deal with the main problem which is the compromise between the number of possible diagnoses to the faulty system and the number of observations which must be given to make the decision. The diagnosis problem is NP-hard and one always needs to cope with an explosion in the number of system model states.

In general faults can be classified according to their lasting time in transient, intermittent or permanent faults. A transient fault occurs accidentally usually after an external interference then disappears before detecting it leaving some effects on the system. An intermittent fault leaves the system in an alternating state between normal and abnormal; like a malfunctioning device in a situation that does not respect its operating conditions of temperature and voltage, or because of the interaction among not fully compatible devices. A permanent fault leaves the system in a faulty state until its reparation, this can be an effect of a physical damage that needs to be repaired. We are interested in this thesis in permanent faults.
In model based diagnosis, faults could be modeled by states, so the diagnosis question would be whether the system is currently, using the current observation, in a faulty or a normal state. They can be also modeled as faulty events assigned to some transitions of the system, and so the diagnosis question will be whether a faulty event occurred or not, after regarding the set of available observation coming from the system's sensors. We are interested here in faults represented as faulty events.

The problem is that the diagnosis question must be answered precisely in order to take the appropriate action; for example in case of an affirmative answer, i.e. the fault has occurred, one can proceed to isolate the faulty components of the system and later repair them. However, in a *definitive* negative answer the system is ensured to be running correctly which is the normal purpose of a system. Otherwise, one can wait to acquire more observation if the current state is undetermined, i.e., can be reached with either a faulty behavior or a normal one. However, the diagnosis decision maybe always uncertain, and thus running a diagnosis algorithm may not be accurate. For example, two sets of observations provided by different sets of sensors or at different times may lead to different diagnoses, one faulty and the other normal. This uncertainty raises the problem of diagnosability which is an essential property to be ensured while designing the system model. It simply means the ability to get a precise diagnosis. After that, the model based diagnosis will be used in applications to explain any anomaly, with a guarantee of correctness and precision at least for each anticipated fault in the model.

2.2 Modeling Formalism

First of all, we define what is a system, following the IEEE definition *"A system is a combination of components that act together to perform a function not possible with any of the individual parts"*. To this end the system is represented by a model, which represents the evolution from one state to a successive one through transitions. A state of the system, or

simply a state, describes the system current status, usually using a vector of state variables which are identified to this purpose. Transitions can be synchronized to a time clock and so each state will represent the system status at a specific time-step, or can be assigned to some event, from an asynchronous set of events, in this case a state will hold the required information to represent the last event's effects which led to the current state. Transitions can also be modeled using a combination of the event and a time tag to represent the event's time occurrence or some temporal condition related to this event.

In this thesis we use timeless model, so transitions are assigned only to a set of events which are asynchronous in the centralized case and will have some synchronous events that connect the different components in the distributed case.

In the fault diagnosis domain, three types of systems are considered, the continuous systems, the discrete systems and the hybrid systems. We are interested in discrete systems and in particular in Discrete Event Systems (DES) which are discrete-states, events-driven systems, that is, their states evolution depends entirely on the occurrence of asynchronous discrete events over time in the centralized case and on both synchronous and asynchronous discrete events in the distributed case.

The discretization of states and events is widely used to represent dynamics of a system in an abstract manner due to its direct reflection to the human way of thinking and visualizing. Even continuous systems can be qualitatively abstracted to discrete ones in some ways. Actually this discretization facilitates modeling, analyzing and designing systems in order to do further steps like synthesizing, controlling these systems and evaluating their performance or optimizing them. The most popular computation models used to study the diagnosis problem are the Petri Nets (PN) and the Finite State Machines (FSM), or more generally the Labeled Transition Systems (LTS), which we will adopt as an input model due to its simplicity, However this model will be translated into a succinct representation in order to use SAT Solvers in this thesis.

2.2.1 Labeled Transition Systems

Labeled transition systems are formal models that are widely used to design and to represent real systems through abstracting their behaviors by sequences of transitions; thus they reflect how the system progresses between its different states respecting the design rules. In other words, the behavior of the system is represented by the language generated by the finite state machine. In their graphical representation, usually we denote states by circles and transitions by arrows that connect these circles, events appear as labels over the arrows.

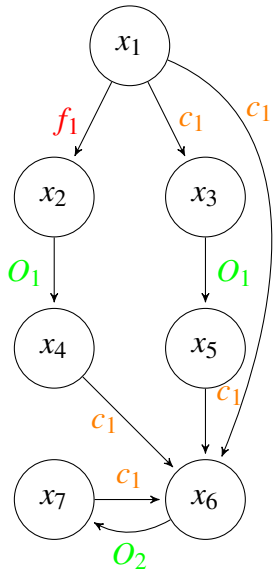

Fig. 2.1 An LTS example.

Example 1 (LTS example)**.** The figure 2.1 represents an LTS system with 7 states, two main types of events: observable like $\{O1, O2\}$ and unobservable like $\{f1, c1\}$ with $f1$ a faulty event.

We adopt the labeled transition system as a modeling formalism and we define it here following [Sampath *et al.* 1995] .

Definition 1. @ A **Labeled Transition System** (LTS) is a tuple $T = \langle Q, \Sigma, \delta, q^0 \rangle$ where:

- Q is a finite set of states,
- $\Sigma = \Sigma_o \cup \Sigma_u \cup \Sigma_f$ is a finite set of events,
- $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation,
- q^0 is the initial state.

with Σ_o the set of observable correct events, Σ_u the set of unobservable correct events and Σ_f the set of unobservable faulty events.

Demanding only one initial state q^0 is actually not a limitation: if there are several ones, one creates a unique new one and connects it by unobservable transitions to each of the previous ones. δ is trivially extended recursively to words in $\Sigma^\star$ (the Kleene closure of Σ): $\delta \subseteq Q \times \Sigma^\star \times Q$. Formally speaking, we denote by $L(T)$ the prefix-closed language generated by T, which is a subset of $\Sigma^\star$. Thus the system behaviors are represented by the set of words

of this language: $L(T) = \{\sigma \in \Sigma^\star | \exists q_s \in Q, (q^0, \sigma, q_s) \in \delta\}$. We call a path ρ_s in the system, any alternating sequence of states and events obtained from the initial state q^0 and ending with a state $q_s \in Q$: $\rho_s = q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} \cdots \xrightarrow{e_n} q_s$, with $e_1, \ldots, e_n \in \Sigma$, $q_0, q_1, \ldots, q_s \in Q$ and $\forall i \in \{0, \ldots, n-1\}, (q_i, e_{i+1}, q_{i+1}) \in \delta$. We denote the length of a path ρ_s by $|\rho_s|$.
A path ρ_s defines a (part of) system behavior σ_s, that can be viewed as a possible run and is called a trajectory of the system: σ_s is the abstraction of ρ_s on its sequence of events between the initial state q^0 and the last state q_s, i.e., $\sigma_s = e_1 \ldots e_n \in \Sigma^\star$ is the trajectory corresponding to ρ_s . Clearly this abstraction is not one-to-one in a non-deterministic system, thus the same sequence of events, or trajectory, can be assigned, in general, to more than one path. We call the projection of such a sequence on its observable events an observable sequence σ_o; it can be obtained by applying the instance P_o of the general projection operator, $P_a(.) : \Sigma^\star \to \Sigma_a^\star$, where Σ_a is the intended subset of events to retain, i.e., on which to abstract. $P_a(.)$ is defined as follows:

$$P_a(.) = \begin{cases} P_a(\varepsilon) = \varepsilon, & \varepsilon \text{ is the empty event,} \\ P_a(e) = \varepsilon, & \text{if } e \notin \Sigma_a, \\ P_a(e) = e, & \text{if } e \in \Sigma_a, \\ P_a(e\sigma) = P_a(e)P_a(\sigma), & \text{where } e \in \Sigma \text{ and } \sigma \in \Sigma^\star. \end{cases} \tag{2.2.1}$$

As a result, the set of possible system runs that can be related back to an observable sequence σ_o is defined like this: $P_o^{-1}(\sigma_o) = \{\sigma \in \Sigma^\star | P_o(\sigma) = \sigma_o\}$. One can notice that differentiating from each other the sets of faulty and non-faulty paths in a system, based on its observations, is highly related to the sizes of these sets of possible trajectories, which in their turn depend on the degree of non-determinism in the system that can be reduced each time a new observations arrives.
We denote the post-language of $L(T)$ after the trajectory σ by $L(T)/\sigma$, it represents the possible continuation of σ in the transition system T. $L(T)/\sigma = \{t \in \Sigma^\star | \sigma.t \in L(T)\}$.

Operations on Labeled Transition Systems

In order to study the diagnosability problem we will need to define formally some operations on the LTS, like synchronizing two LTS and performing a delay closure on a LTS.

Definition 2. (Synchronization) Given two LTS, $T_1 = \langle Q_1, \Sigma_1, \delta_1, q_1^0 \rangle$ and $T_2 = \langle Q_2, \Sigma_2, \delta_2, q_2^0 \rangle$, we define their synchronization $T_1 || _{\Sigma_s} T_2$, on a predefined synchronization set of events $\Sigma_s \subseteq \Sigma_1 \cap \Sigma_2$, as a new LTS $T_1 || _{\Sigma_s} T_2 = \langle Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{1||2}, (q_1^0, q_2^0) \rangle$ with $\delta_{1||2}$ defined as follows:

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

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

If we restrict the transition relation $\delta_{1||2}$ only to the first item, i.e., we accept only the synchronization set of events Σ_s in the result, then the operation is called a *product*. This simply can happen when we synchronize two copies of the same system on their common vocabulary.

Definition 3. (Delay Closure) Given an LTS $T = \langle Q,\Sigma,\delta,q_0\rangle$, its delay closure with respect to Σ_d, where $\Sigma_d \subseteq \Sigma$, is $\complement_{\Sigma_d}(T) = (Q_d,\Sigma_d,\delta_d,q_0)$, where:

- $(q,\sigma,q') \in \delta_d$ if $\sigma \in \Sigma_d$ and $\exists s \in (\Sigma\backslash\Sigma_d)^*, (q,s\sigma,q') \in \delta$.
- $Q_d = \{q_0\} \cup \{q \in Q \mid \exists q' \in Q_d, \exists \sigma \in \Sigma_d, (q',\sigma,q) \in \delta_d\}$.

Thus, this operation eliminates transitions labeled by events not in Σ_d and is identical to the classical silent transitions elimination in asynchronous automata.

2.3 Diagnosability

Diagnosability of the considered systems is a property defined to verify the possibility to distinguish any possible faulty behavior in the system from any other behavior without this fault (i.e., correct or with a different fault) within a finite time after the occurrence of the fault. A fault is diagnosable if it can be surely identified from the partial observation available in a finite delay after its occurrence. The first introduction to the notion of diagnosability was by [Sampath *et al.* 1995]. The authors studied diagnosability of LTS, which is defined in definition 1. They formally defined diagnosability like the following:

Definition 4. (Diagnosability) A fault f is diagnosable in a system T iff (if and only if)

$$\exists k \in \mathbb{N}, \forall s^f \in L(T), \forall t \in L(T)/s^f, |t| \geq k \Rightarrow$$
$$\forall p \in L(T), (P_o(p) = P_o(s^f.t) \Rightarrow f \in p.$$

In this formula, s^f denotes any word in $\Sigma^\star f$. The above definition states that for each trajectory s^f ending with fault f in T, for each t that is an extension of s^f in T with enough events, every trajectory p in T that is equivalent to $s^f.t$ in terms of observation should contain in it f.

In other words, the fault f is non-diagnosable iff it exists a pair of observation-equivalent infinite trajectories, one with f and the other without f. We call such a pair a *critical pair*. The absence of such a pair will provide information about fault signature. Finding such a pair will help in positioning the sensors to manage the observations requirements in order to increase diagnosability and ensure robust fault detection and identification.

2.3.1 General Assumptions

Following other studies about diagnosability we adopt the following set of assumptions to be hold in all this thesis.

Assumption 1. (Liveness) The language $L(T)$ of the trajectories of the studied transition system is live.

This means that for any state, there is at least one transition issued from this state. This assumption ensures that the post-language of $L(T)$ after any trajectory is never empty, so contains arbitrarily long sequences.

Assumption 2. (Convergence) The language $L(T)$ of the trajectories of the studied transition system is convergent.

This means that there is no cycle made up only of unobservable events. As studying diagnosability relies on the observations, so accepting infinite behaviors of the system without getting any observation makes the study meaningless.

A system T is said to be diagnosable iff any fault $f \in \Sigma_f$ is diagnosable in T. In order to avoid exponential complexity in the number of faults processed during diagnosability analysis, only one fault at a time is considered to check its diagnosability.

Assumption 3. (Fault uniqueness) There exists only one fault event f ($\Sigma_f = \{f\}$), without restriction on the number of its occurrences.

This is actually not a restriction. It just means that each fault will be processed separately and successively for checking its diagnosability. During this process, other faults are considered as unobservable correct events, allowing to get linear complexity in the number of faults.

Assumption 4. (Fault permanence) We assume that the studied faults are permanent.

This assumption is actually not necessary if one is interested only in detecting without ambiguity in finite time after its occurrence any fault that happened. But in general, the purpose of this detection is to trigger repair or reconfiguration actions because the fault is assumed to be still present, i.e., to be permanent. We will see later that diagnosability can be extended from the detection of simple events to the detection of complex surveillance patterns, and that it is easy to model intermittent faults with such patterns.

Other assumptions about system and fault modeling will be added when needed in the other chapters.

Next we review the main works done to check diagnosability in centralized and distributed DES. Other works can be seen as variations or optimization on these works and we will mention them in the next chapters in their appropriate places in this thesis.

2.3.2 Diagnosability Checking in Centralized DES

The main difficulty in diagnosability checking algorithms is related to the states number explosion since [Sampath *et al.* 1995] introduced an approach to test the diagnosability property by constructing a *deterministic diagnoser* (hence the potential exponential number of states), starting from a non-deterministic *pre-diagnoser*, and searching for a critical pair of paths, i.e., two arbitrarily long paths which are equivalent from an observation point of view and only one of them contains the fault. The existence of a critical pair witnesses the non-diagnosability while its absence proves the diagnosability of the studied fault. In the following, we recall the definition of these two structures and how such a critical pair can be found.

The pre-diagnoser is a structure that can be obtained by abstracting a system by its observations, while keeping in its states the trace of the fault occurrence. Formally it is defined like the following:

Definition 5. (Pre-Diagnoser) The pre-diagnoser of the system $T = \langle Q, \Sigma, \delta, q^0 \rangle$ is an observable LTS, denoted by $D = (Q_D, \Sigma_D, \delta_D, q_D^0)$, where:

- $Q_D \subseteq Q \times 2^{\Sigma_f}$ is the set of states (q, ℓ) with $q \in Q$ and $l \subseteq \Sigma_f$ that are reachable by δ_D from q_D^0 (see below);
- $\Sigma_D = \Sigma_o$ is the set of observable events;
- $\delta_D \subseteq Q_D \times \Sigma_D \times Q_D$ is the set of transitions defined by: $\delta_D = \{((q,\ell), e, (q',\ell'))$ $\in Q_D \times \Sigma_D \times Q_D | \exists$ a path $\rho = (q \xrightarrow{u_1} q_1 ... \xrightarrow{u_m} q_m \xrightarrow{e} q')$ in T, with $u_k \in \Sigma_u \cup \Sigma_f, \forall k \in \{1,...,m\}$, $e \in \Sigma_o$ and $\ell' = \ell \cup (\{u_1, \ldots, u_m\} \cap \Sigma_f)$;

- $q_D^0 = (q^0, \emptyset)$ is the initial state.

In the pre-diagnoser, we only keep the observable events and attach the fault information to each retained state as a fault label. Precisely, for a unique fault event f, if f has occurred from the initial state up to a given state, then the fault label for this state is $\{f\}$. Otherwise, it is empty. And as we consider only permanent faults, all reachable states from a state with a fault label $\{f\}$ will hold the same label. Figure 2.2 shows the pre-diagnoser of the system in Figure 2.1.

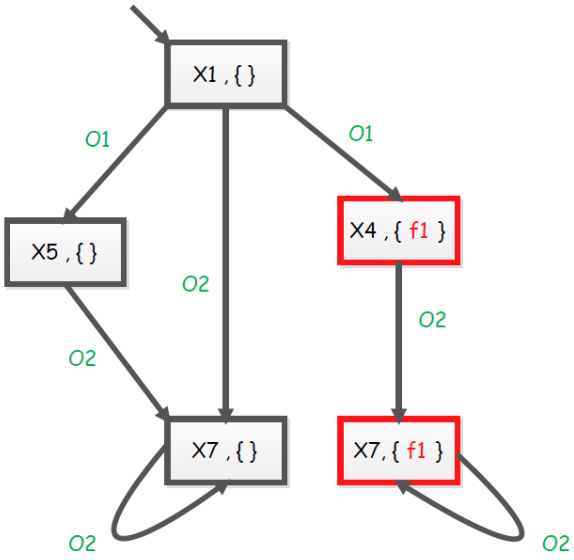

Fig. 2.2 The pre-diagnoser of the LTS in figure 2.1

After having built the pre-diagnoser, the deterministic diagnoser can be obtained by determinizing this structure, which consists in gathering all states that are reachable from a state by the same observable event in one state, while keeping the fault information in each state of the grouped state. We define the deterministic diagnoser as follows:

Definition 6. (Deterministic Diagnoser) Let $T = \langle Q, \Sigma, \delta, q^0 \rangle$ be an LTS and $D = (Q_D, \Sigma_D, \delta_D, q_D^0)$ its pre-diagnoser, we denote its deterministic diagnoser by $D_d = (Q_{D_d}, \Sigma_{D_d}, \delta_{D_d}, q_{D_d}^0)$, where:

- $Q_{D_d} \subseteq 2^{Q_D}$ is the set of states that are reachable by δ_{D_d} from $q_{D_d}^0$ (see below);
- $\Sigma_{D_d} = \Sigma_D = \Sigma_o$ is the set of observable events;
- $\delta_{D_d} : Q_{D_d} \times \Sigma_{D_d} \to Q_{D_d}$ is the transition function defined by:
  $\delta_{D_d}(q_d, \sigma) = \bigcup_{q_D \in q_d \wedge (q_D, \sigma, q'_D) \in \delta_D} \{q'_D\}$;

- $q^0_{D_d} = q^0_D = (q^0, \emptyset)$ is the initial state.

Since each state in this deterministic diagnoser is an aggregation of states from the states of the pre-diagnoser, which in their turn hold a unique fault label $\{f\}$ or empty, then a state $q_1 \in Q_{D_d}$ could hold different fault labels. Actually we have, in a general abstraction, three possible cases:

- F-certain state: in which all grouped (pre-diagnoser) states are with a fault label $\{f\}$. This means it can be reached only by passing by a fault occurrence.
- N-certain state: where all grouped (pre-diagnoser) states are with an empty fault label. This means it cannot be reached by passing by a fault occurrence.
- F-uncertain state: where only a sub-part of the grouped (pre-diagnoser) states has the fault label $\{f\}$ while the other part has the empty fault label.

We recall that a critical pair consists of two infinite paths (in the meaning of cycle), only one of them has the faulty event and they are equivalent in terms of observation. We call a cycle in D_d, an F-undetermined cycle iff:

- all its states are F-uncertain states;
- it has a corresponding cycle in the pre-diagnoser made up of only states with fault label $\{f\}$;
- it has a corresponding cycle in the pre-diagnoser made up of only states with fault label empty.

In order to search for a critical pair of trajectories in the deterministic diagnoser structure, we can restrict the search for a cycle which is F-undetermined since the determinism of the structure ensures the synchronization of the observations.

It is worth to notice that not each cycle of F-uncertain states in this structure is surely an F-undetermined cycle, while the second and third conditions together imply the first one.

However, in the general case, this approach is exponential in the number of states of the system, which makes it impractical for large systems. A polynomial algorithm to check diagnosability is proposed in [Jiang *et al.* 2001, Yoo & Lafortune 2002]. The authors introduced a new structure which is called Twin Plant. In order to build this structure we start from the pre-diagnoser for a given system, then we synchronize the pre-diagnoser with itself based on the observable events, i.e., each observable event should be synchronized, to obtain all pairs of trajectories issued from the initial state with the same observations.

Definition 7. The **Twin Plant** of the system T is the observable LTS $TP = D \parallel_{\Sigma_o} D$, where D is the pre-diagnoser of T.

As we said before, a synchronization process between two copies of the system on its set of events, is equal to a product operation. Each state of the Twin Plant is a pair of pre-diagnoser states that provide two possible diagnoses with the same observations. Given a Twin Plant state, if the fault f is contained in exactly one of the two associated pre-diagnoser states, which means that the occurrence of f is not certain up to this Twin Plant state with the same observations, it is called an ambiguous state with respect to f. An ambiguous state cycle is a cycle containing only ambiguous states. We define a *critical path* as a path in the Twin Plant issued from the initial state and made up of a prefix followed by an ambiguous state cycle. It corresponds exactly to a critical pair. So, after that, we verify the diagnosability by searching for such a critical path.

Lemma 1 ([Jiang *et al.* 2001, Yoo & Lafortune 2002])**.** *A system is non-diagnosable iff its Twin Plant contains a critical path.*

This result is illustrated on the figure 2.3 where the presence of the critical path $((x1,\{\});(x1,\{\})) \xrightarrow{O1} ((x4,\{f1\});(x5,\{\})) \xrightarrow{O2} ((x7,\{f1\});(x7,\{\})) \xrightarrow{O2} ((x7,\{f1\});(x7,\{\}))$ in the Twin Plant of the LTS example of the figure 2.1 proves that this system is not diagnosable.

2.3.3 Diagnosability Checking in Distributed DES

A distributed Discrete Event System, or DDES, is a system with a set of communicating components, each one of them can be represented as an LTS and they share a set of events among each other. Under the assumption of a global observation of the system, the author of [Pencolé 2004] proposed the first approach to check diagnosability of Distributed Discrete Event Systems. He considered communications to be correct events that are not observable. Formally speaking, he defined a DDES as a set of m local models $T_i, 1 \leq i \leq m$, sharing synchronous communication events, where a local model is defined as follows:

Definition 8. A **local Labeled Transition System** (lLTS) is a tuple $T_i = \langle Q_i, \Sigma_i, \delta_i, q_i^0 \rangle$ where:

- Q_i is a finite set of states,
- $\Sigma_i = \Sigma_{i_o} \cup \Sigma_{i_u} \cup \Sigma_{i_f} \cup \Sigma_{i_c}$ is a finite set of events occurring in T_i,
- $\delta_i \subseteq Q_i \times \Sigma_i \times Q_i$ is the transition relation,

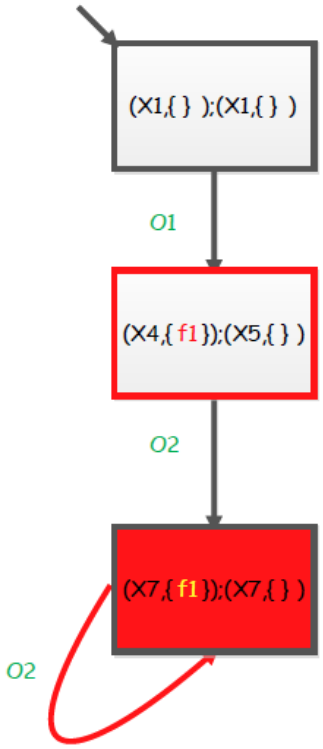

Fig. 2.3 Part of the Twin Plant built on synchronizing the pre-diagnoser in figure 2.2 with itself. In red the ambiguous state cycle witnessing non-diagnosability.

- q_i^0 is the initial state.

with Σ_{i_o} a finite set of observable correct local events, Σ_{i_u} a finite set of unobservable correct local events, Σ_{i_f} a finite set of unobservable faulty local events and Σ_{i_c} a finite set of communication events, the only ones to be shared by at least another local model of a neighboring component of T_i.

Figure 2.4 depicts a system with three components, that share only the communication events $\{c1, c2\}$.

Assumption 5. (Global observation) The system is globally observed.

This means that the observations in the system are globally ordered among the different components of a distributed system.

Assumption 6. (Synchronous communication) The communication events between the different components are synchronous.

In fact, studying asynchronous communication is out of the scope of this thesis.

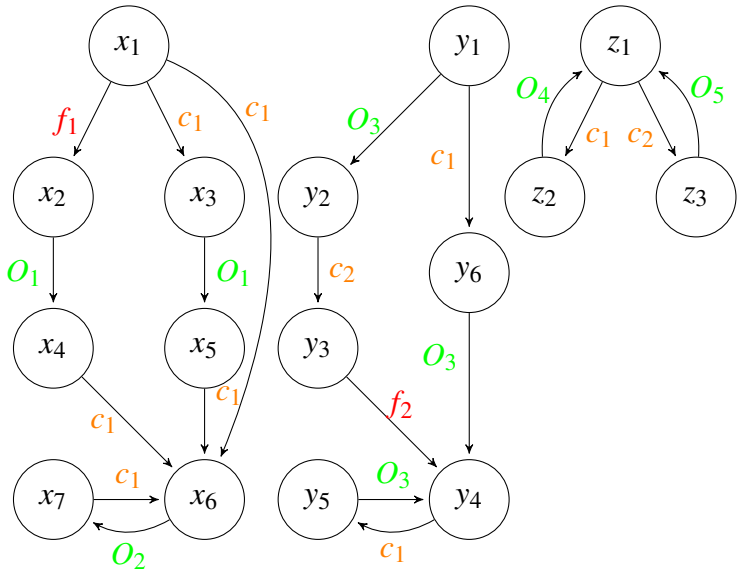

Fig. 2.4 Distributed DES with three components, that share only the events $\{c1, c2\}$

Assumption 7. (Communication correctness) The communication events between the different components are correct.

Notice that assuming communication events correct is not a restriction but a matter of modeling: if some communication event may be faulty, then the communication channel involved has just to be modeled as a new component by itself, containing at least one faulty local event.

Under the above assumptions, the problem of diagnosability in DDES is the one defined following the definition 4. Thus, it is to verify if the studied faults are diagnosable in the global system model (or a sub-part of it), which is the product of the local models synchronized on the communication events and on which delay closure with respect to these communication events is then applied: $T = \complement_{\Sigma_c}(||_{\Sigma_c} T_i)$, where $\Sigma_c = \cup_i \Sigma_{i_c}$ and $\Sigma_o = \cup_i \Sigma_{i_o}$, $\Sigma_u = \cup_i \Sigma_{i_u}$, $\Sigma_f = \cup_i \Sigma_{i_f}$. But one wants this verification to be achieved incrementally, starting at the level of the components without prior building of the global model.

The author of [Pencolé 2004] introduced an incremental diagnosability test which avoids building the Twin Plant for the whole global system if not needed. For this one starts by building a local Twin Plant for the faulty component to test the existence of a local critical path. If such a path does not exist, we know the system is diagnosable. But, if such a path exists, one should build local Twin Checkers of the neighboring components, i.e., those components which share communication events with the faulty one. The local Twin Checker is a structure similar to the local Twin Plant, i.e., where each path in it represents a pair of behaviors with the same observations, except that there is no fault information in it since it

is constructed from non-faulty components. After constructing local Twin Checkers, one tries to solve the ambiguity resulting from the existence of a critical path in the local Twin Plant, by synchronizing this local Twin Plant with the local Twin Checker of one neighbor on their communication events. In other words, one is trying to distinguish the faulty path from the correct one by exploiting the observable events in the neighboring components. Thus, the occurrences of observable events that are consistent with the occurrences of the communication events could solve the ambiguity. The process is repeated until the diagnosability is answered, which necessarily happens in the worst case when the whole system is visited. Another important contribution in this work was to delete the unambiguous parts after each synchronization on the communication events, in order to reduce the amount of information transferred to the next check (if needed).
The figure 2.5 depicts the local pre-diagnoser of the first component (on the left) of the system depicted in the figure 2.4 and the figure 2.6 part of its local Twin Plant. This one displays a local critical path, proving that the fault f_1 is not locally diagnosable in the first component.

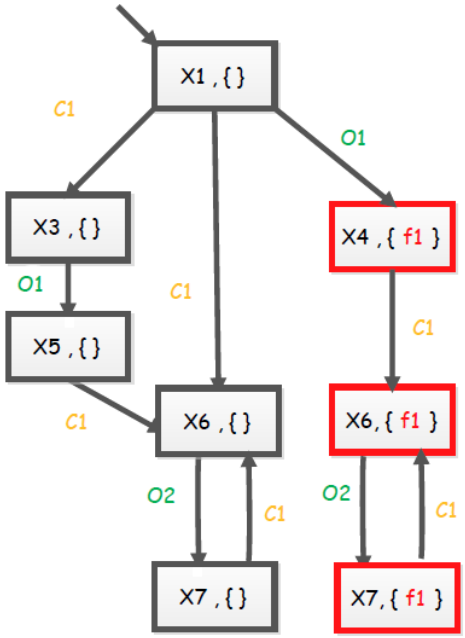

Fig. 2.5 The local pre-diagnoser of the first component (on the left) of the system depicted in the figure 2.4

The sizes of the considered parts of each local Twin Plant (or Twin Checker), also called local verifier, is reduced in the work of [Schumann & Pencolé 2007], where the authors describe the diagnosability problem as a distributed search problem. Thus, the global behavior is determined based on the local Twin Plants without computing any part of the

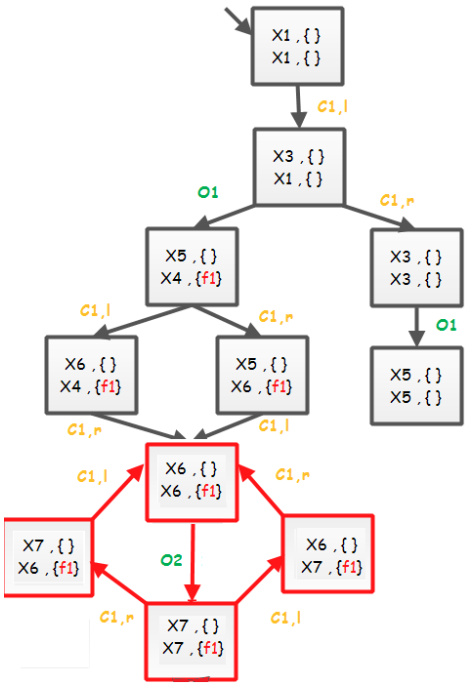

Fig. 2.6 Part of the local Twin Plant based on the local pre-diagnoser depicted in the figure 2.5

global Twin Plant. They propagate the fault information from the faulty component to other components by passing through a computable set of possibly non-diagnosable states in the different local Twin Plants of the different components depending on the connectivity between them and the faulty component. As a result, every state that is not possibly non-diagnosable is certainly diagnosable and so is deleted from the local Twin Plant, which leads to a *reduced* local Twin Plant. After that, the diagnosability of a fault is decided based on the set of distributed Twin Plants. Thus the fault is diagnosable iff none of the reduced Twin Plants contains an observable possibly non-diagnosable cycle (OPNC). A reduced Twin Plant is firstly obtained from the Twin Plant of each component in the system, then reduced Twin Plants are pairwise incrementally synchronized in order to remove remaining OPNCs to prove the diagnosability if possible, otherwise the approach gives a synthetic view of the non-diagnosability by returning all indistinguishable behaviors in the system. Thus one can deduce from the non-diagnosable states all possible critical pairs in the system. This approach is also adaptable to the available resources, thus it can stop when it runs out of memory and returns the current set of Twin Plants with OPNCs which contains all possible reasons for a potential non-diagnosability and which tells also that any set of the original components of the system which participated in any of the current reduced Twin Plants is not sufficient to diagnose the fault.

The work by [Ye & Dague 2010] has optimized the construction of local Twin Plants, by exploiting the fact that one distinguishes two behaviors (faulty and normal) and one synchronizes at two levels (observations first and communications later). The authors improved the construction of the Twin Plants proposed by [Pencolé 2004] by exploiting the different origin of the communication events (left and right copies) at the observation synchronization level to assign them directly to the two behaviors studied (left copy to the faulty behavior and right copy to the normal one). This helped in deleting the redundant information, then in abstracting the amount of information to be transferred later to next steps if the diagnosability is not answered.

Online/Offline Diagnosability Checking and Complexity

As we said before, the main problem while verifying diagnosability is to deal with states number explosion. This verification is usually done in an offline mode, in that the Twin Plant is first constructed, then a critical path is searched in it. Some recent approaches are proposed using Petri nets [Liu *et al.* 2014] to do the verification on-the-fly while constructing the Twin Plant and later by building a hybrid diagnoser for verifying diagnosability [Boussif *et al.* 2015] by combining enumerative and symbolic representations, passing by a symbolic observer graph [Haddad *et al.* 2004], in order to build a deterministic diagnoser where on-the-fly technique can be used to reduce the required time and memory resources in diagnosability verification. However, approaches that use the non-deterministic pre-diagnoser are still, to the best of our knowledge, to be done in an offline mode.

The complexity of the Twin Plant approach proposed in [Jiang *et al.* 2001] is polynomial of the 4th degree, in terms of states number. This can be seen easily, from its definition, where the number of states in the pre-diagnoser is bounded by $(|Q| \times 2^{|\Sigma_f|})$, then the number of states in the Twin Plant is bounded by $(|Q|^2 \times 2^{2|\Sigma_f|})$, which allows a search space of $(|Q|^4 \times 2^{4|\Sigma_f|} \times |\Sigma_o|)$. Thus finding a critical path in the Twin Plant is polynomial in the number of system states and exponential in the number of faults. Therefore, we consider one fault at a time while checking diagnosability of the system, as we mentioned in assumption 3. The worst case while checking diagnosability appears when the studied system is actually diagnosable. It implies proving the nonexistence of a counter-example witnessing non-diagnosability, i.e., all possibilities need to be tested as for proving the nonexistence of a plan in a planning problem, and usually in this case some approximations are used to avoid exploring all the search space, but we do not consider such approximations in this thesis. Testing diagnosability was proved to be NLOGSPACE-hard for enumerative representations, and PSPACE-hard for succinct (symbolic) representations [Rintanen 2007]. However, when using succinct representations, one can apply more abstract reasoning through using modern

efficient tools like BDD (Binary Decision Diagram) tools or model checkers and SAT solvers. Actually the reduction of the diagnosability problem to a path finding problem by [Jiang *et al.* 2001] made the problem transferable to a satisfiability problem like what is done in planning problems [Kautz & Selman 1992]. The authors in [Rintanen & Grastien 2007] formulated the diagnosability problem (in its Twin Plant version) into a SAT problem assuming a centralized DES with simple fault events. The work in this thesis can be considered as extensions and improvements over what they have done. We will review succinct transition systems used by their work in subsection 2.4.2.

2.4 SAT Problem

The satisfiability problem, or simply SAT problem, is a central theoretical problem in computer science, actually it is the canonical NP-problem [Cook 1971] in this field, It consists in answering the following question: given *Boolean variables* and a conjunction of a set of *clauses* from these variables, which forms a *propositional formula* in the so called *Conjunctive Normal Form*, denoted by CNF, is there a possible *assignment* of all these variables (or some of them), such that the logical formula takes the value *True* for this assignment.

Let us first define some of the terms in the last sentence. First, a Boolean variable, also called a propositional variable, is a symbol of 0-ary predicate which takes its values in the set $\{True, False\}$ or simply $\{1, 0\}$ of logical truth values. An assignment (resp. partial assignment) is a valuation of the variables (resp. of some of them). An assignment satisfying the given formula is called a *model* for this formula. A formula that owns a model is said to be satisfiable. A formula is built over a set of propositional variables by using the following logical operators or connectives:

- the binary conjunction operator *AND*, denoted by $\wedge$,
- the binary disjunction operator *OR*, denoted by $\vee$,
- the unary negation operator, denoted by $\neg$,
- the binary implication operator, denoted by $\rightarrow$,
- the binary equivalence operator, denoted by $\leftrightarrow$.

Actually, the operators above are not independent and we can restrict these operators to, e.g., only the disjunction and the negation operators, the other connectives being easily expressible

using these two operators like the following. Let Φ and Ψ be two logical formulas, these equivalences (denoted by $\equiv$) hold:

- $\Phi \wedge \Psi \equiv \neg(\neg\Phi \vee \neg\Psi)$,
- $\Phi \rightarrow \Psi \equiv \neg\Phi \vee \Psi$,
- $\Phi \leftrightarrow \Psi \equiv (\Phi \rightarrow \Psi) \wedge (\Psi \rightarrow \Phi)$

For satisfiability studying, it is more convenient to keep the negation, disjunction and conjunction operators, as this allows one to push the negation operators against the propositional variables. We call a literal a propositional variable or its negation. A clause is a disjunction of literals, thus it is satisfied by an assignment if at least one of its literals is set to *True* by this assignment. A CNF formula is a conjunction of clauses, thus to satisfy a CNF formula we must satisfy all its clauses, thus one unsatisfiable clause is sufficient to make the whole CNF formula unsatisfiable. Any logical formula can be polynomially translated into a CNF formula [Tseitin 1970] while keeping its satisfiability possibilities unchanged. Thus starting from a logical formula, we put it first in a Negation Normal Form (NNF), by simply pushing every negation operator applied on a subformula to its literals using the De Morgan laws, then we introduce new variables incrementally in order to substitute each one by each subformula in the NNF formula, starting from its deepest subformulas, without forgetting to ensure equivalent satisfiability property of the new formula by adding the equivalence between each introduced variable and the corresponding substituted subformula. This process is repeated incrementally until the whole formula is a conjunction of clauses, i.e., a CNF formula.

Example 2 (Tseitin transformation)**.** Let $\Phi = ((a \wedge b) \vee (c \wedge \neg d)) \vee ((c \vee b) \vee \neg a)$. In order to transform it into a CNF, we will introduce x, y, z, w as auxiliary variables:

$$
\begin{aligned}
\Phi &\equiv (x \leftrightarrow (a \wedge b)) \wedge ((x \vee (c \wedge \neg d)) \vee ((c \vee b) \vee \neg a)) \\
&\equiv (y \leftrightarrow (c \wedge \neg d)) \wedge (x \leftrightarrow (a \wedge b)) \wedge ((x \vee y) \vee ((c \vee b) \vee \neg a)) \\
&\equiv (z \leftrightarrow (c \vee b)) \wedge (y \leftrightarrow (c \wedge \neg d)) \wedge (x \leftrightarrow (a \wedge b)) \wedge ((x \vee y) \vee (z \vee \neg a)) \\
&\equiv (w \leftrightarrow (z \vee \neg a)) \wedge (z \leftrightarrow (c \vee b)) \wedge (y \leftrightarrow (c \wedge \neg d)) \wedge (x \leftrightarrow (a \wedge b)) \wedge ((x \vee y) \vee w) \\
&\equiv (w \leftrightarrow (z \vee \neg a)) \wedge (z \leftrightarrow (c \vee b)) \wedge (y \leftrightarrow (c \wedge \neg d)) \wedge (x \leftrightarrow (a \wedge b)) \wedge (x \vee y \vee w)
\end{aligned}
$$

Each one of the added equivalences can be transformed using the above relations between the logical operators, like this:

$$f \leftrightarrow (g \vee h) \equiv (\neg f \vee g \vee h) \wedge (f \vee \neg g) \wedge (f \vee \neg h)$$
$$f \leftrightarrow (g \wedge h) \equiv (\neg f \vee g) \wedge (\neg f \vee h) \wedge (f \vee \neg g \vee \neg h)$$

Then, we obtain the following CNF formula that is satisfiability-equivalent to Φ:

$$\begin{aligned}\Phi \equiv & (\neg w \vee z \vee \neg a) \wedge (w \vee \neg z) \wedge (w \vee a) \\ & \wedge (\neg z \vee c \vee b) \wedge (z \vee \neg c) \wedge (z \vee \neg b) \\ & \wedge (\neg y \vee c) \wedge (\neg y \vee \neg d) \wedge (y \vee \neg c \vee d) \\ & \wedge (\neg x \vee a) \wedge (\neg x \vee b) \wedge (x \vee \neg a \vee \neg b) \\ & \wedge (x \vee y \vee w)\end{aligned}$$

We give here some simple illustrative example of a SAT problem:

Example 3 (SAT Instance)**.** Let x, y, z be propositional variables and $\Phi = (\neg x \vee y) \wedge (\neg y \vee z)$ a CNF formula. Then, verifying the possibility to satisfy Φ, denoted by $Sat(\Phi)$?, is the SAT problem.
A solution to this problem is an assignment returned by the solver so that all the clauses of Φ are satisfied. Here, the assignment $x = 0, y = 1, z = 1$ is a possible solution. In such case the solver answers Yes, we say Φ is SAT.
Otherwise, if no assignment satisfies Φ, the solver returns No. We say in this case that Φ is UNSAT.

Many mathematical and computer science problems can be reduced to a SAT problem. Many industrial problems can be reduced to it also, the motivation of this reduction being twofold. First, many of these problems can be abstracted naturally to constraints problems that consist in satisfying a set of requirements where each requirement can be satisfied in several ways and where the ways for different requirements may contradict each other; therefore they can be easily mapped to a CNF formula. Second, a revolution occurred in the performances of modern SAT solvers in the last decade, in particular the Conflict Driven Clause Learning (CDCL) Solvers. The CDCL solvers were introduced in [Silva & Sakallah 1997], then an efficient SAT solver was introduced in [Moskewicz *et al.* 2001], which inspired the development of many very efficient solvers that became later reference solvers in the SAT competition like [Eén & Sörensson 2005, Audemard & Simon 2009]. Nowadays, CDCL solvers can handle problems with millions of clauses and hundreds of thousands of propositional variables. That is why we find SAT applications nowadays in many domains like:

- Formal methods, such as Bounded Model Checking [Biere *et al.* 1999], Test generation, etc.

- AI, as in Planning [Kautz & Selman 1992], Knowledge representation, Games.
- Design Automation, as in fault diagnosis [Grastien *et al.* 2007, Bjesse *et al.* 2001, Velev & Bryant 2003].
- Other applications in security, bio-informatics, mathematical problems and in the core of other constraints solvers: SMT, MAXSAT, #SAT, etc.

Moreover, SAT solvers can be indirectly used in our daily life like in the verification tasks in operating systems and in hardware design of processors. This success of the SAT technology is due to the effort of an active community of researchers that led to the aggregation of many well designed algorithms and heuristics that are engineered together to answer very efficiently a SAT problem.
Our work in this thesis does not contribute to the SAT research but uses this technology to deal with the studied problem, which may be seen as an additional application to this successful technology! Therefore, we will recall here its basics principles and only what we think is useful for the reader to understand our contributions.

2.4.1 SAT Algorithms and Heuristics

Let us first recall the principal rule of logical reasoning which is the *resolution rule*, also exploited in the SAT algorithms.

Resolution Rule

It consists in eliminating from each two clauses every branching variable, i.e., a variable that appears positively in one clause and negatively in the other one. Thus, picking up in a set of clauses two clauses $x \vee C$ and $\neg x \vee C'$ and adding to the set their so-called *resolvant* $C \vee C'$ does not change the satisfiability of the set of clauses:

$$\frac{x \vee C \quad \neg x \vee C'}{C \vee C'}$$

Example 4.

$$\frac{\neg x \vee y \quad \neg y \vee z}{\neg x \vee z}$$

Using the relations cited above between the logical connectives, we can read this example as: if x implies y and y implies z, then x implies z.

Then we have the following fundamental result: a set (conjunction) of clauses is unsatisfiable iff the empty clause can be produced from it by a repeated application a finite number of times of the resolution rule.

Unit Propagation

A special case of application of the resolution rule is when one of the two parent clauses is a *unit* clause, i.e., contains only one variable (either because already present like this in the input formula or as a result of previous steps of resolution rule application). This can help in shortening other clauses or even producing an empty clause that proves the unsatisfiability. Inspecting such unit clauses to exploit them when they appear is called unit propagation.

Davis and Putnam algorithm (1960) and DPLL (1962)

The first algorithm proposed to solve a SAT problem was by [Davis & Putnam 1960], denoted by DP60. It is simply an iterative application of the resolution rule, and waiting until either the formula is empty i.e. it is satisfiable or one clause becomes empty i.e. the formula is unsatisfiable. This algorithm requires a lot of memory, even with some elementary improvements like deleting pure literals (i.e. those who appear only in one sign in the formula) and shrinking clauses with unit propagation, thus DP60 was still impractical for large instances.

After that, a more scalable backtracking algorithm was introduced by [Davis *et al.* 1962], denoted by DPLL. It consists in ordering the set of variables and applying an iterative division of the search space through guessing the current variable's value: if the current guess leads to an empty clause the algorithm backtracks to the previous level and flips the guess in order to continue the search for a potential model. Thanks to the backtracking strategy this algorithm is complete. One can choose the ordering to be based on the variables frequencies in the formula, or simply adopt the lexicographical or anti-lexicographical orders. Moreover, this ordering can be different according to the branches. This approach can exploit in a better way pure literal deletion: as it chooses what to satisfy, the considered variables and so their clauses are deleted from the formula, and in the same way unit propagation is oriented to satisfy the remaining literal in each unit clause. As the corresponding variable can be found also in other clauses in the formula, this makes propagating its assigned value in all its occurrences in the formula a fruitful process; thus it could lead to satisfy other clauses in the formula or generate a conflict when the problem is unsatisfiable.

Example 5. (DPLL). Let the formula Φ be the conjunction of the following clauses:

$$\begin{aligned}
\phi_1 &= \neg x_1 \vee \neg x_5 \vee x_2 \\
\phi_2 &= \neg x_3 \vee \neg x_4 \\
\phi_3 &= x_1 \vee \neg x_4 \vee x_3 \\
\phi_4 &= \neg x_4 \vee \neg x_2 \\
\phi_5 &= x_3 \vee \neg x_5 \\
\phi_6 &= x_1 \vee x_5 \vee x_4 \vee x_2
\end{aligned}$$

If we adopt the lexicographical ordering, $x_1 \leq x_2 \leq x_3 \leq x_4 \leq x_5$, we get the following search tree:

1	$x_1 = 0 \Rightarrow$	$\{\phi_1$ *satisfied*$\}$				
2		$x_2 = 0 \Rightarrow \{\phi_4$ *satisfied*$\}$				
3			$x_3 = 0 \Rightarrow$	$\{\phi_2$ *satisfied*$\}$		
4				$x_4 = 0 \Rightarrow \{\phi_3$ *satisfied*$\}$		
5					$x_5 = 0 \Rightarrow$	$\{$ *conflict*$\}$
6					$x_5 = 1 \Rightarrow$	$\{$ *conflict*$\}$
7				$x_4 = 1 \Rightarrow \{$ *conflict*$\}$		
8			$x_3 = 1 \Rightarrow$	$\{\phi_3, \phi_5$ *satisfied*$\}$		
9				$x_4 = 0 \Rightarrow \{\phi_2$ *satisfied*$\}$		
10					$x_5 = 0 \Rightarrow$	$\{$ *conflict*$\}$
11					$x_5 = 1 \Rightarrow$	$\{\phi_6$ *satisfied*$\}$

Notice that the algorithm backtracks after steps $5, 6, 7$ and 10, then in step 11 returns the assignment $\{x_1 = 0, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 1\}$ as a model of the formula Φ.

Although DPLL is more scalable than DP60, it is still impractical for real world instances.

Main components of a modern SAT Solver

The SAT solvers knew their current revolution after introducing learning techniques over the DPLL algorithm [Silva & Sakallah 1997] and adding many other intelligent heuristics

and data structures that improved their performance and scalability [Moskewicz *et al.* 2001, Eén & Sörensson 2005]. The paradigm of a modern SAT solver can be seen as a backtracking algorithm that tries systematically to assign values to the propositional variables in the input formula, preceded by an important step of preprocessing and empowered by four main components, briefly explained in the following, which are:

- Branching component: which variable to assign now?
- Learning component: in case of conflict, what should be learned as a "new" constraint about the problem?
- Clause Management component: after having failed a lot of time and learned a lot of conflicts, which clauses are the more informative to maintain in the memory and how to store them?
- Restarting component: restarting the search is not a disruptive process, as many important pieces of learned information are stored in the new setting of the problem and this can help to get shorter proofs, but when to restart the search?

The answers to the questions posed by these components can help in reducing significantly the search space. Actually, these questions are answered in the SAT community in many different ways. These ways are filtered and they are more and more stable, due to the active research community of satisfiability problems.

Let us start by the Learning component. The purpose of this module is to exploit the solver failed tries while searching for a potential valuation that satisfies the formula. In other words, each variable assignment of a variable is either a part of a model or not. After a series of successive assignments (a series of decisions), we get a partial assignment of the variables, and in the same way it is either "correct" and then leads to a model or incorrect due to a discovered conflict. The CDCL solvers, as the name indicates, are based on analyzing such conflicts in order to correct previously made decisions. Thus they learn new constraints about the problem, which were implicitly encoded in the input CNF formula. The following example and the associated figure 2.7 illustrate how the CDCL algorithm analyses a conflict.

Example 6. (CDCL example) Let the CNF formula Φ consisting of the following clauses:

$$\phi_1 = \neg a \vee b \qquad \phi_2 = \neg b \vee l \qquad \phi_3 = \neg b \vee c \vee d \qquad \phi_4 = \neg b \vee \neg d \vee \neg h$$
$$\phi_5 = \neg f \vee e \qquad \phi_6 = \neg d \vee f \vee g \qquad \phi_7 = \neg g \vee h \vee \neg j \vee i \qquad \phi_8 = \neg i \vee f \vee k$$
$$\phi_9 = \neg l \vee m \vee \neg i \qquad \phi_{10} = \neg k \vee \neg n \qquad \phi_{11} = n \vee p \qquad \phi_{12} = \neg m \vee \neg p$$

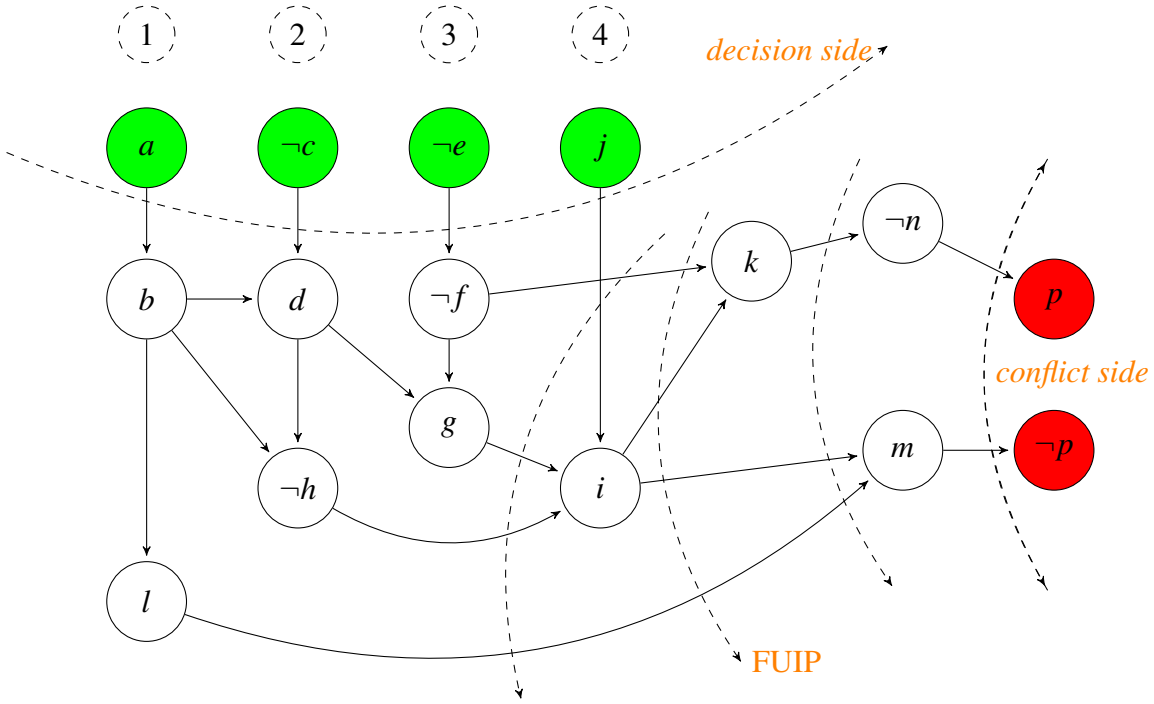

Fig. 2.7 Implication graph of an application of the CDCL example 6.

After a series of decisions, $a@1, \neg c@2, \neg e@3, j@4$, the solver will detect the empty clause from the conflict $p \wedge \neg p$, which indicates that $a \wedge \neg c \wedge \neg e \wedge j$ is a conflict, i.e., cannot be satisfied. The algorithm can learn this conflict by adding the clause $\neg a \vee c \vee e \vee \neg j$ to the original formula. However, this empty clause can be reproduced by many ways through decisions of variables participating in its generation and which are closer to it than our decided variables, and so faster in deriving it. Which means that many conflict clauses can represent what we intend to learn, i.e., excluding an assignment that leads to the same empty clause. These different representations can be obtained from the different possible bi-partitions (cuts) of the implication graph into a conflict side and a decision or reason side. We aim at making acquired knowledge as relevant as possible to the conflict. For this, we search the Unique Implication Points (UIPs, i.e. any node at the current decision level such that any path from the decision variable to the conflict node must pass through it) in the implication graph and we choose the first one (FUIP), here i. In other words we add to the clauses data base the new clause whose literals are the negation of all the variables reachable by paths from the reason side and having edges towards the conflict side crossing the FUIP cut. In our example the learned conflict clause is $C = f \vee \neg i \vee \neg l$.

Actually, the FUIP cuts can help also in improving the backtracking in the CDCL algorithm. Thus instead of just flipping the last decision variable, we can jump to the maximum decision level of the learned conflict literals which are different from the FUIP variable. In our example the FUIP variable is i, then in this case we can backtrack to the level max(f@3,l@0)= 3, and flip the value of its decision variable, i.e., set $e = 1$, instead of backtracking to $j = 0$. This method is called back-jumping or non-chronological backtracking [Moskewicz *et al.* 2001].

In the Branching component, most of the modern solvers adapt the Variable State Independent Decaying Sum (VSIDS) heuristic [Moskewicz *et al.* 2001] to choose the next variable to branch the formula on it. In this heuristic, a score is assigned to each variable and the most frequent variables in the last conflict clauses are rewarded, by increasing their scores, by an analysis done in a predefined slot of time. This slot has an order of $1/30s$, actually it corresponds to the application domain of the SAT instance.

In the Clause Management component, the lazy two watched literals (2WL) data structure answers the question how to store the clauses. Actually, the VSIDS heuristic was introduced in a package of the well engineered solver Chaff, in order to guide the search and compensate the 2WL structure, which drops monitoring all literals in each clause and thus makes the search blind! From another side the 2WL structure saves a lot of memory. It depends on the fact that any N-literals clause is not "useful" before it becomes a unit clause. Therefore watching only two of its literals is enough to know when it will become a unit clause. In fact, the unit propagation is triggered only if one of the watched literals is set to *False* and there is not any left non-falsified literal to watch in the clause. Even though keeping the learned clauses is important to ensure completeness of the solver, however they add additional memory overhead because of their huge number. It is important to handle this problem in the Clauses Management component. To this end, the authors in [Audemard & Simon 2009] observed that not all learned clauses have the same quality and they defined a measure of clauses quality, called Literals Blocks Distance (LBD), which depends on the number of blocks which a learned clause's literals are partitioned in. A block is defined by the decision level. In other words, a learned clause whose literals are separated in two blocks is more important to save than another clause whose literals are separated in more than two blocks. In fact, they use a fixed value of LBD = 2 in their solver Glucose which won the SAT competition in 2009, when they introduced this idea.

In the Restarting component, many strategies appeared to serve the fact that restarting helps in finding shorter proofs through exploitation of the new learned conflicts [Haim & Heule 2014, Gomes *et al.* 2000, Huang 2007]. Restarts slots usually grow slowly until being large enough to ensure completeness and they can make huge different on different benchmarks. One keeps track of the phase or the sign of assigned literals and one branches first on the recorded phase when taking a decision in the new restart. Adaptive restart strategies are adopted since the 2009 SAT competition, like being reviewed in the light of the quality of the learned clause [Audemard & Simon 2009] and many other strategies.

As a result of these components, the modern SAT solvers can be seen as clauses generators that can compromise between learning good new constraints about the problem and keeping the clauses data structure agile, free from an exhaustive overhead of storage and management of a huge number of learned clauses. Actually, one may accept relaxing the completeness insurance if one is sure that the problem is well encoded and so that all needed information is implicitly in the encoding, in such a way that the solver will discover this hidden information, which is usually the case of well described industrial instances.

We should say that preprocessing the formula before feeding it to the solver can significantly reduce its complexity and the SatELite preprocessor is the de facto standard preprocessor since 2005 used in most solvers. It consists in variables and clauses elimination process [Fourdrinoy *et al.* 2007, Eén & Biere 2005].

Incremental SAT

In many applications, the logical formula Φ may contain a set of hypotheses (in terms of literals) $\mathscr{H}$ that should be taken into account during satisfiability testing, thus the testing process consists in doing a set of similar tests of the form $\{SAT(\Phi \wedge h_i) | h_i \in \mathscr{H}\}$. In other words the SAT solver is supposed to handle some of the problem constraints, especially those which are independent of the current h_i, in each call and re-learn their corresponding conflicts again and again, which makes recycling the previous learned conflicts a good idea to improve the performance and save time.

In order to make the conflict learned during the test under an assumption h_i usable in further tests, where h_i may hold or not, every learned conflict must keep the trace of all assumptions involved in its deduction. To this end, we add the hypotheses to each clause of the original formula Φ, then when the solver gets a new input formula to test of the form $SAT(\Phi \wedge \bigcap h_i)$, it will first apply the assumptions in the first level of decision to filter the

learned clauses and keep only those which are consistent with the current assumptions, then it will proceed to classical test. In this way, we can ensure the soundness of the test and save the time that would be needed to discover what is already known from previous calls. The next example shows in detail how this process can be done.

Example 7. (Incremental SAT) Let us consider the formula Φ of the example 6. Suppose that we want, for some reason, to test the satisfiability of the first four clauses under the assumption x and similarly the second four clauses under the assumption y, and the last four clauses under the disjunction of the assumptions x and z: $\Phi' = (\bigwedge_{1\leq i\leq 4} \phi_i \wedge x) \wedge (\bigwedge_{5\leq i\leq 8} \phi_i \wedge y) \wedge (\bigwedge_{9\leq i\leq 12} \phi_i \wedge (x \vee z))$. Thus, in this way each learned clause will hold the trace of assumptions used while testing the formula.

2.4.2 Succinct Transition Systems

Succinct Transition Systems were introduced in [Rintanen & Grastien 2007] in order to compress the representation of the system states and allow a maximum amount of non-interfering events to be fired simultaneously. Thus, the system states are represented by the valuations of a set of Boolean state variables (n states need $\lceil log(n) \rceil$ state variables). The events are represented by their occurrences as a pair of preconditions and effects for each occurrence. The interference relation between two events is represented depending on the consistency between the events effects or between the effects of one event and the preconditions of the other one.

Definition 9. A **Succinct Transition System** (SLTS) is described by a tuple $T = \langle A, \Sigma_o, \Sigma_u, \Sigma_f, \delta, s_0 \rangle$ where:

- A is a finite set of propositional state variables,
- Σ_o is a finite set of observable correct events,
- Σ_u is a finite set of unobservable correct events,
- Σ_f is a finite set of unobservable faulty events,
- $\delta : \Sigma = \Sigma_o \cup \Sigma_u \cup \Sigma_f \rightarrow 2^{\mathscr{L} \times 2^L}$ assigns to each event a set of pairs $\langle \phi, c \rangle$, where each pair represents an occurrence of the event such that precondition ϕ is given by a formula in $\mathscr{L}$, the propositional language built on A, that has to be satisfied by the source state of the transition and effects c are given by a set of elements in L, the literals built from A, expressing the positive and negative changes of the valuation of the destination state of the transition w.r.t. the valuation of its source state,

- s_0 is the initial state (a valuation of A).

It is straightforward to show that any LTS can be represented as an SLTS: one takes $\lceil log(|Q|)\rceil$ Boolean variables and represents states by different valuations of these variables; one assigns to each occurrence of an event e labeling a transition (x,e,y) a pair $\langle \phi, c\rangle$, with ϕ expressing the valuation of x and c the valuation changes between x and y. And reciprocally, any SLTS can be mapped to an LTS (see Definition 2.4 in [Rintanen & Grastien 2007]).

2.4.3 SAT-based Diagnosability Encoding

Back to our studied property, an immediate rephrasing of the definition 4 shows that T is non-diagnosable iff it exists a pair of trajectories corresponding to cycles (and thus to infinite paths), a faulty one and a correct one (due to the fault uniqueness assumption 3), sharing the same observable events. Which is equivalent to the existence of an ambiguous (i.e. made up of pairs of states respectively reachable by a faulty path and a correct path) cycle in the product of T by itself, synchronized on observable events, which is at the origin of the so called *Twin Plant* structure introduced in [Jiang *et al.* 2001].
This non-diagnosability test was formulated in [Rintanen & Grastien 2007] as a satisfiability problem in propositional logic. For this, the authors distinguished between an occurrence of an event in the faulty sequence and in the normal sequence by introducing two copies of it (faulty and normal) which the same idea that in [Jiang *et al.* 2001]. The difference is that this approach constructs the logical formula and the solver has the task to find a model if any, in other words the (progressive and not complete in general) construction of the Twin Plant is due to the solver and only the search space is given to it, provided with diagnosability property encoding. Thus, for each possible step in the system it may contain simultaneous events that belong to faulty and normal sequences but must synchronize the occurrences of observable events whenever they take place. After that, the authors consider n steps by doing the conjunct of their formulas, then add to the result the logical formula that represents the occurrence of a cycle in both sequences (normal and faulty) after the occurrence of the fault in the faulty sequence. At last they feed the resulted formula to a SAT Solver. The satisfiability of this formula is equivalent to finding a critical path, i.e., to the non-diagnosability of the fault.

We recall below this encoding with the variables and the formulas used, where superscripts t refer to time points and (e_o^t) and $(\hat{e}_o^t)$ refer respectively to the faulty and correct events occurrences sequences (corresponding states being described by valuations of (a^t) and $(\hat{a}^t)$)

of a pair of trajectories witnessing non-diagnosability (so sharing the same observable events represented by (e^t) and forming a cycle). The increasing of the time step corresponds to the triggering of at least one transition and the extension by an event of at least one of the two trajectories. $T = \langle A, \Sigma_o, \Sigma_u, \Sigma_f, \delta, s_0 \rangle$ being an SLTS, the propositional variables are thus:

- a^t and $\hat{a}^t$ for all $a \in A$ and $t \in \{0, \ldots, n\}$,
- e_o^t for all $e \in \Sigma_o \cup \Sigma_u \cup \Sigma_f$, $o \in \delta(e)$ and $t \in \{0, \ldots, n-1\}$,
- $\hat{e}_o^t$ for all $e \in \Sigma_o \cup \Sigma_u$, $o \in \delta(e)$ and $t \in \{0, \ldots, n-1\}$,
- e^t for all $e \in \Sigma_o$ and $t \in \{0, \ldots, n-1\}$.

The following formulas express the constraints that must be applied at each time step t or between t and $t+1$.

1. The event occurrence e_o^t must be possible in the current state:

$$e_o^t \rightarrow \phi^t \quad \text{for } o = \langle \phi, c \rangle \in \delta(e) \tag{2.4.1}$$

   and its effects must hold at the next time step:

$$e_o^t \rightarrow \bigwedge_{l \in c} l^{t+1} \quad \text{for } o = \langle \phi, c \rangle \in \delta(e) \tag{2.4.2}$$

   We have the same formulas with $\hat{e}_o^t$.

2. The present value (*True* or *False*) of a state variable changes to a new value (*False* or *True*, respectively) only if there is a reason for this change, i.e., because of an event that has the new value in its effects (so, change without reason is prohibited). Here is the change from *True* to *False* (the change from *False* to *True* is defined similarly by interchanging a and $\neg a$):

$$(a^t \wedge \neg a^{t+1}) \rightarrow (e^t_{i_1 o_{j_1}} \vee \cdots \vee e^t_{i_k o_{j_k}}) \tag{2.4.3}$$

   where the $o_{j_l} = \langle \phi_{j_l}, c_{j_l} \rangle \in \delta(e_{i_l})$ are all the occurrences of events e_{i_l} with $\neg a \in c_{j_i}$. We have the same formulas with $\hat{a}^t$ and $\hat{e}^t_{i_{l o_{j_l}}}$.

3. At most one occurrence of a given event can occur at a time and the occurrences of two different events cannot be simultaneous if they interfere (i.e., if they have two

contradicting effects or if the precondition of one contradicts the effect of the other):

$$\neg(e^t_o \wedge e^t_{o'}) \quad \forall e \in \Sigma, \forall \{o,o'\} \subseteq \delta(e), o \neq o' \tag{2.4.4}$$

$$\neg(e^t_o \wedge e'^t_{o'}) \quad \forall \{e,e'\} \subseteq \Sigma, e \neq e', \forall o \in \delta(e), \forall o' \in \delta(e'), o \text{ and } o' \text{ interfere} \tag{2.4.5}$$

We have the same formulas with $\hat{e}^t_o$.

4. The formulas that connect the two events sequences require that observable events take place in both sequences whenever they take place (use of e^t for synchronization):

$$\bigvee_{o \in \delta(e)} e^t_o \leftrightarrow e^t \text{ and } \bigvee_{o \in \delta(e)} \hat{e}^t_o \leftrightarrow e^t \quad \forall e \in \Sigma_o \tag{2.4.6}$$

5. To avoid trivial cycles (silent loops with no state change) we require that at every time point at least one event takes place:

$$\bigvee_{e \in \Sigma_o} e^t \vee \bigvee_{e \in \Sigma_u \cup \Sigma_f, o \in \delta(e)} e^t_o \vee \bigvee_{e \in \Sigma_u, o \in \delta(e)} \hat{e}^t_o \tag{2.4.7}$$

The conjunction of all the above formulas for a given t is denoted by $\mathscr{T}(t,t+1)$.
A formula for the initial state s_0 is:

$$\mathscr{I}_0 = \bigwedge_{a \in A, s_0(a)=1} (a^0 \wedge \hat{a}^0) \wedge \bigwedge_{a \in A, s_0(a)=0} (\neg a^0 \wedge \neg \hat{a}^0) \tag{2.4.8}$$

At last, the following formula can be defined to encode the fact that a pair of trajectories is found with the same observable events and no fault in one trajectory (first line (2.4.9)), but the fault f (recall that $\Sigma_f = \{f\}$) in the other (second line (2.4.10)), which are infinite (in the form of cycles at step n, necessarily non-trivial from (2.4.7) and thus containing at least one

observable event from assumption 2; third line (2.4.11)), witnessing non-diagnosability:

$$\Phi_n^T = \quad \mathscr{I}_0 \wedge \mathscr{T}(0,1) \wedge \cdots \wedge \mathscr{T}(n-1,n) \quad \wedge \tag{2.4.9}$$

$$\bigvee_{t=0}^{n-1} \bigvee_{o \in \delta(f)} f_o^t \quad \wedge \tag{2.4.10}$$

$$\bigvee_{m=0}^{n-1} \left(\bigwedge_{a \in A} ((a^n \leftrightarrow a^m) \wedge (\hat{a}^n \leftrightarrow \hat{a}^m)) \right) \tag{2.4.11}$$

From this encoding in propositional logic, follows the result (theorem 3.2 of [Rintanen & Grastien 2007]) that an SLTS T is not diagnosable if and only if $\exists n \geq 1, \Phi_n^T$ is satisfiable. It is also equivalent to $\Phi_{2^{2|A|}}^T$ being satisfiable, as the Twin Plant states number is an obvious upper bound for n, but often impractically high. However the authors notice that one is not forced to test all reachable states in many cases where an approximation for the reachable states can be applied, but without explaining how such approximation can be found (see in [Rintanen & Grastien 2007] some ways to deal with this problem).

Actually the results presented in this work show a good scalability in comparison with the twin plant approaches which they said to be impractical for system with number of states larger than $m = 10000$. And as we see above this approach considered the centralized DES as there is no communication events in their model. This motivated us to consider this approach our starting point, and try to add communication events to consider the distributed DES case and to study the effect of employing incremental SAT mode in both centralized and distributed cases through providing the experimental results of our study, which will be presented in Chapter 3. From another side, we noticed that many similar problems like predictability (see Chapter 4) and pattern diagnosability (see Chapter5) can be encoded using similar techniques without passing by sophisticated structure used in the literature to treat such cases. Especially that, the results presented in [Ye 2011] manifest a rapid explosion of their methods even for small systems and patterns sizes. Although, in our personal communication with author she justified this explosion by an inefficient implementation, but we still think that the SAT-based approaches can improve the scalability and facilitate the properties analysis.

Chapter 3

SAT-Based Diagnosability Analysis in Distributed DES

In this chapter, we will present our first and second contributions. The first one is the extension of the existing SAT encoding in [Rintanen & Grastien 2007], which concerns the centralized DES, in order to deal with the Distributed DES (DDES), where we consider a set of communicating components using a set of synchronous communication events. We handle in our encoding the case where the communication events set is not purely observable or purely unobservable, thus it can be a mixture of both types. The second contribution is adapting the encoding to use incremental SAT solver mode, which can help to reuse a previously founded knowledge while checking the diagnosability property. We will discuss our experimental results before ending this chapter by suggesting some improvements over the original encoding .

3.1 Motivation/Introduction

Nowadays, systems are getting more and more complex and one way to reduce their complexity is by breaking them down into many communicating components. Another need to distribute a system into different components could be the nature of the provided service by this system. However, this distribution implies additional difficulties in modeling the communication among components without making the management process more complicated. In particular, the fault diagnosis process should be precise in order to ensure the design requirements and functionalities. For example, in a distributed structure, the fault may occur in one component, but very often cannot be precisely identified using only the available observations in this faulty component. However, in the case of a synchronous

communication between this faulty component with its neighbors, the communication events have the role of a relay that connect components, therefore getting access to new observations is used to help in monitoring the system. If we consider a global observer of the system, it would be waiting for additional observations from other components to be able to identify the fault occurrence precisely. One question to be considered here is, how to know if such discriminating observation will be available in a finite time after the fault occurrence. In fact, such question needs to be answered since the design stage of a system. Another related question is, if the previous question has a negative answer, i.e., a discriminating observation may not arrive, a designer should be informed why it might happen. Actually these questions concern the diagnosability analysis of faults in DDES, which has been already studied [Pencolé 2004, Schumann & Pencolé 2007, Ye & Dague 2010] and some answers were provided that we presented as we reviewed these works in section 2.3.3.

One issue that could be always improved is the scalability of an approach, i.e., the size of systems that can be processed using a specific approach. In this context, we explore here the usage of a generic approach based on the propositional logic; in particular we aim at exploiting the big success of the SAT solving technology, as we showed in section 2.4, especially as this technology has been studied in [Rintanen & Grastien 2007] but only considering centralized DES. We aim also at exploiting the flexibility of altering (modifying, adding) constraints in the encoding, then letting the SAT solvers handle them in order to a more efficient diagnosability analysis. This way could be easier than altering an algorithm to take into account some specific considerations. For example, in all the above studies about diagnosability in DDES, the authors considered a purely unobservable set of communications; actually they should add additional processing to handle the case of a mixed input set of both observable and unobservable communications, like for example, that only unobservable communications should be differentiated between the two copies of the local pre-diagnoser while synchronizing them on their observations. Such a technical detail can reduce an important amount of new states which are built in the different twin structures, by differentiating the communications of the faulty system copy from those related to the correct system copy. Things can be more complex when communication and observation *features* of events have different implications on the studied property as we will see an example in predictability analysis in chapter 4. We will show how we can simply add some constraints to treat such details by using the succinct representation of the system and encoding them into SAT.

In order to model DDES with SLTS, we need to extend these ones by adding communication events *to each component*.

3.2 Distributed Succinct Transition Systems

3.2.1 Modeling

Firstly, we define a communication event as a shared event between at least two different components. As we mentioned before, we are interested in synchronous communication events. Thus, we assume that firing such event in one component requires doing the same in all the components having this event in their events sets. In other words, from an enumerative representation point of view, communication events are synchronized such that they all occur simultaneously in all components where they appear. More precisely, a transition by a communication event c may occur in a component iff a simultaneous transition by c occurs in all the other components where c appears (has at least one occurrence). In particular, all events before c in trajectories in all these components necessarily occur before all events after c in these trajectories. The implicit global model of the system is thus nothing else that the synchronization, on the communication events, of the models of the components, where these communication events are then eliminated by delay closure.

From a constraints programming point of view, to be able to bring out a communication event in one component, exactly one occurrence of this event, in each other owner component, must have its required preconditions held to allow the firing of this event. Therefore, the definition of such communication events would be dependent of the whole system state. Thus, for each communication event, all its owners would participate in its definition. As we have recalled in section 2.4.3, the succinct representation of the system, as introduced in [Rintanen & Grastien 2007], defines, for each event, a set of occurrences and, for each occurrence, its relation with all other occurrences in the system. Therefore, it implies many constraints to ensure a relation consistent with the system design. Thus, it requires that every event has some preconditions and must have some effects and also that at most one occurrence of a given event can occur at a time. After that, it defines the interference between two events, in order to prevent it, when their effects contradict or when the effects of one contradict the precondition of the other (i.e., they share one variable but with two opposite signs). By introducing the communication events, one may add them directly to the system and handle all these constraints for each one. As a result, one can ensure the synchronization of communication events but this will complicate designing the communication events in

many ways, especially in considering all involved components for each constraint and each communication event and also in adding or removing some of these components. From a SAT point of view, encoding the corresponding constraints will generate long clauses, which is not recommended for SAT solvers.

We propose pushing the potential complexity of communication events modeling to the SAT solvers. Thus we will deal with these events like any other event in the system and we will encode synchronization separately (such technical detail would not be as easy to deal with in an enumerative representation of the system). Therefore, we define a distributed SLTS with k different components (sites) as:

Definition 10. A **Distributed Succinct Transition System** (DSLTS) with k components is described by a tuple $T = \langle A, \Sigma_o, \Sigma_u, \Sigma_f, \Sigma_c, \delta, s_0 \rangle$ where (subscripts i refer to component i):

- A is a union of disjoint finite sets $(A_i)_{1 \leq i \leq k}$ of component own state variables, $A = \cup_{i=1}^{k} A_i$,
- Σ_o is a union of disjoint finite sets of component own observable correct events, $\Sigma_o = \cup_{i=1}^{k} \Sigma_{oi}$,
- Σ_u is a union of disjoint finite sets of component own unobservable correct events, $\Sigma_u = \cup_{i=1}^{k} \Sigma_{ui}$,
- Σ_f is a union of disjoint finite sets of component own unobservable faulty events, $\Sigma_f = \cup_{i=1}^{k} \Sigma_{fi}$,
- Σ_c is a union of finite sets of (observable or unobservable) correct communication events, $\Sigma_c = \cup_{i=1}^{k} \Sigma_{ci}$, which are the only events shared by at least two different components (i.e., $\forall i, \forall c \in \Sigma_{ci}, \exists j \neq i, c \in \Sigma_{cj}$),
- $\delta = (\delta_i)$, where $\delta_i : \Sigma_i = \Sigma_{oi} \cup \Sigma_{ui} \cup \Sigma_{fi} \cup \Sigma_{ci} \rightarrow 2^{\mathscr{L}_i \times 2^{L_i}}$, assigns to each event a set of pairs $\langle \phi, c \rangle$ in the propositional language of the component i where it occurs (so, for communication events, *in each component separately* where it occurs), i.e. each pair represents an occurrence of the event such that precondition ϕ is given by a formula in $\mathscr{L}_i$, the propositional language built on A_i, and effects c are given by a set of elements in L_i, the literals built from A_i,
- $s_0 = (s_{0i})$ is the initial state (a valuation of each A_i).

Notice that we allow in whole generality communication events to be, partially or totally, unobservable, so one has in general to wait further observations to know that some

communication event occurred between two or more components. On the other side, as already commented with assumption 7, we do not consider assuming these communications to be faultless as a limitation. Thus, if a communication process or protocol may be faulty, it has just to be modeled as a proper component with its own correct and faulty behaviors (the same that, e.g., for a wire in an electrical circuit). In this sense, communications between components are just a modeling concept, not subject to diagnosis. The assumption 5 will also be adopted here: the observable information is global, i.e., centralized, allowing to keep definition 4 for diagnosability. In fact, when observable information is only local to each component, distributed diagnosability checking has been proved to be undecidable in general [Ye & Dague 2013].

3.2.2 Encoding DSLTS Diagnosability as Satisfiability Problem

Let T be a DSLTS made up of k components denoted by indexes i, $1 \leq i \leq k$. In order to express the diagnosability analysis of T as a satisfiability problem, we have to extend the formulas of subsection 2.4.3 to deal with communication events between components.
Let $\Sigma_c = \Sigma_{co} \cup \Sigma_{cu}$ be the communication events, with $\Sigma_{co} = \cup_{i=1}^{k} \Sigma_{coi}$ the observable ones and $\Sigma_{cu} = \cup_{i=1}^{k} \Sigma_{cui}$ the unobservable ones.

The idea is to consider each communication event as any other event in each of its owners and, as it has been done with events e^t for $e \in \Sigma_o$ for synchronizing observable events occurrences in the two trajectories, to introduce in a similar way a global reference variable for each communication event at each time step, in charge of synchronizing any communication event occurrence in any of its owners with occurrences of it in all its other owners. We use one such reference variable for each trajectory, e^t and $\hat{e}^t$, for unobservable events $e \in \Sigma_{cu}$, and only one for both trajectories, e^t, for observable events $e \in \Sigma_{co}$ as it will also in addition play the role of synchronizing observable events between trajectories exactly as the e^t for $e \in \Sigma_o$. So, we add to the previous set of propositional variables the new following ones:

- e_o^t, $\hat{e}_o^t$ for all $e \in \Sigma_c$, $o \in \delta(e) = \cup_i \delta_i(e)$ and $t \in \{0, \ldots, n-1\}$,
- e^t for all $e \in \Sigma_c$ and $t \in \{0, \ldots, n-1\}$,
- $\hat{e}^t$ for all $e \in \Sigma_{cu}$ and $t \in \{0, \ldots, n-1\}$.

Formulas in $\mathscr{T}(t, t+1)$ are extended as follows.

1. Formulas (2.4.1), (2.4.2), (2.4.3) and (2.4.5) extend to be applied *unchanged* to any event in each component i, in particular to e_o^t and $\hat{e}_o^t$ $\forall e \in \Sigma_c$, expressing that a

communication event must be possible, has effects in each of its owner components and that two such different events in this component i cannot be simultaneous if they interfere (notice that interference is not possible between two different components).

2. Formulas (2.4.4) extend to prevent two simultaneous occurrences of any event in each component i, in particular a given communication event in such component:

$$\neg(e^t_{o_i} \wedge e^t_{o'_i}) \quad \forall e \in \Sigma, \forall i, \forall \{o_i, o'_i\} \subseteq \delta_i(e), o_i \neq o'_i \tag{3.2.1}$$

and the same with $\hat{e}$. Obviously these constraints do not extend to different owner components of a communication event, by the very definition of communication events.

3. Finally, the new following formulas express the communication process itself, i.e., the synchronization of the occurrences of any communication event e in all its owner components. Let $S(e)$ be the set of indexes of the owner components of e, then formulas (2.4.6) extend to communication events as follows:

$$\bigvee_{o_i \in \delta_i(e)} e^t_{o_i} \leftrightarrow e^t \text{ and } \bigvee_{o_i \in \delta_i(e)} \hat{e}^t_{o_i} \leftrightarrow \hat{e}^t \quad \forall e \in \Sigma_{cu}\ \forall i \in S(e) \tag{3.2.2}$$

$$\bigvee_{o_i \in \delta_i(e)} e^t_{o_i} \leftrightarrow e^t \text{ and } \bigvee_{o_i \in \delta_i(e)} \hat{e}^t_{o_i} \leftrightarrow e^t \quad \forall e \in \Sigma_{co}\ \forall i \in S(e) \tag{3.2.3}$$

The formula defining Φ^T_n is unchanged. However the verification that the found cycles (third line (2.4.11)) contain at least one observable event can be made explicit, instead of relying as in [Rintanen & Grastien 2007] on formula (2.4.7) together with assumption 2. Following this new encoding, the formula (2.4.7) can be safely suppressed and the third line (2.4.11) in the definition of Φ^T_n should be replaced by:

$$\bigvee_{m=0}^{n-1} \Big(\bigwedge_{a \in A} ((a^n \leftrightarrow a^m) \wedge (\hat{a}^n \leftrightarrow \hat{a}^m)) \Big) \wedge \Big(\bigvee_{t=m}^{n-1} \bigvee_{e \in \Sigma_o \cup \Sigma_{co}} e^t \Big) \tag{3.2.4}$$

where the final disjunction of events e^t along the time steps of the cycles can exploit any observable event $e \in \Sigma_o \cup \Sigma_{co}$.

We have thus the result that a DSLTS T is not diagnosable if and only if $\exists n \geq 1, \Phi^T_n$ is satisfiable, which is also equivalent to $\Phi^T_{2^{2\Sigma_{i=1}^k |A_i|}}$ being satisfiable.

3.2.3 Implementation and Experimental Testing

Despite the fact that diagnosability problem was introduced twenty years ago and that many works have considered it in centralized and distributed structures, there is no standard benchmark available for testing purposes. However, many works considered specific benchmarks or toy examples to proof the new concepts or ideas. We have started our tests on the system in figure (3.1) (already presented in figure (2.4)), which was the running example in [Pencolé 2004], as it has some interesting variants (e.g., considering only a subset of the components; studying diagnosability either of fault f_1 or of fault f_2, the other one being thus considered as an unobservable correct event) to test different cases in our encoding. Thus, it contains several communication events with multiple occurrences (three communicating components) with either global communication (all three components share the same communication event c_1) or partial communication (only two components share the communication event c_2).

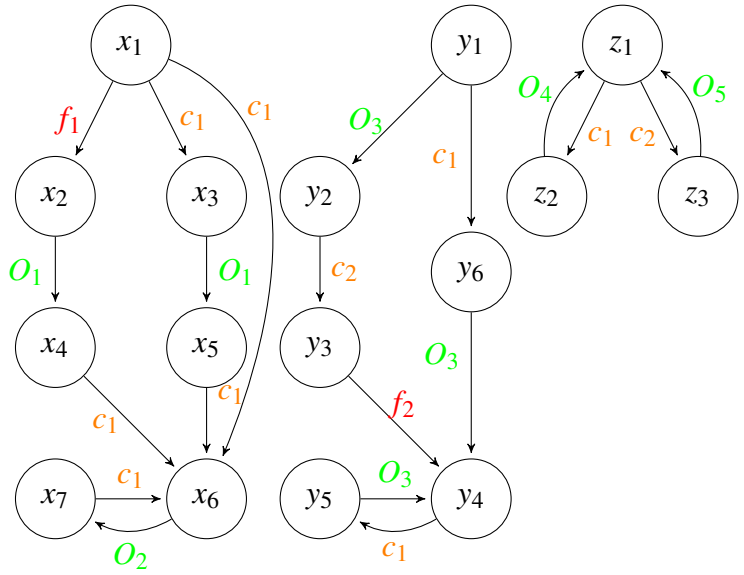

Fig. 3.1 A DDES made up of 3 components C1, C2 and C3 from left to right. $c_{i,1\leq i\leq 2}$ are unobservable communication events, $O_{i,1\leq i\leq 5}$ are observable events and $f_{i,1\leq i\leq 2}$ are faulty events.

We have implemented our encoding in Java. We used the well designed API of the SAT solver Sat4j [Le Berre & Parrain 2010]. More efficient solvers could have been chosen but we took this one because it is compatible with our clauses generator written also in Java and only a limited speed up could be awaited from C++ solvers (a speed up of 4, i.e., reduction of 75% of the runtime is often observed).

The total number of propositional variables, denoted by $|V|$, in the generated formula Φ_n^T after n steps is given by:

$$|V| = n \times (2|A| + 3\sum_{i=1}^{|\Sigma_o|} |e_{i_o}| + \sum_{i=1}^{|\Sigma_f|} |e_{i_o}| + 2\sum_{i=1}^{|\Sigma_u|} |e_{i_o}|) \tag{3.2.5}$$

where $|A|$ is the total number of state variables, $|\Sigma_o|$ the total number of observable events, $|\Sigma_f|$ the total number of faults (one in our case, from assumption 3), $|\Sigma_u|$ the total number of unobservable correct events, $|e_{i_o}|$ the total number of occurrences of the event e_i which appears respectively in Σ_o, Σ_f and Σ_u.

We take a distributed LTS model T and a steps number n as input and then we build both the succinct representation of T and its diagnosability checking formulation up to step n by generating all variables for n steps from the beginning. This has two opposite consequences. From one side, it provides a verbose system description even without unfolding it. From another side, one can exploit the fact that now new variables will be introduced as the system description is complete and only proceed to unfolding along the time, as this allows the ordering of these variables by their time step. Actually, the DIMACS format represents variables by integers and, as we can see in equation 3.2.5, there is a fixed number of variables generated for each time step. And, concerning constraints generation, one would do it only once for one time step, then get next steps constraints by just shifting the integers.

The results are in Table 3.1, where the columns show the considered (sub)system and the considered fault (in 3 cases), the steps number n, the satisfiability result, the numbers of variables and clauses and the runtime in milliseconds.

These results mean that f_2 is not diagnosable in $C2$ alone while it becomes diagnosable when synchronizing $C2$ and $C3$. For this last result, we have increased the steps number until reaching $2^{2|A|}$ (i.e., 2^{10} for the subsystem $\{C2, C3\}$), which is the theoretical upper bound of the Twin Plant states represented in the logical formula. As, in general, it is not always possible to reach this bound in practice, we propose in section 3.3 to use incremental SAT to improve the management of the steps number increasing. While f_1 is not diagnosable even after synchronizing all three components together. Actually, the numbers of variables and clauses generated in these tests are small in comparison to what SAT solvers can handle (up to hundred thousands propositional variables and millions of clauses). The first three cases of the table are mentioned as a proof of concept. However, to test the tool on larger systems and because of the absence of benchmark in the literature, we have multiplied these components as shown in the last rows of the table.

System	Fault	|Steps|	SAT?	|Variables|	|Clauses|	runtime(ms)
$C2$	f_2	4	No	106	628	27
$C2$	f_2	5	Yes	131	783	15
$C2, C3$	f_2	5	No	205	1017	5
$C2, C3$	f_2	32	No	1258	6444	32
$C2, C3$	f_2	64	No	2506	12876	43
$C2, C3$	f_2	128	No	5002	25740	139
$C2, C3$	f_2	256	No	9994	51468	118
$C2, C3$	f_2	512	No	19978	102924	166
$C2, C3$	f_2	1024	No	39946	205836	7791
$C1, C2, C3$	f_1	8	No	576	3546	91
$C1, C2, C3$	f_1	9	Yes	646	3987	110
$C1, 10 \times C2, 10 \times C3$	f_1	9	Yes	3851	22,879	402
$C1, 20 \times C2, 20 \times C3$	f_1	9	Yes	7101	42,059	515
$C1, 50 \times C2, 50 \times C3$	f_1	9	Yes	16,851	99,599	4417
$C1, 100 \times C2, 100 \times C3$	f_1	9	Yes	33,101	195,499	15,412

Table 3.1 Diagnosability Testing Results on the example of Figure 3.1.

We should mention here that our representation of the communication events can reduce the number of clauses over the direct representation mentioned before and this reduction is proportional to the number of occurrences of the communication events.

3.2.4 Discussion

The results in Table 3.1 show the efficiency of this approach in proving the non-diagnosability. The number of states in the last line is very large and it is obvious that proving the non-diagnosability of such a system using the Twin Plant following the approach in [Pencolé 2004, Schumann & Pencolé 2007, Ye & Dague 2010] is not practical. Thus, the fault f_1 is not diagnosable in the whole system, which means that the critical path will always exist. Even when we try to disconfirm it incrementally, by exploiting the communications with the neighboring components, we will be forced to continue the synchronization until covering all the system and, for each step, we have to build the twining structures and search for a critical path into them, then eliminate their non-ambiguous parts, a task that will be always difficult in such huge systems. However, by using SAT solvers, the system will be unfolded only when the previous fold does not contain the pair of infinite faulty and correct trajectories that coincide on their observations, i.e., when a witness of non-diagnosability is not found. Another difference is that, even in the unfolded part of the system, a SAT solver will not construct the Twin Plant explicitly, it will rather concentrate on constructing only the

critical pair parts, moreover the non-participating parts can help in pruning further failures through the new constraints learned during the search. This difference makes interesting the exploitation of the power of modern SAT solvers that allows us to encode the whole system and discover earlier such cases.

Here we have some global communications (which are shared by all components) and, where they participate in both trajectories in the same order, they cannot serve to distinguish them. However, having different communications in the critical pair leaves the door open to disambiguate the pair by exploiting neighbors information coming through communications. Actually, this is an idea present in the traditional approaches which can also return a diagnosable subsystem that can be very valuable during a diagnosis process.

Unfortunately, the on-the-fly construction of the Twin Plant in our SAT-based approach and the fact that only the first critical pair found is returned by the SAT solver make the incremental management of the components and returning a diagnosable subsystem more difficult. In fact it requires updating with each added component the constraints (3.2.2) and (3.2.3) to consider the new embedded communication events. This could imply restarting the search from scratch in some cases. Another similar limitation for the SAT-based approach could be the length of the critical path. Despite the fact that this approach allows diagnosability testing in very large systems, it cannot dynamically increase the steps number to ensure testing all reachable states while searching for the cycles witnessing non-diagnosability (which could be very far from the search starting point or very long). The worst case for these limitations is when considering diagnosable systems. Thus, from a search point of view, it requires exploring the whole possible search space to prove the non-existence of the indented witness of non-diagnosability.

In order to relax or tame these potential limitations, we propose in the next section an approach to ensure the knowledge about the system known during one call of the solver to be recycled in the later calls. To this end we exploit the incremental search mode in SAT solvers explained in subsection 2.4.1.

3.3 Diagnosability Checking using Incremental SAT

We adapt satisfiability algorithms for checking diagnosability of both centralized (subsection 2.4.3) and distributed (subsection 3.2.2) DES in order to incrementally process the maximum length of paths with cycles searched for witnessing non-diagnosability and we provide experimental results.

3.3.1 Diagnosability as Incremental Satisfiability

We distinguish two cases while testing diagnosability using SAT solvers in order to verify the satisfiability of the logical formula Φ_n^T for a given n as described in [Rintanen & Grastien 2007]. The first case is when we find a model for Φ_n^T, which *definitely* indicates the non-diagnosability of the studied fault. The second case is when we do not find such a model, which indicates that the non-diagnosability of the studied fault cannot be proved until n steps. In other words, after testing all the possible first n steps, we did not find a pair of trajectories of length at most n such that each exactly one of them contains the fault, they are equivalent in terms of observations and each one contains a cycle making them potentially infinite. However, in order to guarantee that the fault is actually diagnosable we have to unfold the system to cover all reachable states. This is impractical in large systems as the theoretical upper bound for n is $2^{2|A|}$. Thus our intended pair might appear for a greater value of n than the one given as input. Actually the artificial benchmark that we will present in subsection 3.3.2 contains such a case, where the very last possible event is required to show the non-diagnosability of the studied fault. However, this is also possible in real systems where such large bounds can be a reason behind keeping the system not diagnosable, the diagnosability property becoming irrelevant in practice if it requires so many time steps to be verified. Testing such cases includes recalling the SAT solver after having increased n and *rebuilt the logical formula* Φ_n^T. This is repeated several times before getting a definitive answer about the diagnosability problem.

Instead of that, we propose altering the formula Φ_n^T in order to be testable in an *incremental* SAT mode through multiple calls to a CDCL solver. As we mentioned earlier, the incremental mode in a SAT solver consists in testing a formula under a set of assumptions after embedding them in the clauses to be controlled in the formula. Thus, these clauses can be activated or dis-activated corresponding respectively to imposing or relaxing an assumption. In fact using CDCL solvers in a specialized, incremental, mode is relatively new [Nadel & Ryvchin 2012] but already widely used in many applications. In this operation mode, the solver can be called many times with different formulas. However, solvers are designed to work with similar formulas, where clauses are removed and added from calls to calls. Learned clauses can be kept as soon as the solver can ensure that clauses used to derive them are not removed. This is generally done by adding specialized variables, called *assumptions*, to each clause that can be removed, as a disjunction of the negated variables. By assuming the variable to be $True$, the clause is activated and by assuming the variable to be $False$, the clause is trivially satisfied and no longer used by the solver. What is interesting for our purpose is that the CDCL solvers can save clauses learned during the previous calls and test multiple assumptions in each new call. This means that after n steps we hope that

the solver will have learned some constraints about the behavior of the system. Although we are interested in testing the diagnosability property on a defined system, this framework is general as the system behavior which can be learned by the solver from the previous calls is independent from this property.

Modeling in incremental SAT

In order to extend to the incremental testing mode the clauses representation given in subsections 2.4.3 and 3.2.2, we propose to divide the formula Φ_n^T in two parts. The first part $\mathscr{T}_n$ describes the first n steps of the behavior of both trajectories, synchronized on the observations. It corresponds to (2.4.9) and is expressed by the conjunction of the formula $\mathscr{I}_0$ representing the initial state and the formulas $\mathscr{T}(t,t+1)$, $0 \leq t \leq n-1$, representing the $(t+1)$th step. The second part $\mathscr{D}_n$ describes the diagnosability property status at step n, i.e., firstly the occurrence of the fault f in the n previous steps of the faulty trajectory (given by the formula $\mathscr{F}_n$ corresponding to (2.4.10)) and secondly the detection of an observable cycle at step n (given by the formula $\mathscr{C}_n$ corresponding to (3.2.4)). So we obtain, for $n \geq 1$:

$$\Phi_n^T = \mathscr{T}_n \wedge \mathscr{D}_n$$

$$\mathscr{T}_n = \mathscr{I}_0 \wedge \bigwedge_{t=0}^{n-1} \mathscr{T}(t,t+1) \qquad \mathscr{D}_n = \mathscr{F}_n \wedge \mathscr{C}_n$$

$$\mathscr{F}_n = \bigvee_{t=0}^{n-1} \bigvee_{o \in \delta(f)} f_o^t$$

$$\mathscr{C}_n = \bigvee_{m=0}^{n-1} \Big(\bigwedge_{a \in A} ((a^n \leftrightarrow a^m) \wedge (\hat{a}^n \leftrightarrow \hat{a}^m)) \Big) \wedge \Big(\bigvee_{t=m}^{n-1} \bigvee_{e \in \Sigma_o} e^t \Big)$$

Add now at each step j a control variable h_j allowing to disable (when its truth value is *False*) or activate (when its truth value is *True*) the formulas $\mathscr{F}_j$ and $\mathscr{C}_j$ and keep at step n all these controlled formulas for $1 \leq j \leq n$. We obtain the following $\overline{\Phi}_n^T$ formula, for $n \geq 1$:

$$\overline{\Phi}_n^T = \mathscr{T}_n \wedge \bigwedge_{j=1}^{n} \mathscr{D}'_j \qquad \mathscr{D}'_j = \mathscr{F}'_j \wedge \mathscr{C}'_j \quad 1 \leq j \leq n$$

$$\mathscr{F}'_j = \neg h_j \vee \mathscr{F}_j \qquad \mathscr{C}'_j = \neg h_j \vee \mathscr{C}_j \quad 1 \leq j \leq n$$

We have thus the equivalence, for all $n \geq 1$:

$$\Phi_n^T \equiv \overline{\Phi}_n^T[H_n]$$

where $H_n = \{\neg h_1, \ldots, \neg h_{n-1}, h_n\}$ is a setting of the control variables and $F[H_n]$ the application of the valuation given by this setting to the formula F.

This allows one, for all $n \geq 1$, to replace the SAT call on Φ_n^T by a SAT call on $\overline{\Phi}_n^T$ under the control variables setting H_n (indicated in a second argument of the call):

$$SAT(\Phi_n^T) = SAT(\overline{\Phi}_n^T, H_n)$$

The idea is now to consider the control variables h_j as assumptions and use incremental SAT calls $IncSAT_j$ under varying assumptions, for $1 \leq j \leq n$. For this, we use the following recurrence relationships for both formulas $\overline{\Phi}_j^T$ and assumptions sets H_j:

$$\begin{array}{ll} \overline{\Phi}_0^T = \mathscr{I}_0 & \overline{\Phi}_{j+1}^T = \overline{\Phi}_j^T \wedge \mathscr{T}(j, j+1) \wedge \mathscr{D}'_{j+1} \quad j \geq 0 \\ H_0 = \{h_0\} & H_{j+1} = H_j[\{\neg h_j, h_{j+1}\}] \quad j \geq 0 \end{array}$$

where the notation $H_j[\{ass_i\}]$ means updating in H_j assumptions h_i by their new settings ass_i, i.e., in the formula above, replacing the truth value of h_j, which was $True$, by $False$, and adding the new assumption h_{j+1} with truth value $True$. From these relationships, the unique call to SAT under given assumptions $SAT(\overline{\Phi}_n^T, H_n)$ can be replaced, starting with the set of clauses $\mathscr{I}_0$ and assumption h_0, by multiple calls, $0 \leq j \leq n-1$, to an incremental SAT under varying assumptions:

$$\begin{aligned} &IncSAT_{j+1}(NewClauses_{j+1}, NewAssumptions_{j+1}) \\ &\quad = IncSAT_{j+1}(\mathscr{T}(j, j+1) \wedge \mathscr{D}'_{j+1}, \{\neg h_j, h_{j+1}\}) \end{aligned} \tag{3.3.1}$$

If $IncSAT_j$ answers SAT, the search is stopped as non-diagnosability is proved; if it answers UNSAT, then $IncSAT_{j+1}$ is called.

Notice that we used a unique assumption h_j for controlling both $\mathscr{F}_j$ and $\mathscr{C}_j$ as non-diagnosability checking requires the presence of both a fault occurrence in the faulty trajectory and of a cycle. But the same framework allows the independent control of formulas by separate assumptions. For sake of simplicity, we also assumed we called $IncSAT$ at each time step, but this is not mandatory and indexes j for the successive calls can be decoupled from indexes t for time steps. We should also say that, even if $IncSAT$ allows us to reactivate an

already disabled clause, we are sure in our case to never use this function (when h_k has been set to *False*, it always remains so) and we can thus force the solver to do a hard simplification process that removes the forgotten clauses permanently. As a result of our adaptation, we will be able to manage incrementally the scaling up of the size of the tested system and of the distance from initial state and length of a cycle witnessing non-diagnosability.

3.3.2 Experimental Results

We show in this subsection a comparison between our adapted version of SAT-based diagnosability checking described in the previous subsection 3.3.1, that uses incremental SAT, and the previous versions, for centralized systems (subsection 2.4.3 following [Rintanen & Grastien 2007]) and for distributed systems (subsection 3.2.2). We have created a scalable example represented in Figure 3.2 which contains $2k+1$ components: one faulty component and two sets of k neighboring correct components. The faulty component has two separated paths, each one containing k different successive unobservable correct events c_i and ending with the same observable cycle of length 1, but only one of them contains the fault f. The centralized model will be limited to this faulty component alone and thus in this case the events c_i, $1 \leq i \leq 2k$, are just unobservable correct events as is u.

In the distributed model, these events c_i are communication events and the faulty component is considered with the other two sets of correct components, where each component in both sets shares one event c_i with the faulty component to ensure a number $2k$ of communications before arriving to the cycles that will witness the non-diagnosability of the fault. Each set of components will be synchronized with only one path, either the faulty path or the correct one. This allows us to study the effect of the cycle distance in both models.

The results are in Table 3.2 for the centralized model (for k = 18, 28, 38, 48, 58 and 98) and in Table 3.3 for the distributed model (for k = 3, 13, 23, 33, 43 and 63). The length of a pair of trajectories with cycles witnessing the non-diagnosability of f in each example is $k+2$ and we consider the satisfiability of the formula Φ^T_{k+2}, so the number of steps required for SAT to provide the answer *Yes* is: $|Steps] = k+2$. In order to obtain a fair comparison between *IncSAT*, which manages internally by handling assumptions the successive satisfiability checks of increasing formulas for $j = 1, \ldots, k+2$, and *SAT*, for which $k+2$ successive calls are made to the solver with respective formulas Φ^T_n for $n = 1, \ldots, k+2$, the sum of the $k+2$ runtimes of the SAT solver calls are considered in this case (last column in the tables).

Although these examples remain relatively simple and do not reflect any potential constraint that could be summarized by some learned clauses (e.g., no interfering events), we can already notice the difference in runtime in favor of our incremental version in the centralized

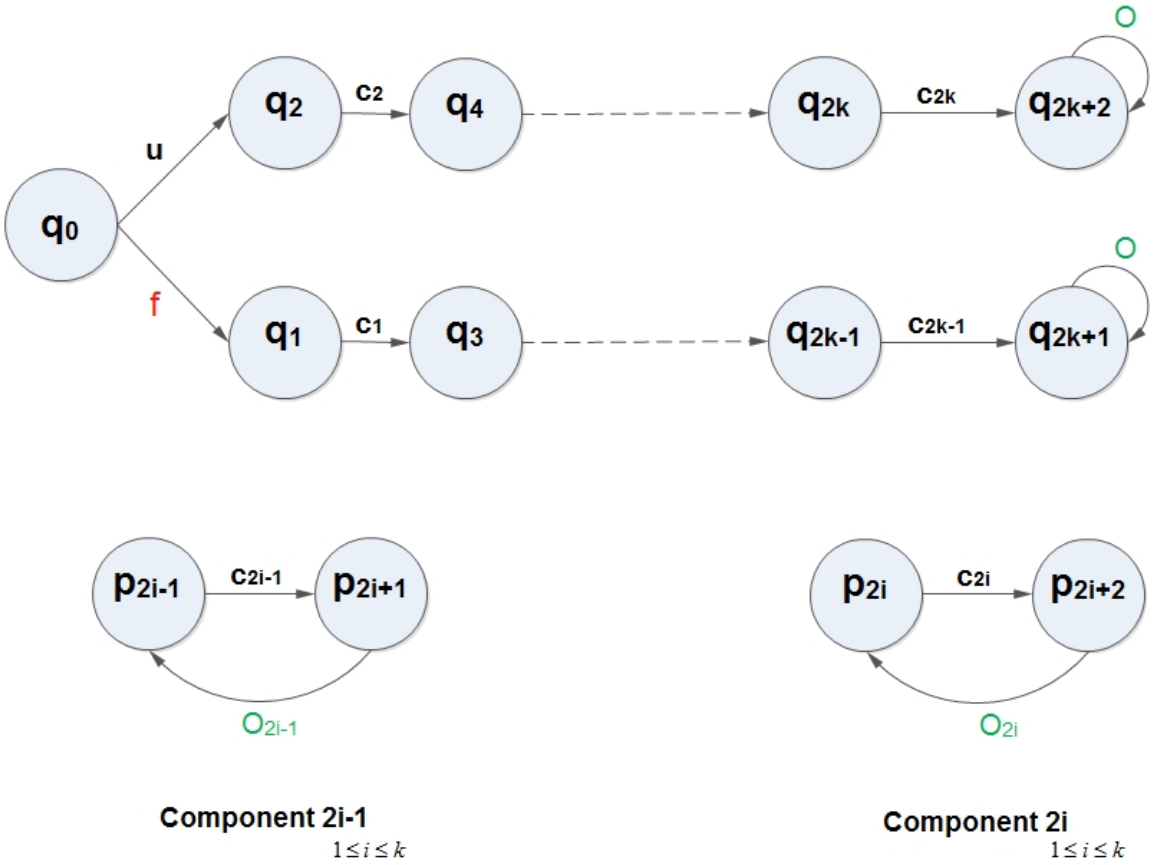

Fig. 3.2 One faulty component that communicates with two sets of k components. Each set communicates with one path (resp. faulty and correct) in the faulty component.

case and for the two largest values of k in the distributed case. This was expected because keeping the solver hot allows it to exploit its acquired knowledge for any potential need during the next calls. In other words, the fact that we generate from the beginning all time steps constraints that concern all variables imply many meaningless clauses that add a load on the solver in the version in [Rintanen & Grastien 2007]. However, this load is avoided in our incremental version because of the clauses learned by the CDCL SAT solver. More precisely, if the model, that represents the critical path, is found in k steps, the difference between the two approaches is the following. In the non-incremental version, the time spent for all calls on Φ_i^T such that $i < k$ will be lost and a new fresh formula about the system unfolded on k steps will be considered to search for the intended model. However, in the incremental version, some knowledge could be acquired from the calls on Φ_i^T such that $i < k$ and the formula that represents the system unfolded on k steps will be enriched by the

|Steps|	|Clauses|	Inc. SAT(s)	SAT(s)
20	42,614	1.5	1.3
30	131,714	10.3	13.1
40	303,736	49.3	77.8
50	576,466	106	223
60	970,156	320	699
100	4,334,018	9410	13,040

Table 3.2 Results on the faulty component of Figure 3.2.

|Steps|	|Comps|	|Clauses|	Inc. SAT(s)	SAT(s)
5	7	1,962	0.04	0.06
15	27	30,313	0.8	0.5
25	47	113,906	6.5	4.8
35	67	277,873	33.8	33.7
45	87	542,033	111	132
65	127	1,490,590	967	1090

Table 3.3 Results on the whole system of Figure 3.2.

compiled knowledge learned from previous calls, which could facilitate the mission for the solver to find the model.

3.4 Conclusion

In this chapter we have extended to DDES the state of the art work done on SAT-based diagnosability testing for centralized DES. Thus we have expressed diagnosability analysis of DDES as a satisfiability problem through introducing distributed succinct transition systems, then building a propositional formula whose satisfiability, checked by SAT solvers, witnesses the non-diagnosability of the studied fault. We allowed both observable and unobservable communication events to be handled simultaneously in our modeling. Our expression of these communication events avoids merging all their owner components and helps in reducing the number of clauses used to represent them; this reduction is proportional to the number of their occurrences. The SAT-based approaches are efficient in proving the non-diagnosability of a fault even in large systems, not testable by traditional Twin Plant approaches. However, they are not practical in proving diagnosability of the faults. In order to tame this weakness, we have proposed an adaptation of the logical formula in order to use incremental SAT calls that helps in managing the scaling up of the distance of the

searched fault and of the cycles (together with their lengths) required in witnessing non-diagnosability, and thus the size of the tested system. Thus we exploited the clauses learned about the system behavior in the previous calls. This approach is more practical and more efficient for complex systems than existing ones, as it avoids starting from scratch at each call.

Chapter 4

SAT-Based Predictability Analysis in Centralized and Distributed DES

In this chapter we study the verification of another important property related to fault analysis in model-based systems, which is the predictability of an event occurrence, in particular a faulty event occurrence. We recall the definition of this property and how it is dealt with in the literature. After that we present our encoding of this problem as a SAT problem in both centralized and distributed structures. Therefore, we use the same approach we used to handle the diagnosability problem, i.e., through passing by the succinct transition representation. We will show how the flexibility and the generality of SAT-based approach help in adapting the formalism to handle the studied property.

4.1 Motivation/Introduction

We explained in the previous chapter the importance of ensuring the diagnosability of a fault in order to get a precise diagnosis. A fault occurrence detection is usually followed by a repair plan. In other words, the fault has already occurred in such case and the operator of the system will fix the potential damages caused by the fault. Although that guaranteeing diagnosability of a system since its design stage can save potential high costs of adding new sensors during the diagnosis process, this might be insufficient in critical systems. Thus, in such systems, a fault occurrence may be very expensive and repairing such systems may not be possible, therefore such faulty events must be predicted to allow the system operator to take some required actions that avoid their occurrences. Clearly, the task of predicting a fault occurrence is harder than detecting it, simply because the latter can analyze the fault effects while the former has to analyze only its premises. Thus the ability to discover

the fault premises implies discovering its effects. Then predictability of a fault reveals its diagnosability.

Recently, the problem of event occurrence predictability was introduced in [Genc & Lafortune 2006b] to generalize the fault diagnosability problem. Thus it extends the observation explanation from a reactive mode in case of diagnosability analysis into a proactive one in the predictability case, through observations analysis only before the fault occurrence. More precisely, it considers the observation equivalence relation before the fault occurrence between two potential behaviors in the system: a first faulty finite trajectory from the initial state until the first occurrence of the fault and a second correct (without fault) infinite trajectory which might exist. Thus, if such two behaviors exist and are observation-equivalent before the fault occurrence, then the fault non-predictability is proven; however this fault could still be diagnosable.

Similarly to diagnosability case, the main difficulty in predictability checking is related to the states number explosion problem. In general, symbolic methods are used to cope with this problem that scale well with the size of the studied system, as we have shown in Chapters 2 and 3. However, for the best of our knowledge, this is not yet applied in the case of predictability analysis. Thus, since its introduction, different works tried to imitate the diagnosability checking approaches, i.e., Twin Plant based methods which suffer from the same problem of scalability and from the fact that they are obliged to pass by the automata synchronization process to build the appropriate data structure and then search a witness for non-predictability. In addition they do not exploit well the differences between predictability and diagnosability cases.

For these reasons, we propose in this chapter an encoding to handle the predictability problem in centralized and distributed structures using a SAT-based approach. To this end, we exploit the flexibility of the existing diagnosability encoding in SAT presented in Chapters 2 and 3 and we adapt it to encode predictability problem as a satisfiability one in both centralized and distributed structures.

4.2 Predictability Problem

The notion of predictability has been used in many works like in [Buss *et al.* 1990, Cao 1989, Fadel & Holloway 1999, Jiang & Kumar 2004]; however the one we are interested in is related to diagnosis in discrete event systems and inspired from the studies of diagnosability problem. In fact, works on the predictability property for DES are fewer and more recent than those about diagnosability. The event occurrence predictability property considers the problem of finding a prefix trajectory of the event occurrence such that, whenever we observe

its projection on observable events, we are sure that the studied event will occur in any future continuation of this observable sequence. Notice that unlike diagnosability which reasons about the potential histories of some observable run in the system, predictability argues about the potential futures of an observable run and thus considering either an observable or non-observable event occurrence can be interesting here while it was meaningless to analyze the diagnosability of an observable event. From another side, the event occurrence whose predictability is analyzed can be faulty or not, which applied also to diagnosability analysis. However, for the sake of consistency with the context of this thesis, we will be interested in fault occurrence predictability, in particular first occurrence predictability (as it implies predictability of possible ulterior occurrences), and so our notations and vocabulary will reflect this hereafter.

The formal definition of predictability of a fault f in a centralized system modeled by an LTS or SLTS T was proposed by [Genc & Lafortune 2006b] as follows.

Definition 11. (Predictability) A fault f is predictable in a system T iff

$$\exists k \in \mathbb{N}, \forall s^f \in L(T), \exists \eta \in \overline{s^f}, \forall p \in L(T), \forall p' \in L(T)/p$$
$$(P(p) = P(\eta) \wedge f \notin p \wedge |p'| \geq k \Rightarrow f \in p')$$

The above definition, where $\bar{t}$ denotes the set of strict prefixes of t, states that a fault f is predictable iff for any trajectory s^f ending with f, there exists at least one strict prefix of s^f, denoted by η, such that for every correct (i.e., not containing f) trajectory p with the same observations as η, all the long enough (depending only on f) continuations of p should contain f. It is obviously sufficient to study predictability of first occurrences of f, i.e., when $s^f \in (\Sigma \setminus \{f\})^* f$, in which case η does not contain f.

In an equivalent meaning, the non-predictability of f is equivalent to the existence of a *finite* faulty sequence that ends with a first occurrence of f and of an *infinite* (i.e., ending by a cycle) correct sequence that is synchronized with the faulty sequence on observable events before the occurrence of f. It is thus clear that non-diagnosability of f implies non-predictability of f, which means that predictability is stronger than diagnosability (if f is predictable, then f is diagnosable).

4.2.1 Traditional Predictability Checking in Centralized and Distributed DES

In order to verify the predictability property introduced in [Genc & Lafortune 2006b] in a centralized system, the authors proposed a deterministic diagnoser approach, just like

the one used in [Sampath *et al.* 1995] which we recalled in section 2.3.2. The diagnoser has an exponential size in terms of the number of system states. This approach consists in determining the set $\mathscr{S}_f$ of boundary N-certain states before the first fault occurrence in this diagnoser. Thus, for each member q in this set, q possesses an immediate successor which is not an N-certain state, i.e., a F-certain or F-uncertain state. Then one can verify that every reachable cycle in the diagnoser from such q is actually a F-certain cycle. In fact, this verification provides a necessary and sufficient condition to ensure the predictability of a studied event occurrence in the system. Verifying the continuations of only original states in this boundary set is sufficient, where a state in $\mathscr{S}_f$ is said not original if it can be reached by another state in the same set. Actually an important amount of verification cases can be abstracted using this technique. Another benefit of the diagnoser is that whenever it is constructed, it can be used for predicting on-line a predictable event occurrence through inspecting the original states in $\mathscr{S}_f$ that reveal with certainty its occurrence in all the possible futures.

Later, the same authors proposed in [Genc & Lafortune 2009] a polynomial method, that checks predictability directly on a verifier structure which is a Twin Plant like structure. They search for a set of boundary states between normal and not normal states in this verifier, which is similar to the set $\mathscr{S}_f$ used in their diagnoser approach. Remember that transitions in the verifier are of the form $(x,x') \xrightarrow{\sigma} (y,y')$, where x,x',y,y' are states in the system, σ is an event and the transitions $(x) \xrightarrow{\sigma} (y)$, $(x') \xrightarrow{\sigma} (y')$ are both defined in the system transition relation if σ is observable and at least one of them is defined if σ is unobservable. However as the set $\mathscr{S}_f$ is to be searched on a structure that represents pairs of behaviors, it is calculated in two steps. Firstly by determining the change from normal to faulty or ambiguous states in the verifier, let it be the set $\mathscr{S}'_f$. Then this set is saturated by adding any potential states that correspond to sequences which will not have the studied event occurrence in any of their future continuations but however do share their prefix with other states that will have at least one occurrence of the studied event in one of their future continuations. Actually the existence of such added states, i.e., $\mathscr{S}'_f \neq \mathscr{S}_f$, plus the liveness of the language $L(T)$ defined by the system, witness the non-predictability of the studied event. The authors stated the following necessary and sufficient condition for an event to be predictable: every reachable cycle in the verifier from states in this set $\mathscr{S}_f$ is a faulty cycle. In other words, this method checks a pair of trajectories violating predictability, i.e., such that only one of them contains the fault, this faulty one can be finite while the second one is infinite and both coincide on their observable events before the fault occurrence. The problem in this method is that the intended pair could lie in two different paths of the Twin Plant structure, which is not suitable for the distributed case. The work in [Ye *et al.* 2013] proposed a Twin

Plant like structure and handled the predictability problem in centralized systems in such a way it facilitates passing to the distributed DES case. For this case, it imitates works that dealt with distributed diagnosability, i.e., through use of local diagnosers that will be synchronized to get the pairs of behaviors representation in order to differentiate, if possible, the conformity between observations in the two behaviors by exploiting the information of the communication events. To this end, the original predictability information is gathered from the faulty component, then its global consistency in the whole system is checked incrementally to decide fault predictability. Thus the construction of the global structure is avoided to reduce the search space. However, in the worst case, this approach has the same search space as constructing a global Twin Plant. This was solved recently by the same authors who presented a better algorithm in [Ye *et al.* 2015] that tries to exploit the differences between predictability and diagnosability checking in order to build a more scalable structure that prunes some unnecessary tests. We will use similar type of information to prune the search space in our SAT encoding of the problem, however we will delegate to the solver the labor of handling such pruning.

Similarly to the diagnosability problem, all the above approaches have firstly to build the data structure that represents the relevant pairs of behaviors and then search inside this structure for those verifying given properties. Actually, the data structure construction can be avoided by using the succinct system model and by encoding the problem as a SAT problem, just like what we presented while checking diagnosability. Although we do not know exactly how the SAT solver will find the counter-example witnessing the violation of the property, we are sure that, even if it does some failed tries before finding the solution, it will exploit these failures to recover its tries on the road to the intended solution.

We adapt our diagnosability analysis framework to define SAT-based predictability analysis for both centralized and distributed systems and provide experimental results.

4.2.2 SLTS Predictability as Satisfiability

Unlike diagnosability, predictability checking process has two different phases, before and after the fault occurrence. The synchronization on observable events between the two trajectories is required only up to this fault occurrence and, after it, only the correct trajectory has to be extended and searched for the presence of a cycle in it.

In order to represent the occurrence of the fault f and differently from the original diagnosability encoding in [Rintanen & Grastien 2007], which does not exploit any relation between the fault occurrences at the different time steps, we add the variables f^t to the set of variables defined in the diagnosability problem encoding. The truth value f^t is $True$ iff f has occurred before the time step t. This will help to propagate the fault information

automatically and guide the solver to search for this specific information about the fault occurrence which is essential to decide the predictability test.

Let $T = \langle A, \Sigma_o, \Sigma_u, \Sigma_f, \delta, s_0 \rangle$ be an SLTS, the propositional variables required for the encoding are:

- a^t and $\hat{a}^t$ for all $a \in A$ and $0 \leq t \leq n$,
- e_o^t for all $e \in \Sigma_o \cup \Sigma_u \cup \Sigma_f$, $o \in \delta(e)$ and $0 \leq t \leq n-1$,
- $\hat{e}_o^t$ for all $e \in \Sigma_o \cup \Sigma_u$, $o \in \delta(e)$ and $0 \leq t \leq n-1$,
- e^t for all $e \in \Sigma_o$ and $0 \leq t \leq n-1$,
- f^t for all $0 \leq t \leq n$.

We provide here the full set of constraints required to encode the predictability problem as a SAT problem. Although most of them are copied from the diagnosability SAT formula, we recall them here for the sake of clarity and completeness of the encoding. The following formulas express the constraints that must be applied at each t or between t and $t+1$.

1. The event occurrence e_o^t must be possible in the current state:

$$e_o^t \rightarrow \phi^t \quad \text{for } o = \langle \phi, c \rangle \in \delta(e) \tag{4.2.1}$$

   and its effects must hold at the next time step:

$$e_o^t \rightarrow \bigwedge_{l \in c} l^{t+1} \quad \text{for } o = \langle \phi, c \rangle \in \delta(e) \tag{4.2.2}$$

   We have the same formulas with $\hat{e}_o^t$.

2. The present value (*True* or *False*) of a state variable changes to a new value (*False* or *True*, respectively) only if there is a reason for this change, i.e., because of an event that has the new value in its effects (so, change without reason is prohibited). Here is the change from *True* to *False* (the change from *False* to *True* is defined similarly by interchanging a and $\neg a$):

$$(a^t \wedge \neg a^{t+1}) \rightarrow (e^t_{i_1 o_{j_1}} \vee \cdots \vee e^t_{i_k o_{j_k}}) \tag{4.2.3}$$

   where the $o_{j_l} = \langle \phi_{j_l}, c_{j_l} \rangle \in \delta(e_{i_l})$ are all the occurrences of events e_{i_l} with $\neg a \in c_{j_i}$.
   We have the same formulas with $\hat{a}^t$ and $\hat{e}^t_{i_l o_{j_l}}$.

3. At most one occurrence of a given event can occur at a time and the occurrences of two different events cannot be simultaneous if they interfere (i.e., if they have two contradicting effects or if the precondition of one contradicts the effect of the other):

$$\neg(e_o^t \wedge e_{o'}^t) \quad \forall e \in \Sigma, \forall \{o, o'\} \subseteq \delta(e), o \neq o' \tag{4.2.4}$$

$$\neg(e_o^t \wedge e_{o'}'^t) \quad \forall \{e, e'\} \subseteq \Sigma, e \neq e', \forall o \in \delta(e), \forall o' \in \delta(e'), o \text{ and } o' \text{ interfere} \tag{4.2.5}$$

We have the same formulas with $\hat{e}_o^t$.

4. The information about f occurrence is propagated by expressing that f has occurred before $t+1$ ($t \leq n-1$) iff it has occurred either before t or between t and $t+1$.

$$f^{t+1} \leftrightarrow f^t \vee \bigvee_{o \in \delta(f)} f_o^t \tag{4.2.6}$$

5. The synchronization of observable events between the two sequences holds only up to the fault occurrence:

$$\begin{aligned} f^t \vee (\bigvee_{o \in \delta(e)} e_o^t \leftrightarrow e^t) \quad \forall e \in \Sigma_o \\ f^t \vee (\bigvee_{o \in \delta(e)} \hat{e}_o^t \leftrightarrow e^t) \quad \forall e \in \Sigma_o \end{aligned} \tag{4.2.7}$$

6. The formula requiring that at every time point at least one event takes place in either one or the other sequence, remains valid up to the fault occurrence; after it, we require that at least one event takes place in the correct sequence:

$$\begin{aligned} f^t \vee \bigvee_{e \in \Sigma_o} e^t \quad \vee \bigvee_{e \in \Sigma_u \cup \Sigma_f, o \in \delta(e)} e_o^t \quad \vee \bigvee_{e \in \Sigma_u, o \in \delta(e)} \hat{e}_o^t \\ \neg f^t \vee \bigvee_{e \in \Sigma_o \cup \Sigma_u, o \in \delta(e)} \hat{e}_o^t \end{aligned} \tag{4.2.8}$$

The conjunction of the above formulas for a given t is denoted by $\mathscr{S}(t, t+1)$.

A formula for the initial state s_0 should take into account the initialization of the fault reference variables:

$$\mathscr{I}_0 = \neg f^0 \wedge \bigwedge_{a \in A, s_0(a)=1} (a^0 \wedge \hat{a}^0) \wedge \bigwedge_{a \in A, s_0(a)=0} (\neg a^0 \wedge \neg \hat{a}^0) \tag{4.2.9}$$

Finally, the formula to encode the non predictability property is obtained as Ψ_n^T, where the presence of a cycle at step n is required only in the correct sequence:

$$\Psi_n^T = \mathscr{I}_0 \wedge \mathscr{S}(0,1) \wedge \cdots \wedge \mathscr{S}(n-1,n) \wedge f^n \\ \wedge \bigvee_{m=0}^{n-1} \left(\bigwedge_{a \in A} (\hat{a}^n \leftrightarrow \hat{a}^m) \right)$$

It follows that an SLTS T is not predictable iff $\exists n \geq 1, \Psi_n^T$ is satisfiable, which is also equivalent to $\Psi_{2^{2|A|}}^T$ being satisfiable (proof analog to that for diagnosability in [Rintanen & Grastien 2007]).

4.2.3 DSLTS Predictability as Satisfiability

Let T be now a DSLTS. The extension of predictability analysis to distributed systems adapts what we presented for diagnosability analysis, with just a few changes concerning the synchronization of observable communication events and the occurrence of at least one event at each step. As the synchronization of observable events holds only before the fault occurrence, we will decouple it from the synchronization of communication events. So, the only change concerning the variables is that we use now one reference variable for each sequence for observable communication events, as for unobservable ones, i.e., we have:

- e^t, $\hat{e}^t$ for all $e \in \Sigma_c$ and $0 \leq t \leq n-1$.

Formulas in $\mathscr{S}(t,t+1)$ are extended as those in $\mathscr{T}(t,t+1)$ were extended, except the following.

1. The synchronization of the occurrences of any communication event e in all its owner components in $S(e)$ is expressed in each sequence and in the same way for both observable and unobservable events:

$$\bigvee_{o_i \in \delta_i(e)} e^t_{o_i} \leftrightarrow e^t \text{ and } \bigvee_{o_i \in \delta_i(e)} \hat{e}^t_{o_i} \leftrightarrow \hat{e}^t \ \forall e \in \Sigma_c \ \forall i \in S(e)$$

   while the synchronization of the occurrences of any observable event in the two sequences before the fault occurrence, expressed in the centralized case by formulas (4.2.7), is extended to any observable communication event:

$$f^t \ \vee \ (e^t \leftrightarrow \hat{e}^t) \quad \forall e \in \Sigma_{co}$$

2. The clauses (4.2.8) are extended to take into account communication events:

$$f^t \vee \bigvee_{e\in\Sigma_o\cup\Sigma_c} e^t \vee \bigvee_{e\in\Sigma_{cu}} \hat{e}^t \vee \bigvee_{e\in\Sigma_u\cup\Sigma_f, o\in\delta(e)} e^t_o \vee \bigvee_{e\in\Sigma_u, o\in\delta(e)} \hat{e}^t_o$$
$$\neg f^t \vee \bigvee_{e\in\Sigma_c} \hat{e}^t \vee \bigvee_{e\in\Sigma_o\cup\Sigma_u, o\in\delta(e)} \hat{e}^t_o$$

We have thus the result that a DSLTS T is not predictable iff $\exists n \geq 1, \Psi^T_n$ is satisfiable, which is also equivalent to $\Psi^T_{2^{2\Sigma_{i=1}^k |A_i|}}$ being satisfiable (proof analog to that for diagnosability).

4.2.4 Experimental Results

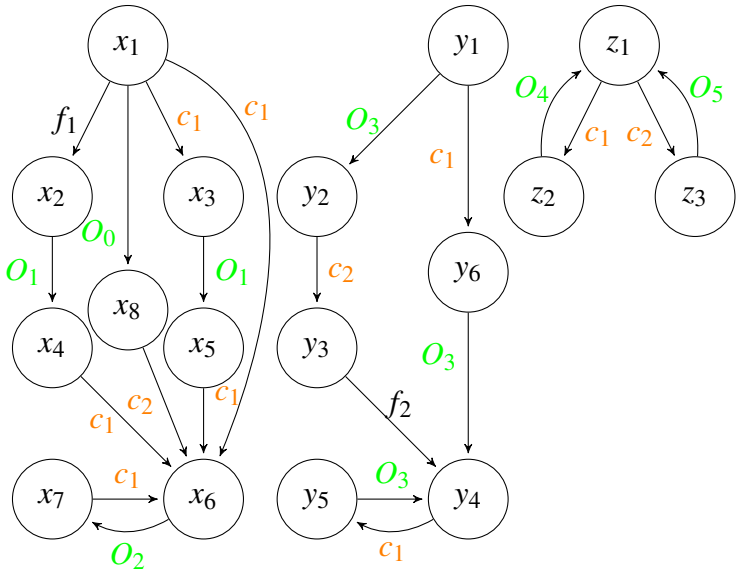

Fig. 4.1 Modified Distributed DES with three components, that share only the events $\{c1, c2\}$

We used the system in Figure 4.1, slightly modified from the one in Figure 3.1 that we used for diagnosability tests. The modifications are done to get more cases that help to study the predictability of the faulty events f_1 and f_2 in centralized and distributed structures. We repeated the diagnosability tests on these events in the modified system as shown in Table 4.1. The predictability results are given in Table 4.2. It is found that f_2 is not predictable in $C2$ alone, which was expected as it is not diagnosable in $C2$.

However it became diagnosable in the system composed of $C1$ and $C2$ and we find that it is actually even predictable in this system, by obtaining the UNSAT answer up to the theoretical upper bound of 4096 steps. On the contrary, although we saw it became also

diagnosable in the system composed of $C2$ and $C3$, we find that it remains not predictable in this system.

And here again, we extend this test to bigger systems by duplicating component $C3$. This shows the efficiency of the method (less than one second for 101 components). Notice that here the steps number remains unchanged as occurrences of non-interfering events are processed simultaneously in the same step, thanks to the succinct encoding in this representation. Concerning the fault f_1, it is found not predictable in the whole system made up of the three components, which was expected as it has been shown not diagnosable in this system.

We verify, as for diagnosability, that proving non-predictability is in general much easier than proving predictability. We verify also that, in case of non-diagnosability (and thus of non-predictability), proving non-predictability is in general easier than proving non-diagnosability (because there are more models of non-predictability than of non-diagnosability, as the second ones are included in the first ones). In the same way, in case of predictability (and thus of diagnosability), proving diagnosability is in general easier than proving predictability.

4.3 Predictability Checking Encoding in Incremental SAT

Similarly to what we did for diagnosability, we extend the clauses representation given in subsections 4.2.2 and 4.2.3 to an incremental mode of operation. We propose to divide the formula Ψ_n^T in two parts. The first part $\mathscr{S}_n$ describes the first n steps, expresses the behavior of the faulty and correct trajectories, synchronized on the observations up to the occurrence of the fault in the faulty trajectory. This is represented by the conjunction of the initial state and the formulas $\mathscr{S}(t,t+1)$, $0 \leq t \leq n-1$, representing the $(t+1)$th step. The second part $\mathscr{P}_n$ describes the predictability property at step n, i.e., the occurrence of a fault in the n previous steps of the faulty trajectory (given by f^n) and the detection of a cycle at step n (given by the formula $\mathscr{C}_n$). So we obtain, for $n \geq 1$:

$$\Psi_n^T = \mathscr{S}_n \wedge \mathscr{P}_n$$

$$\mathscr{S}_n = \mathscr{I}_0 \wedge \bigwedge_{t=0}^{n-1} \mathscr{S}(t,t+1) \qquad \mathscr{P}_n = f^n \wedge \mathscr{C}_n$$

$$\mathscr{C}_n = \bigvee_{m=0}^{n-1} \left(\bigwedge_{a \in A} (\hat{a}^n \leftrightarrow \hat{a}^m) \right)$$

Add now at each step j a control variable h_j allowing to disable (when its truth value is $False$) or activate (when its truth value is $True$) the formulas f^j and $\mathscr{C}_j$ and keep at step n all

System	Fault	|Steps|	SAT?	|Variables|	|Clauses|	Time(ms)
$C2$	f_2	4	No	112	561	6
$C2$	f_2	5	No	138	699	11
$C2$	f_2	6	Yes	164	837	15
$C1, C2$	f_2	6	No	356	356	25
$C1, C2$	f_2	32	No	1838	12751	94
$C1, C2$	f_2	64	No	3662	25487	225
$C1, C2$	f_2	128	No	7310	50959	112
$C1, C2$	f_2	256	No	14606	101903	180
$C1, C2$	f_2	512	No	29198	203791	1855
$C1, C2$	f_2	1024	No	58382	407567	784
$C1, C2$	f_2	4096	No	233486	1630223	23453
$C2, C3$	f_2	6	No	252	1237	15
$C2, C3$	f_2	32	No	1292	6541	46
$C2, C3$	f_2	64	No	2572	13069	71
$C2, C3$	f_2	128	No	5132	26125	61
$C2, C3$	f_2	256	No	10252	52237	216
$C2, C3$	f_2	512	No	20492	104461	143
$C2, C3$	f_2	1024	No	40972	208909	381
$C1, C2, C3$	f_1	8	No	586	3723	40
$C1, C2, C3$	f_1	9	Yes	657	4186	45
$C1, 10 \times C2, 10 \times C3$	f_1	9	Yes	3862	22907	342
$C1, 20 \times C2, 20 \times C3$	f_1	9	Yes	7112	42087	592
$C1, 50 \times C2, 50 \times C3$	f_1	9	Yes	16862	99627	3141
$C1, 100 \times C2, 100 \times C3$	f_1	9	Yes	33372	196723	26930

Table 4.1 Diagnosability Testing Results on the example of Figure 4.1.

these controled formulas for $1 \leq j \leq n$. We obtain the following $\overline{\Psi}_n^T$ formula, for $n \geq 1$:

$$\overline{\Psi}_n^T = \mathscr{S}_n \wedge \bigwedge_{j=1}^{n} \mathscr{P}'_j \qquad \mathscr{P}'_j = \mathscr{F}'_j \wedge \mathscr{C}'_j \quad 1 \leq j \leq n$$

$$\mathscr{F}'_j = \neg h_j \vee f^j \qquad \mathscr{C}'_j = \neg h_j \vee \mathscr{C}_j \quad 1 \leq j \leq n$$

We have thus the equivalence, for all $n \geq 1$:

$$\Psi_n^T \equiv \overline{\Psi}_n^T[H_n]$$

where $H_n = \{\neg h_1, \ldots, \neg h_{n-1}, h_n\}$ is a setting of the control variables and $F[H_n]$ the application of the valuation given by this setting to the formula F.

System	Fault	|Steps|	SAT?	|Variables|	|Clauses|	Time (ms)
$C2$	f_2	3	No	92	414	7
$C2$	f_2	4	Yes	120	549	12
$C1,C2$	f_2	1024	No	66574	404495	10109
$C1,C2$	f_2	4096	No	266254	1617935	91299
$C2,C3$	f_2	4	No	196	817	14
$C2,C3$	f_2	5	No	242	1018	21
$C2,C3$	f_2	6	Yes	288	1219	27
$C1,C2,C3$	f_1	3	No	267	1399	29
$C1,C2,C3$	f_1	4	Yes	350	1859	40
$C2,10\times C3$	f_2	6	Yes	1408	5219	24
$C2,20\times C3$	f_2	6	Yes	2528	9219	50
$C2,50\times C3$	f_2	6	Yes	5888	21219	125
$C2,100\times C3$	f_2	6	Yes	11488	41219	277

Table 4.2 Predictability Testing Results on the example of Figure 4.1.

This allows one, for all $n \geq 1$, to replace the SAT call on Ψ_n^T by a SAT call on $\overline{\Psi}_n^T$ under the control variables setting H_n (indicated in a second argument of the call):

$$SAT(\Psi_n^T) = SAT(\overline{\Psi}_n^T, H_n)$$

The idea is now to consider the control variables h_j as assumptions and use incremental SAT calls $IncSAT_j$ under varying assumptions, for $1 \leq j \leq n$. For this, we use the following recurrence relationships for both formulas $\overline{\Psi}_j^T$ and assumptions H_j:

$$\begin{aligned} &\overline{\Psi}_0^T = \mathscr{I}_0 \quad \overline{\Psi}_{j+1}^T = \overline{\Psi}_j^T \wedge \mathscr{S}(j, j+1) \wedge \mathscr{P}'_{j+1} \quad j \geq 0 \\ &H_0 = \{h_0\} \quad H_{j+1} = H_j[\{\neg h_j, h_{j+1}\}] \quad j \geq 0 \end{aligned}$$

where the notation $H_j[\{ass_i\}]$ means updating in H_j assumptions h_i by their new settings ass_i, i.e., in the formula above, replacing the truth value of h_j, which was $True$, by $False$, and adding the new assumption h_{j+1} with truth value $True$. From these relationships, the unique call to SAT under given assumptions $SAT(\overline{\Psi}_n^T, H_n)$ can be replaced, starting with the set of clauses $\mathscr{I}_0$, by n calls, $0 \leq j \leq n-1$, to an incremental SAT under varying assumptions:

$$\begin{aligned} &IncSAT_{j+1}(NewClauses_{j+1}, NewAssumptions_{j+1}) \\ &\qquad = IncSAT_{j+1}(\mathscr{S}(j, j+1) \wedge \mathscr{P}'_{j+1}, \{\neg h_j, h_{j+1}\}) \end{aligned} \tag{4.3.1}$$

If $IncSAT_j$ answers SAT, the search is stopped as non-predictability is proved; if it answers UNSAT, then $IncSAT_{j+1}$ is called.

Notice that we used a unique assumption h_j for controling both f^j and $\mathscr{C}_j$ as non-predictability checking requires the presence of both a fault occurrence in the faulty trajectory and a cycle in the correct trajectory. But the same framework allows the independent control of formulas by separate assumptions.

For sake of simplicity, we also assumed we called *IncSAT* at each step, but this is not mandatory and indexes j for the successive calls can be decoupled from indexes t for steps.

We have not implemented this encoding but we expect results similar to those we presented when using incremental SAT to test diagnosability in centralized and distributed systems.

4.4 Discussion and Conclusion

We have adapted the diagnosability test formalism presented in Chapter 3 to express predictability analysis as a SAT problem, both for centralized DES and for DDES. In each case, we have provided experimental results.

The results show that this approach is scalable to system sizes that are not possible to cover by the Twin Plant approaches used in the literature. Our approach is very efficient to decide the non-predictability in centralized and distributed systems and better than traditional approaches in this latter case. We used in our experiments an example that contains different cases of study like a fault that is predictable in subsystems but is not in other subsystems and another fault that is non-predictable even in the global distributed system. For this latter case our approach provides a witness of the non-predictability in a few steps, while even a component-incremental approach like the one in [Ye *et al.* 2015] has to build the global model that synchronizes all components to get this answer. Actually in our encoding of the DSLTS we encode components with disjoint sets of state variables which allow all local transitions in these components to be fired simultaneously. We showed how elegant is this approach and how it allows smooth derivation of the predictability test from the diagnosability test, in comparison to the non-direct algorithms adopted in the literature to pass from diagnosability to predictability. Another important point to notice in our predictability testing approach is the facility of handling both observable and non-observable communications in the same system; synchronizing an observable communication event can indeed give rise to different processes during the test, thus it is a fault dependent synchronization. However, this case is not considered in the literature, although it is harder than the case of checking diagnosability with such settings. We have also provided an adaptation to use incremental SAT mode to

increase the efficiency and scalability of the approach to test predictable faults and here also we showed how flexible is this formalism. In fact this flexibility in adding or removing constraints to encode different features can be used for testing other similar properties in the fault diagnosis domain and could be always simpler and more scalable than classical algorithms like the Twin Plant one. After encoding the problem in propositional logic the SAT solver will use its generic algorithms to answer such problems and then one can translate the answer to the appropriate semantics depending on the property under consideration.

From another point of view, our approach does not build structures which would be useful in an online test to serve as a predictor and cannot return a predictable subsystem automatically, which also can be very useful in operation mode. However we can impose a strategy to add and test components incrementally to the faulty one to check if they represent together a predictable subsystem. But to this end one may repeat the test from scratch after each addition of a component, which is not the case in the Twin Plant based methods that propagate only required information from one test to the next one resulting in an important reduction of the constructed parts of the global synchronization structure. Another way which could be better to overcome this limitation is to exploit the relationships between satisfiability and bounded or unbounded model checking methods to encode and analyze fault diagnosability and predictability. Research of invariants by full-proof methods like temporal induction should avoid unrolling up to a theoretical bound, as it is the case here when the system is diagnosable or predictable.

Chapter 5

SAT-based Encoding of Pattern Diagnosability in DES

This chapter addresses the problem of pattern diagnosability using SAT solvers. We introduce the problem, then we review how this problem is handled in the literature where the main difficulties are inherited again from the idea of building the whole data structure before searching for a counter-example in it, which can be avoided using a SAT based approach. We provide an encoding in SAT for this problem in the centralized case, without experimental results for the moment, showing that it is possible to adapt the formalism to handle such general patterns. We give some insights about the extension to the distributed case and to pattern predictability analysis.

5.1 Motivation/Introduction

We studied in the previous chapters the problems of faulty event diagnosability and predictability in centralized and distributed systems. Actually in some applications, one may be interested in detecting and isolating or even predicting the occurrence of a specific sequence of events. Thus the occurrence of only one or some of these events is considered normal w.r.t. the acceptable behavior of the system. However the occurrence of all these events together in a predefined order may form a threaten, a fault or just an important phenomenon in the system whom its operator (or designer) wants to be able to investigate or to ensure its detection.

This problem is referred to in the DES diagnosis community as pattern diagnosis. A pattern can represent a faulty situation, as multiple faults, cascading faults, intermittent faults or a specific behavior that it is important to detect and identify, this why it is generally

called surveillance pattern or supervision pattern. In full generality a pattern is a complete deterministic finite state machine (automaton) with the same events vocabulary as the system model. Actually assuming determinism and completeness is to facilitate its synchronization with the system in order to check its recognition. This is not a limitation as an automaton can always be determinized and completed; moreover our encoding in SAT remains unchanged if the pattern is not deterministic. It represents thus a (possibly infinite) set of predefined finite sequences of events. A trajectory of the system "contains" the pattern if it is a word recognized by the automaton, i.e., corresponding to a path in the pattern from its initial state to a final state. Obviously any extension of this trajectory has to contain the pattern, thus to be also recognized. So, the only requirement on the pattern is that its set of final states is stable. It means that a pattern may represent any rational events language that is closed by extension i.e., any extension of a word of the language belongs also to the language.

An example of application can be found when studying anomalies in sequences of interactions with a security system. Thus some successive interactions may represent intrusion attempts to get access to unauthorized situation. However if the system model is not designed to evolve (through changing its states and producing observations) such threats may not be detectable. A simple example of this is a multiple attempt to access an email account, thus a simple one or two attempts to enter a password can be considered a normal behavior while a greater number of them can be seen as an anomaly[1]. Let the system that represents such email account have two states $Q = \{q_0, q_1\}$ such that $q_0 = Closed$ and $q_1 = Open$; let it have three events $\Sigma = \{t, s, b\}$ where $t = Try$, $s = Success$ and $b = Browse$ and let $\Sigma_o = \{s, b\}$. Consider the pattern $P = \{ttt\}$ which represents three failed tries. Let the system modeled by transitions $\delta = \{(q_0, t, q_0), (q_0, s, q_1), (q_1, b, q_1)\}$, then we can see that the sequence defined by P cannot be detected. In order to handle correctly such situation, the system has to be modeled in a diagnosable way to distinguish the occurrence of such suspicious sequence.

Actually, the need for such diagnosis can be required in centralized systems as well as in distributed systems. This makes ensuring the diagnosability or predictability of patterns an essential issue since the design stage to get a precise diagnosis or prediction of the studied pattern. In fact, one would be interested in detecting or predicting the occurrence of a sequence of successive events with or without (other) interleaving events. From another point of view, detecting the occurrence of a sequence made up of only observable events is less challenging than detecting the occurrence of a sequence composed of both observable and non-observable events or even of only non-observable events. Actually, in the first case, it is sufficient to synchronize the pattern's observation sequence with the system's observable behavior.

[1] More details about intrusion detection can be found in the bibliography [Mé & Michel 2001]

The problem of pattern diagnosability [2] was first introduced by [Genc & Lafortune 2006a, Jéron *et al.* 2006] for centralized DES, then extended in [Ye *et al.* 2010, Ye & Dague 2012] to distributed DES. The main weakness of these approaches is that the bigger the size of the pattern, the more possible the risk of state explosion happen. This is due to the fact that building a data structure and then verifying in it the diagnosability is the backbone of such approaches. The problem may get worse when the pattern is distributed over a large number of components in a distributed system, i.e., contains events that belong to these components. In order to scale the pattern size, we propose to encode it in a succinct system representation and to simply synchronize it with the system's component(s) just like if it was a new added component. We also propose an improvement in order to exploit the precise information given by its structure, namely to limit the explicit synchronization to the only significant events of the pattern, i.e., those events that change a state of the pattern.

5.2 Definitions

Let the system T be an LTS given by the tuple $T = \langle Q, \Sigma, \delta, q^0 \rangle$ as defined in Chapter 2 (see Definition 1), except that $\Sigma_f = \emptyset$ as there is no more fault events. We define a pattern Ω over T as a complete deterministic LTS with a stable set F_Ω of final states, formally defined as follows.

Definition 12. A **Pattern** Ω is defined by a tuple $\Omega = \langle Q_\Omega, \Sigma, \delta_\Omega, q^0_\Omega, F_\Omega \rangle$ where

- Q_Ω is a new finite set of states (disjoint from Q),
- Σ is a finite set of events (the same as for G),
- $\delta_\Omega : Q_\Omega \times \Sigma \to Q_\Omega$ is a total function representing a deterministic and complete transition relation ($\forall q \in Q_\Omega, \forall e \in \Sigma, \exists! q' \in Q_\Omega, \delta_\Omega(q,e) = q'$),
- q^0_Ω is the initial state,
- F_Ω is the stable set of final states ($\forall q \in F_\Omega, \forall e \in \Sigma, \delta_\Omega(q,e) \in F_\Omega$).

The fact that the set of final states is stable allows us w.l.o.g. to merge all of them into one single absorbing final state. We denote by $L_m(\Omega)$ the marked language of Ω, i.e., the set of words recognized by Ω.

We call an event in the pattern a significant event [Ye & Dague 2012] if it has at least one occurrence that makes the pattern pass from one state to another state. We denote by $S_\Omega(\Sigma)$ the set of all significant events in the pattern Ω: $S_\Omega(\Sigma) = \{e \in \Sigma | \exists q \in Q_\Omega, \delta_\Omega(q,e) \neq q\}$.

[2] Encoding and checking methods are similar for pattern predictability in DES and DDES

The definition of pattern diagnosability follows and generalizes the definition of fault event diagnosability in Chapter 2 (see Definition 4).

Definition 13. (Pattern Diagnosability) A pattern Ω is diagnosable in a system T (we say that T is Ω-diagnosable) iff

$$\exists k \in \mathbb{N}, \forall s \in L(T) \cap L_m(\Omega), \forall t \in L(T)/s, |t| \geq k \Rightarrow$$
$$\forall p \in L(T), (P_o(p) = P_o(s.t) \Rightarrow p \in L_m(\Omega).$$

The above definition states that for any trajectory s in T recognizing the pattern Ω, for any extension t of s in T with enough events, any trajectory p in T that is equivalent to $s.t$ in terms of observation should also recognize Ω. Following the same reasoning as for fault event non-diagnosability made in Chapter 2, non-Ω-diagnosability is equivalent to the existence of a pair of observation-equivalent infinite trajectories in T, one recognizing Ω (the "faulty" one) and the other not (the "correct" one), that we will continue to call a *critical pair* [Genc & Lafortune 2006a].

The difference with simple fault recognition is that a simple fault is recognized simply by the fault event occurrence, while recognizing a pattern means following one of its internal paths from its initial state until one of its final states is reached.

5.3 Related Works

As we mentioned above the seminal works that introduced the pattern diagnosability are [Genc & Lafortune 2006a, Jéron *et al.* 2006], where they studied the centralized system case. Thus they build a global pattern verifier by first synchronizing on all events the pattern with the system (this synchronization, which is actually a product, is always possible and unique in the pattern as this one is deterministic and complete) then building an observer structure like the one defined in [Cassandras & Lafortune 2009] which can be simply obtained by the application of the projection operator on observable events on each trajectory and the synchronization on those observable events of the structure obtained with itself. The idea in these works is to generalize the simple fault event diagnosability by also checking if there exists a critical path in the global pattern verifier. If a path in it has an ambiguous state cycle (where a state is ambiguous if it is made up of a pair of states where exactly one is reached by a trajectory having recognized the pattern) that contains at least one observable event, the non-diagnosability of the pattern is witnessed. Later in [Ye *et al.* 2010, Ye & Dague 2012], the method was extended to distributed systems and, in order to avoid the construction of the global pattern verifier, the authors propose to compute an abstracted pattern verifier for

the subsystem where the pattern recognition is completed to search for partial critical paths; which are paths in the verifier containing an ambiguous state cycle with at least one observable event for all involved components of the considered subsystem. Then they demonstrate how to decide whether a partial critical path in this subsystem can be generalized to a global critical path after checking global consistency using the communication information. In this distributed way, the verifier structure finally obtained and used for diagnosability checking is usually a quite small subpart of the global pattern verifier. Especially if the pattern size is small and that only some components are concerned in its events. But what if it is not the case, i.e., if the size of the pattern is large and it covers most or even all components? We think that such cases are also important and for this we think that employing SAT solvers can be interesting/useful to overcome these limitations, in the light of the size of systems whom we showed they can be dealt with using such approach.

One can imagine a distributed system with a large number of components such that each component has to load and verify some configuration and some input data before proceeding to process them. Let a global observer of the system want to know with certainty that all components have been initialized then have loaded the configuration and then read the input data. If the system model provides the observer by some observation for events about initialization and data processing but not about loading configuration or reading input data, then the ability to answer with certainty the query of the observer can be reduced to the diagnosability checking of the pattern that consists of one sequence of events of the form initialize then load then read. In such cases all components are needed to answer the query and the synchronization approaches may be not practical, however SAT solvers can answer such query even for large systems. We can see how patterns can be employed to answer interesting queries which can be given by assigning semantics to a formal representation as a finite state machine.

We provide here two examples of patterns in figures 5.2 and 5.3 to analyze if they are diagnosable in the system depicted in figure 5.1. The first pattern in figure 5.2 that represents an occurrence of c_1 preceded by f_1 is not diagnosable because we can find two infinite trajectories with only one of them recognizing the pattern like $\rho_1 = f_1 O_1 c_1 (O_2 c_1)^*$ and $\rho_2 = c_1 O_1 c_1 (O_2 c_1)^*$ where we observe for both of them $P_o(\rho_1) = P_o(\rho_2) = O_1 O_2^*$ without knowing if the pattern occurred or not. However the second pattern in figure 5.3 that represents an occurrence of c_1 not preceded by c_2 is diagnosable as we cannot find a counter-example, in fact it is because any occurrence of c_2 is revealed by the observable event O_0 which does not appear in any sequence that contains c_1 not preceded by c_2, in other words O_0 is well placed to reveal c_2.

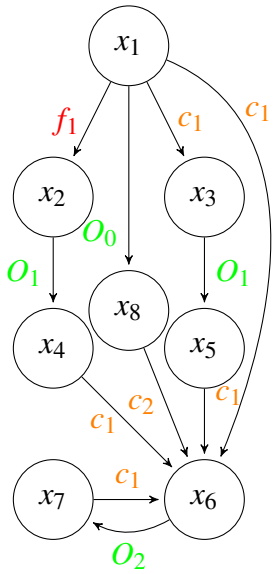

Fig. 5.1 A system to study its diagnosability w.r.t. patterns in Figures 5.2 and 5.3.

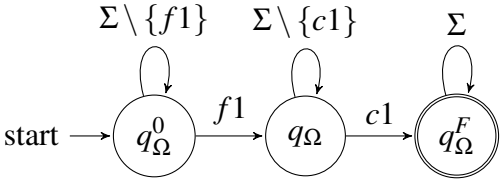

Fig. 5.2 A pattern that represents a sequence with event $f1$ preceding $c1$.

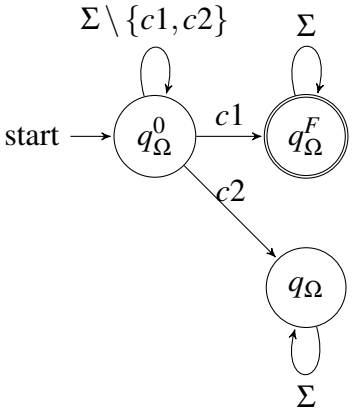

Fig. 5.3 A pattern that represents an occurrence of $c1$ not preceded by $c2$.

5.4 SAT Encoding of Pattern Diagnosability in SLTS

Let a system modeled by an LTS, we know that we can represent it by a succinct transition system, let it be $T = \langle A, \Sigma_u, \Sigma_o, \delta, s_0 \rangle$ with the same definition as introduced in simple fault

diagnosability analysis (see Definition 9) but without the fault events and let $\Sigma = \Sigma_o \cup \Sigma_u$. Let Ω be the pattern whose occurrence diagnosability will be studied, we can also represent it by a succinct system $\Omega = \langle B, \Sigma, \delta_\Omega, s_{\Omega 0}, F_\Omega \rangle$ with the addition of a finite stable set F_Ω of final states. As already noticed, we can always merge together all final states without changing the language recognized by Ω. Therefore we will consider the pattern with only one final state given by the variables assignment $s_{\Omega F}$. As we mentioned above, the only interesting events to inspect while recognizing the pattern are the significant events, i.e., those events in Σ whose at least one occurrence changes one state of Ω, thus they are the only events which hold useful information to tell if the pattern is recognized or not. We denote by $S_\Omega(\Sigma)$ the set of significant events in Ω.

The propositional formulas used to test the pattern diagnosability in the transition system use the following propositional variables:

- a^t and $\hat{a}^t$ for all $a \in A$ and $t \in \{0, \ldots, n\}$.
- b^t and $\hat{b}^t$ for all $b \in B$ and $t \in \{0, \ldots, n\}$.
- e^t_o and $\hat{e}^t_o$ for all $e \in \Sigma$ and $o \in \delta(e)$ and $t \in \{0, \ldots, n-1\}$.
- e^t for all $e \in \Sigma_o$ and $t \in \{0, \ldots, n-1\}$.
- $e^t_{\Omega_o}$ and $\hat{e}^t_{\Omega_o}$ for all $e \in S_\Omega(\Sigma)$ and $o \in \delta_\Omega(e)$ and $t \in \{0, \ldots, n-1\}$.
- e^t_Ω and $\hat{e}^t_\Omega$ for all $e \in S_\Omega(\Sigma)$ and $t \in \{0, \ldots, n-1\}$.

The propositional variables with a hat are in the sequence that does not recognize the pattern, the variables without a hat are in the sequence that recognizes the pattern. For the sake of simplicity and the consistency with terms we used in this thesis, we will use the term correct sequence for the first type and the term faulty sequence for the second type. The two sequences with an Ω subscript represent trajectories in the pattern Ω, the two sequences without it represent trajectories in the system T.

Each propositional variable e^t_o (resp. $\hat{e}^t_o$) indicates an event $e \in \Sigma$ occurring in the faulty (resp. correct) sequence in the system T at timestep t with occurrence $o \in \delta(e)$. Propositional variables e^t, for $e \in \Sigma_o$ an observable event, have the role to synchronize the faulty and correct sequences in T, i.e., e^t_o and $\hat{e}^t_o$, on observable events at the timestep t.

In a similar way, each propositional variable $e^t_{\Omega_o}$ (resp. $\hat{e}^t_{\Omega_o}$) indicates a significant event $e \in S_\Omega(\Sigma)$ occurring in the faulty (resp. correct) sequence in the pattern Ω at timestep t with occurrence $o \in \delta_\Omega(e)$. Propositional variables e^t_Ω (resp. $\hat{e}^t_\Omega$), for $e \in S_\Omega(\Sigma)$ a significant event, have the role to synchronize the faulty (resp. correct) sequences in T and Ω, i.e., e^t_o and $e^t_{\Omega_o}$ (resp. $\hat{e}^t_o$ and $\hat{e}^t_{\Omega_o}$). Thus e^t_Ω and $\hat{e}^t_\Omega$ follow the state changes in the pattern to check

that the faulty sequence will arrive to the final state of Ω and the correct sequence will never reach it.

In order to simulate each step in the system with synchronization on observable events between the faulty and correct trajectories in T and synchronization on significant events of each one with corresponding trajectories in Ω, the following constraints must be applied.

1. The event occurrence e_o^t must be possible in the current state of T:

$$e_o^t \rightarrow \phi^t \quad \forall o = \langle \phi, c \rangle \in \delta(e) \tag{5.4.1}$$

   And the same for $\hat{e}_o^t$.
   Note that, as Ω is complete, any event is always possible in the current state of Ω, so no constraint is necessary.

2. The event occurrence effects must hold at the next time step in T:

$$e_o^t \rightarrow \bigwedge_{l \in c} l^{t+1} \quad \forall o = \langle \phi, c \rangle \in \delta(e) \tag{5.4.2}$$

   where l is a literal that represents a state variable in the change c at timestep t, and l^{t+1} is another literal that represents the same state variable at timestep $t+1$. And the same for $\hat{e}_o^t$.
   The effects of an event occurrence are dealt with similarly in Ω:

$$e_{\Omega_o}^t \rightarrow \bigwedge_{l \in c_\Omega} l^{t+1} \quad \forall o = \langle \phi_\Omega, c_\Omega \rangle \in \delta_\Omega(e) \tag{5.4.3}$$

   And the same for $\hat{e}_{\Omega_o}^t$.

3. The present value (*True* or *False*) of a state variable changes to a new value (*False* or *True*, respectively) only if there is a reason for this change, i.e., because of an event that has the new value in its effects. So, when there is no reason to change the value, this change must be prohibited. Here is the change from *True* to *False* for the system state variables (the change from *False* to *True* is defined similarly by interchanging a and $\neg a$:

$$(a^t \wedge \neg a^{t+1}) \rightarrow (e_{i_1 o_{j_1}}^t \vee \cdots \vee e_{i_k o_{j_k}}^t) \tag{5.4.4}$$

   where the $o_{j_l} = \langle \phi_{j_l}, c_{j_l} \rangle \in \delta(e_{i_l})$ are all the occurrences of events e_{i_l} with $\neg a \in c_{j_i}$. And the same with $\hat{a}^t$ and $\hat{e}_{i_{l_{o_{j_l}}}}^t$.

The same condition must be applied for the pattern state variables and events $e \in S_\Omega(\Sigma)$:

$$(b^t \wedge \neg b^{t+1}) \rightarrow (e^t_{i_1\Omega_{o_{j_1}}} \vee \cdots \vee e^t_{i_k\Omega_{o_{j_k}}}) \tag{5.4.5}$$

where the $o_{j_l} = \langle \phi_{\Omega_{j_l}}, c_{\Omega_{j_l}} \rangle \in \delta_\Omega(e_{i_l})$ are all the occurrences of events e_{i_l} with $\neg b \in c_{\Omega_{j_l}}$. And the same with $\hat{b}^t$ and $\hat{e}^t_{i_l\Omega_{o_{j_l}}}$.

4. At most one occurrence of a given event in T can occur at a time and the occurrences of two different events cannot be simultaneous if they interfere (i.e., if they have two contradicting effects or if the precondition of one contradicts the effect of the other):

$$\neg(e^t_o \wedge e^t_{o'}) \quad \forall e \in \Sigma, \forall \{o, o'\} \subseteq \delta(e), o \neq o' \tag{5.4.6}$$

$$\neg(e^t_o \wedge e'^t_{o'}) \quad \forall \{e, e'\} \subseteq \Sigma, e \neq e', \forall o \in \delta(e), \forall o' \in \delta(e'), o \text{ and } o' \text{ interfere} \tag{5.4.7}$$

We have the same formulas with $\hat{e}^t_o$.
The same condition must be applied to the events in Ω by replacing each e^t_o with $e^t_{\Omega_o}$. And the same for $\hat{e}^t_{\Omega_o}$.

5. Observable events have to be synchronized in the faulty and correct sequences in T:

$$\bigvee_{o \in \delta(e)} e^t_o \leftrightarrow e^t \quad \text{and} \quad \bigvee_{o \in \delta(e)} \hat{e}^t_o \leftrightarrow e^t \quad \forall e \in \Sigma_o \tag{5.4.8}$$

In the same way, faulty sequences in T and Ω have to be synchronized on all events, and the same for correct sequences (formulas are unchanged if Ω is not deterministic):

$$\begin{aligned} &\bigvee_{o \in \delta(e)} e^t_o \leftrightarrow e^t_\Omega \quad \text{and} \quad \bigvee_{o \in \delta_\Omega(e)} e^t_{\Omega_o} \leftrightarrow e^t_\Omega \quad \forall e \in S_\Omega(\Sigma) \\ &\bigvee_{o \in \delta(e)} \hat{e}^t_o \leftrightarrow \hat{e}^t_\Omega \quad \text{and} \quad \bigvee_{o \in \delta_\Omega(e)} \hat{e}^t_{\Omega_o} \leftrightarrow \hat{e}^t_\Omega \quad \forall e \in S_\Omega(\Sigma) \end{aligned} \tag{5.4.9}$$

One can notice that when a significant event e is observable one can replace the lefthand sides of the left equivalences in (5.4.9) by the reference variable e^t used in the equations (5.4.8). This will generate shorter clauses which is preferred by the SAT solvers.

6. To avoid trivial cycles we require that at every time point at least one event takes place in T:

$$\bigvee_{e \in \Sigma_o} e^t \quad \vee \quad \bigvee_{e \in \Sigma_u, o \in \delta(e)} e^t_o \quad \vee \quad \bigvee_{e \in \Sigma_u, o \in \delta(e)} \hat{e}^t_o \tag{5.4.10}$$

There is no need of this condition in Ω as each significant event in it is synchronized with a system event and we do not care about non significant events.

The conjunction of all the above formulas for a given t is denoted by $\mathscr{T}(t,t+1)$.

A formula for the initial state $(s_0, s_{\Omega 0})$ is:

$$\begin{aligned}\mathscr{I}_0 \quad = \quad & \bigwedge_{a\in A, s_0(a)=1} (a^0 \wedge \hat{a}^0) \quad \wedge \quad \bigwedge_{a\in A, s_0(a)=0} (\neg a^0 \wedge \neg \hat{a}^0) \\ \wedge \quad & \bigwedge_{b\in B, s_{\Omega 0}(b)=1} (b^0 \wedge \hat{b}^0) \quad \wedge \quad \bigwedge_{b\in B, s_{\Omega 0}(b)=0} (\neg b^0 \wedge \neg \hat{b}^0)\end{aligned} \tag{5.4.11}$$

Formulas expressing that the final state assignment $s_{\Omega F}$ in the pattern is satisfied in the faulty sequence, resp. not satisfied in the correct one, at timestep t are given by:

$$\begin{aligned}\mathscr{F}^t_\Omega \quad &= \quad \bigwedge_{b\in B, s_{\Omega F}(b)=1} b^t \wedge \bigwedge_{b\in B, s_{\Omega F}(b)=0} \neg b^t \\ \hat{\mathscr{F}}^t_\Omega \quad &= \quad \bigvee_{b\in B, s_{\Omega F}(b)=0} \hat{b}^t \vee \bigvee_{b\in B, s_{\Omega F}(b)=1} \neg \hat{b}^t\end{aligned} \tag{5.4.12}$$

At last, the following formula can be defined to encode the fact that a pair of trajectories in T from the initial state is found with the same observable events (first line (5.4.13)), whose exactly one of them recognizes the pattern Ω (second line (5.4.14)) and which are infinite (in the form of cycles at step n, third line (5.4.15)), witnessing non-Ω-diagnosability:

$$\Phi^T_n(\Omega) \quad = \quad \mathscr{I}_0 \quad \wedge \quad \mathscr{T}(0,1) \wedge \cdots \wedge \mathscr{T}(n-1,n) \quad \wedge \tag{5.4.13}$$

$$\mathscr{F}^n_\Omega \quad \wedge \quad \hat{\mathscr{F}}^n_\Omega \quad \wedge \tag{5.4.14}$$

$$\bigvee_{m=0}^{n-1} \Big(\bigwedge_{a\in A} ((a^n \leftrightarrow a^m) \wedge (\hat{a}^n \leftrightarrow \hat{a}^m)) \Big) \tag{5.4.15}$$

From this propositional encoding follows the result that an SLTS T is not Ω-diagnosable iff $\exists n \geq 1, \Phi^T_n(\Omega)$ is satisfiable, which is also equivalent to $\Phi^T_{2^{2(|A|+|B|)}}(\Omega)$ being satisfiable (actually the upper bound $2^{2|A|}$ is enough if we suppose Ω deterministic).

Actually, if $\Phi^T_n(\Omega)$ is satisfiable then one can translate directly the valuation of the variables in a model of it into two infinite trajectories in T that coincide on their observations and exactly one of them recognizes Ω. Reciprocally if T is not Ω-diagnosable, hence there exists two infinite trajectories in T that are synchronized on the observable events and only one of them recognizes Ω; then we can construct from these trajectories a valuation of all propositional variables which ensures the satisfiability of $\Phi^T_n(\Omega)$.

5.5 SAT Encoding of Pattern Diagnosability in DSLTS

Considering the distributed DES means including a system of more than one component and also that significant events in the pattern may contain events from more than one component of the system (if communication events are allowed in the pattern). One can easily see that adding more than one component to the previous formalism can be done smoothly just by synchronizing each significant event in the pattern with its owner component(s) in the system. Thus, if it is a private event in one component then we just adopt the above formulas to do the synchronization, else if it is a communication event, thus shared between two or more components, we have just to add a synchronization formula between the reference variable, already used in simple fault DSLTS diagnosability encoding to synchronize its occurrences in all its owner components, and the corresponding event in the pattern. Notice that having both observable and unobservable communication events can be also handled in a similar way to what we have done in simple fault diagnosability case. In other words, lefthand sides in the left equivalences in (5.4.9) can be replaced with the unique reference variable in case of observable communication events to be synchronized with the pattern (cf. formula (3.2.3)). Similarly, for unobservable communication events, replacement is done with the two reference variables of communication events, one in each trajectory (cf. formula (3.2.2)). It worth to say that in [Ye *et al.* 2010] the communication events were excluded from the pattern, which facilitated the synchronization process in their Twin Plant based algorithm; however we can see here that we can smoothly extend our encoding from the simple fault diagnosability to the pattern diagnosability in DDES, without restriction on the pattern events. Also we believe that including the communication events in the pattern can be interesting to study the communications behavior of the system, which can be useful to decide some their features at the design stage, like making them observable or not.

5.6 Conclusion

We reviewed in this chapter the pattern diagnosability problem, how it generalizes the fault event diagnosability problem and how it can be used to answer complicated queries about the behaviors of the system. Thus any extension-closed rational language can represent a pattern to study its diagnosability in the system. Then we presented an encoding in SAT for this problem in the centralized case and showed how to extend it to the distributed case, allowing in full generality observable or unobservable communication events to occur in the pattern, which was not dealt with in the literature. We did not yet test this encoding and thus have no experimental results for the moment, however we expect a scalability improvement that helps

in overcoming the limitations in the literature such as supposing a small pattern in centralized systems or moreover, in a distributed structure, considering usually only some components of the system to be concerned by the pattern. The encoding of pattern predictability problem can be also done in a similar way by using our fault event predictability encoding and the approach we adopted here by considering the pattern as a new added component to the system. We look for testing these encodings for both diagnosability and predictability in centralized and distributed systems on some real examples.

Chapter 6

Diagnosability Planning for Controllable DES

We introduce in this chapter a new problem which we called diagnosability planning in controllable discrete event systems. It consists in calculating, from a set of given elementary control actions, an optimal plan that drives the system from a set of potential current states to a surely diagnosable set of states. We first present our motivations and the potential interests intended from studying such problem. Then we describe the controllable DES, we analyze the problem and prove its complexity class. After that, we provide three methods to solve the problem and we test them, according to two cost functions, on an artificial benchmark that we created for this purpose.

6.1 Motivation/Introduction

We have studied in the first part of this thesis the problems of diagnosability and predictability checking. We mentioned that such properties are ensured *since the design stage* in order to get their benefits during *the operation stage*. Otherwise adding sensors afterwards to be able to diagnose a running system can be much more expensive. To this end, each counter-example that proves the violation of a property during a design model test is usually followed by some sensors modification. The process is incrementally repeated until proving the property before delivering it to be operated. During this process many criteria can be adopted to get an optimal sensor placement and many works have considered this problem of "restoring diagnosability" of a system model under some conditions [Briones *et al.* 2008, Ribot *et al.* 2007, Yan 2004]. However these approaches are considered at the design stage, i.e., all modifications are done before putting the system in operation.

Moreover, diagnosability is usually ensured regarding to some limited number of predefined initial states, i.e., no critical path can be constructed *starting from them.* Although in the operation mode diagnosable faults will be detected, the natural reaction after this detection is to apply a sequence of actions in order to repair the system. In many cases these actions will drive the system into "new initial" states where diagnosability has not been yet ensured. Similarly, when a fault is predictable, some actions will be applied to prevent its occurrence since foreseeing it. Therefore, we need actions to be applicable in such reparation or to make intentional changes of the system states. Hence controllable systems are defined to provide two layers of modeling. The first one represents a model of spontaneous and autonomous correct and faulty behaviors of a partially observed discrete event system. The second layer provides a set of elementary actions, by the means of simple binary transitions, that can be fired or prevented intentionally by a controller to force or prevent specific changes of the system state. This set of exogeneous actions is usually in correspondence with a subset of the system events set, as pairs of output/input communication events modeling the interaction of the environment with the system.

In order to control each potential situation and ensure the diagnosability of a running system, the problem of active diagnosis has been introduced in [Sampath *et al.* 1998]. It consists in synthesizing a controller that limits the system possible behaviors in order to keep it diagnosable. However, this may affect the main mission of the system like in an autonomous system where all capabilities may be required to achieve its mission. The work [Chanthery & Pencolé 2009] proposed a solution that does not restrict directly the number of actions, by building a complete active diagnoser, simulating all possible runs of the system, then use planning to get rid of its ambiguous states. The initial states in this planning process are the ambiguous states in the active diagnoser and the goal states are the non-ambiguous states.[1] However their planning algorithm is naive and consists in iterating on each ambiguous state in the active diagnoser and choosing a plan among all its admissible plans, according to an optimization criterion. Moreover it is not implemented. Actually their on-line planning may contradict the mission of the system simulated over the active diagnoser and this is still a problem with this approach. Another problem is that enumerating all possible plans from each ambiguous state could be impractical. In fact the decision problem of active diagnosability is proven to be EXPTIME-complete in [Haar *et al.* 2013] and the size of a minimal synthesis, over the exponential sized active diagnoser, still has an exponential size.

[1] Multiple faults are considered in the active diagnoser, but plans are searched one by one then filtered to satisfy all goals if possible.

In order to mitigate the above problems, we introduce in this chapter the concept of *diagnosability planning* which can be seen as a "light" version of active diagnosability. Thus it reduces the interactions with a running system. It replaces the strict controlling of each system step, achieved with the active diagnoser, by the construction of a plan that ensures/restores the diagnosability of the system whenever it is lost. This plan, called *diagnosability plan*, derives a given input belief state (several potential current states) into a diagnosable belief state. After its application the system is then left to run freely. Differently from the approach in [Chanthery & Pencolé 2009] the active diagnoser is not constructed and the plan is calculated in an intelligent way. Thus in our approach the control phase, through planning, is called only on demand and thanks to its output the system is allowed to run freely with guaranteed diagnosability. These two successive stages of our approach keep the diagnosability planning problem, including its diagnosability tests, in PSPACE in comparison to the EXPTIME complexity for the more complex active diagnosability used usually in such cases.

An application of this approach would be interesting after an accidental disruption of the system because of a repair plan or an external intervention on the system. Thus after such cases the system would be described by a set of potential states from where its diagnosability is not guaranteed. This approach is also interesting to reduce the diagnostic interaction with running systems, like power networks where these interactions have an important cost in both materials and their effects on the service availability.

From a planning point of view, our problem differs from the classical planning problems and also from the conformant planning ones (dealing with uncertainty, so with belief states) in that there is no explicit final goal states as input of the problem. Our goal states are those from where the diagnosability property holds, which are unknown at the beginning. We get around this by using diagnosability test, through its auxiliary Twin Plant structure, to characterize such goal states. And we reuse along the plan search diagnosability information found from previously built parts of the Twin Plant structure by tagging the unambiguous Twin Plant states according to this test with one of two tags: good or bad. As we know, a Twin Plant state represents a pair of system states with fault occurrence information. Only the good states of the Twin Plant will serve to define our goal states. The pairs that are considered are members of the power set of the current belief state whose diagnosability is being checked. Actually, all possible pairs in the power set of the belief state must be good to get a diagnosable belief state. Nevertheless, this intended belief state may have been preceded during the plan search by several non-diagnosable belief states resulting from failed candidate plans. They include thus some partial tagging (at least one bad pair) that can be recycled to prune the further searches. In fact we will exploit these tags in two ways,

firstly to prune the construction of Twin Plants used to verify the diagnosability of the current intended goal belief state, secondly to prune the test of specific plan candidates during the search. More details will follow.

This chapter is organized as follows. Section 6.2 introduces the concepts needed: our model of controllable DES, the definition of diagnosability regarding a belief state and its analysis by using the Twin Plant structure, the conformant planning problem; and then defines the problem of diagnosability planning. Section 6.3 provides complexity results. Section 6.4 proposes an algorithm for computing a diagnosability plan, where information about already constructed parts of the Twin Plant is exploited for pruning the search and the diagnosability checking, and illustrates it on an example. Section 6.5 presents the construction of a scalable benchmark and experimental results of the algorithm on several of its instances. Section 6.6 presents related works. Section 6.7 concludes and draws perspectives of future research.

6.2 Diagnosability Planning Problem for Controllable DES

6.2.1 Controllable Discrete Event Systems

As mentioned above, this work is done in the context of controllable DES. We assume that the system runs in two distinct modes that do not intertwine: the active one when the system runs freely, i.e., its states are changed autonomously through partially observable transitions without any controlled exogenous event, and the reactive one in which only *feasible* exogenous actions are applied to change the system state through reactive transitions.

Definition 14. A **Controllable Labeled Transition System** (CLTS) is a tuple $G = \langle Q, \Sigma, \delta, I \rangle$ with $\Sigma = \mathscr{A} \cup \mathscr{E}$ where:

- Q is a finite set of states,
- $\mathscr{E}$ is a finite set of events,
- $\mathscr{A}$ is a finite set of actions,
- $\delta \subseteq Q \times \Sigma \times Q$ is the (active for labels in $\mathscr{E}$, reactive for labels in $\mathscr{A}$) transition relation,
- $I \subseteq Q$ is the initial belief state.

$\mathscr{E}$ is partitioned into $\{\Sigma_o, \Sigma_u, \Sigma_f\}$, where Σ_o is a finite set of observable correct events, Σ_u is a finite set of unobservable correct events, Σ_f is a finite set of unobservable faulty events. We suppose that every state in Q is actively reachable from a state in I. We assume that the system is complete in terms of actions, i.e., every action is applicable in every state. This assumption can be easily made by modeling every action missing in a state q

as a loop transition in q (so leaving q unchanged). We do not suppose that the actions are deterministic.

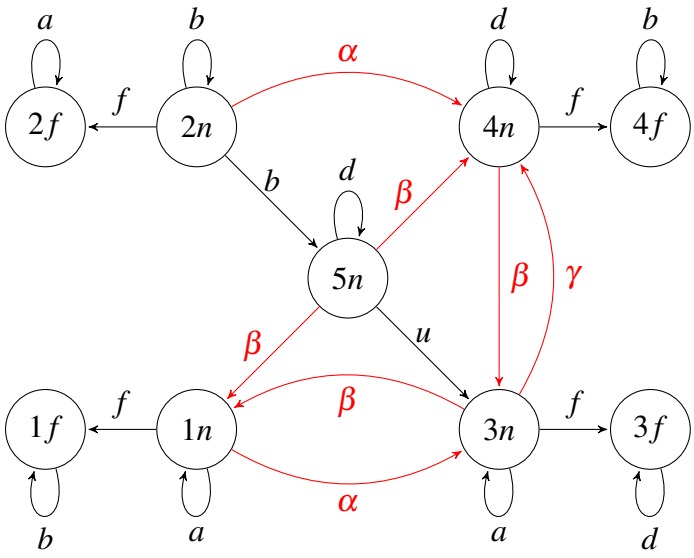

Fig. 6.1 Illustrative Example of a CLTS

Figure 6.1 shows a CLTS that comprises 9 states with $I = \{1n, 2n, 3n, 4n, 5n\}$, $\mathscr{A} = \{\alpha, \beta, \gamma\}$, $\Sigma_f = \{f\}$, $\Sigma_o = \{a, b, d\}$ and $\Sigma_u = \{u\}$. In this diagram, any action that is missing in a state is assumed to leave the state unchanged (for instance, applying β in $1n$ leads to state $1n$: $(1n, \beta, 1n) \in \delta$). Notice that actions, here, do not affect the possibility of the fault occurrence, i.e., no actions sequence may lead to a fault-free behavior, otherwise the problem would be trivial boiling down to take such an actions sequence.

6.2.2 Diagnosability in Controllable DES

We keep the definitions and notations of the subsection 2.2. Paths and language are w.rt. active transitions, i.e., events in $\mathscr{E}$ and initial belief state I.

Definition 15. An **active path** (or simply a path) is a sequence of active transitions $\rho = q_0 \xrightarrow{e_1} \cdots \xrightarrow{e_n} q_n$, with $e_1, \ldots, e_n \in \mathscr{E}$, $q_0, \ldots, q_n \in Q$ and $\forall i \in \{0, \ldots, n-1\}, (q_i, e_{i+1}, q_{i+1}) \in \delta$. We call $e_1 \ldots e_n \in \mathscr{E}^*$ the trajectory of ρ.

$L_I(G)$ will denote the prefix-closed language of G from the initial belief state I, i.e., the set of words from $\mathscr{E}^*$ that are the trajectories of some active paths in G that start from a state in I. Elements of $L_I(G)$ are called I-trajectories. The definition of diagnosability from an initial belief state I follows immediately from Definition 4.

Definition 16. A fault $f \in \Sigma_f$ is **diagnosable** in a system G with initial belief state I iff

$$\exists k \in \mathbb{N}, \forall s^f \in L_I(G), \forall t \in L_I(G)/s^f, |t| \geq k \Rightarrow$$
$$\forall \rho \in L_I(G), (P_o(\rho) = P_o(s^f.t) \Rightarrow f \in \rho).$$

Notice that this definition boils down to Definition 4 with a single initial state by adding to G such a state q^0 and transitions from q^0 to each state in I, labeled by an unobservable correct event. Assumptions 1, 2 and 3 are kept unchanged in this chapter.

In our example, the initial belief state is not diagnosable as it exists for example, from its initial states $1n$ and $2n$, an arbitrary long observable sequence of the event a (a^+) that might represent faulty (moving to $2f$ then looping there) or normal (looping in $1n$) I-trajectory in the system.

Diagnosability checking of LTS is known to be polynomial in the number $|Q|$ of states [Jiang *et al.* 2001] as we reviewed in the chapter 2. We will restate here some useful information about this approach for the sake of readability. The polynomial algorithm to check diagnosability is based on the Twin Plant method [Jiang *et al.* 2001, Yoo & Lafortune 2002]. The first step in this method is the construction, from the original system model G, of a non-deterministic automaton, called pre-diagnoser, designed to preserve all observable information from G and to append to every state an estimate of failure information. In the pre-diagnoser, we only keep the observable events and attach the fault information (in the form of a fault label equal to $\{f\}$ if the fault has occurred on the path followed from a sate in I to the given state and to $\emptyset$ otherwise) to each retained state. For the example in Figure 6.1, in its pre-diagnoser, the fault label for all states $1n$, $2n$, $3n$, $4n$, $5n$ in the initial belief state is $\emptyset$ since there is no fault occurrence up to them. The fault label for all other states is $\{f\}$.

The second step is to build the Twin Plant by synchronizing the pre-diagnoser with itself based on the observable events, in order to obtain all pairs of system I-trajectories with the same observations as Twin Plant trajectories. Each state of the Twin Plant is thus a pair of pre-diagnoser states that provide two possible diagnoses with the same observations. Given a Twin Plant state, if the fault label $\{f\}$ is attached to exactly one of its two associated pre-diagnoser states, i.e., the occurrence of f is not certain with the given observations, it is called an ambiguous state (w.r.t. f). An ambiguous state cycle is a cycle containing only ambiguous states. A *critical path* is a path in the Twin Plant issued from a pair of initial states with a prefix followed by an ambiguous state cycle. A system is non-diagnosable iff its Twin Plant contains a critical path.

Now consider our illustrative example. In its Twin Plant, we have the path $((1n,\emptyset)(2n,\emptyset)) \xrightarrow{a}$ $((1n,\emptyset)(2f,\{f\})) \xrightarrow{a} ((1n,\emptyset)(2f,\{f\}))$. This is actually a critical path since it is issued

from a pair of initial states $1n$, $2n$ and contains a self cycle with an ambiguous state $((1n,\emptyset)(2f,\{f\}))$ associated with an observable event a. Thus, the system corresponding to this example is not diagnosable since one can never be sure about the fault occurrence with an arbitrary number of a observed.

6.2.3 Planning

Definition 17. A **plan** π for a CLTS G is a sequence of actions $a_i \in \mathscr{A}$.

Applying the plan $\pi = a_1 \dots a_m$ from a current belief state $S \subseteq Q$ leads to the belief state $\pi(S)$ defined recursively by $\pi(S) = S$ if $m = 0$ and $\pi(S) = \pi'(S')$ if $m > 0$, where $S' = \{q' | \exists q \in S \ (q, a_1, q') \in \delta\}$ and $\pi' = a_2 \dots a_m$. Suppose now $\pi = \alpha\beta$, for our example, $\pi(I) = \pi'(I')$, where $\pi' = \beta$ and $I' = \{3n, 4n, 5n\}$, thus $\pi(I) = \{1n, 3n, 4n\}$.

In planning, the objective is generally to find a plan that leads the system into a belief state that is included in one among a given set of "acceptable" states (e.g., states where the system is safe or where the system provides the service that we expect from it): $\pi(I) \subseteq B$ for some $B \subseteq Q$, B acceptable. Here the definition of the objective is not at the level of an individual state but at the level of a belief state (set of states).

Definition 18. A **planning problem** is a pair $\langle G, O \rangle$ where $G = \langle Q, \Sigma, \delta, I \rangle$ is a CLTS and $O \subseteq 2^Q$ is a collection of belief states. The solution to the planning problem is a plan π such that $\pi(I) \subseteq B$ with $B \in O$.

6.2.4 Problem Definition

In our problem, the objective of planning is to find a plan that leads the system to a diagnosable belief state.

Definition 19. Given a CLTS $G = \langle Q, \Sigma, \delta, I \rangle$, **diagnosability planning** is the problem of finding a plan π such that $\langle Q, \Sigma, \delta, \pi(I) \rangle$ is diagnosable.

Diagnosability planning can be equivalently phrased as the problem of solving the planning problem $\langle G, O \rangle$ where O is the set of maximal belief states from where G is diagnosable: $O = \{I' \subseteq Q | \langle Q, \Sigma, \delta, I' \rangle$ is diagnosable and I' maximal$\}$. The difficulty here is that, unlike usual planning problems, this set O is not explicitly given as an input but implicitly specified by a property and we will show how we can characterize it. A plan is *correct* iff it satisfies the objective, here we will call it a *diagnosability plan*.

Our goal is to find an optimal diagnosability plan for given criterion (or set of criteria). First of all, the criterion will be the length of the plan (number of actions in the sequence) that

we want to minimize. We denote $\pi^\star$ an optimal plan. In our example, the plan $\pi^\star = \alpha\beta\beta$ is an optimal diagnosability plan that leads the system to the belief state $\pi^\star(I) = \{1n, 3n\}$ from where no critical path can be constructed.

6.3 Complexity

In this section, we compute the complexity of the decision problem associated with diagnosability planning. We demonstrate the following result:

Theorem 6.3.1. *Diagnosability planning is* PSPACE-COMPLETE.

Hardness can be shown by reducing classical propositional planning to Conformant Planning (i.e., with undeterminism of initial state and/or actions, thus belief-state planning as defined in Definition 18) with eXplicit States (CPXS), and then CPXS to diagnosability planning. Classical propositional planning is known to be PSPACE-COMPLETE [Bylander 1994], which will give the result. As we did not find the first reduction in the literature, we provide a proof sketch below.

A classical propositional planning is defined in a succinct way by a set of propositional variables and a set of actions. Each action has a precondition (subset of variables that must be *True* for the action to be applicable) and two sets of positive/negative effects (variables that become *True/False* upon the application of the action). Furthermore the problem specifies the list of variables *True* in the initial state and the list of variables *True* in the goal states. The goal is to find a sequence that leads from the initial state to one goal state.

For each variable v we create two states v_0 and v_1 in the CPXS problem that represent respectively the fact that the variable v has *False* or *True* assignment in the classical planning problem. A state in the classical planning problem is therefore represented by a belief state (that never contains both states v_0 and v_1 for a same v) in the CPXS problem. The initial belief state and the final objective belief states are set accordingly, corresponding to the initial state and the goal states. For any action a and any variable v we create in the CPXS problem a transition labeled by a from v_i to v_j that models the effect of a from the perspective of variable v, so according to the precondition/effects of a and the semantics of v_0/v_1 described above (transition to the same state if action a is not applicable). It is easy to see that the solutions for the reduced CPXS problem are thus the same as the solutions for the original classical planning problem. The second reduction can be done by adding active transitions labeled by events, in such a way that no ambiguous state cycle can be reached from any CPXS objective belief state while any other belief state gives birth to an ambiguous state cycle. A solution plan to the CPXS problem corresponds thus to a diagnosability plan for the controllable DES obtained.

Membership to PSPACE can be shown by a proof similar to membership of classical propositional planning. We iterate over all the possible belief states (PSPACE), verify that this belief state is diagnosable (PTIME), and search for a conformant plan from the initial belief state to this belief state (PSPACE).

6.4 Solving the Diagnosability Planning Problem

6.4.1 Analyzing the Problem

Solving the Diagnosability Planning problem requires finding a plan that leads the system from a given belief state into a diagnosable belief state, called a goal belief state. This consists in alternating the generation of candidate plans and the diagnosability test of the belief state reached by each candidate plan. Thus, one must ensure the absence of any critical path that could be constructed from this final belief state. The traditional way to do diagnosability test is just applying from scratch the Twin Plant approach: we call this approach *Normal method*. Regarding the planning steps, we have to generate the candidate plan by browsing the search space of these candidates with a browsing algorithm, like Breadth First Search (BFS) for example, which ensures plan size minimality. The search space size can vary depending on the initial belief state and the type of available actions. For example, theoretically, if we have only one state in the initial belief state with deterministic actions, then the worst case of the solution plan size, if it exists, is $|Q|$. If the initial belief state contains more than one state or the actions are not deterministic, the plan size is bounded by $2^{|Q|}$, therefore in the worst case the plan can be encoded with $|Q|$ bits. We consider here multiple states initial belief state and non-deterministic actions.

From another hand, during the search of the intended plan, the different Twin Plants constructed starting from different belief states will generally share some states with each other. This makes interesting the idea of recycling previously built parts of these Twin Plants (which can be seen as included in the implicit global Twin Plant of the system that would correspond to $I = Q$). In particular, if we find any critical path during the test of a candidate plan, we know that each time we will meet again its starting state, which is the root of this constructed Twin Plant, we will be sure to recover a critical path. This state represents a pair of states in the source belief state. Hence the idea is to tag such a pair as a *bad pair* after each test in order to avoid re-testing it later if it occurs in another planned belief state. But more interestingly, these pairs can be used to prune the next Twin Plants construction and even to guide the search of a diagnosability plan.

Definition 20. We call $\{q_1, q_2\} \in Q \times Q$ a **bad pair** iff $((q_1, \emptyset), (q_2, \emptyset))$ is the starting state of a critical path in the Twin Plant of G. If $q_1 = q_2$, then it is called a bad unit. We denote the set of all currently known bad pairs as $\mathscr{B}$.

Actually, when a critical path is discovered, not only its starting state but all its non-ambiguous states (i.e., before the fault occurrence) give rise to bad pairs. Notice that the system $G = \langle Q, \Sigma, \delta, I \rangle$ is not diagnosable iff $I \times I$ contains a bad pair. So, for a current state of knowledge, a sufficient condition for the non-diagnosability of G is $(I \times I) \cap \mathscr{B} \neq \emptyset$.

In a similar way if the diagnosability test failed to find any critical path issued from a non-ambiguous pair of states we can call this pair a *good pair*.

Definition 21. We call $\{q_1, q_2\} \in Q \times Q$ a **good pair** iff there is no critical path in the Twin Plant of G issued from $((q_1, \emptyset), (q_2, \emptyset))$. If $q_1 = q_2$, then it is called a good unit. We denote the set of all currently known good pairs as $\mathscr{G}$.

Actually, when no critical path exists, not only the starting state of the Twin Plant but all its non-ambiguous states give rise to good pairs. Notice that the system $G = \langle Q, \Sigma, \delta, I \rangle$ is diagnosable iff $I \times I$ contains only good pairs. So, for a current state of knowledge, a sufficient condition for the diagnosability of G is $I \times I \subseteq \mathscr{G}$.

In our example, the pair $\{1n, 2n\}$ is a bad pair since from it a critical path can be constructed, as shown before. $\{1n, 3n\}$ is a good pair because there is no critical path issued from it.

One natural way to build the Twin Plant for any belief state S is to process S globally by reducing it to only one virtual initial state $\{q^S\}$ related by unobservable transitions to any state in S and to take into account learned tagged pairs while constructing the traditional Twin Plant from $((q^S, \emptyset), (q^S, \emptyset))$. We call this approach *Lazy Learning method*, as it will learn only bad pairs and so will not exploit the good pairs; actually it will not discover them until the diagnosability plan is found with no more needs to recycle them. However, this approach may lead to small size constructed Twin Plants by concentrating only on bad pairs. Another way to a better exploitation of the tagged pairs is to construct the Twin Plants for each pair of states in the belief state, i.e. for each element of $S \times S$. Thus we stop the process immediately if we meet any known bad pair and, before this, we prune developing any branch of the Twin Plants each time we meet a known good pair. We call this approach, allowing learning both good and bad pairs and a useful recycling of both, *Eager Learning method*.

6.4.2 Learning and Exploiting Bad and Good pairs

In Diagnosability Test

As we said above, meeting a known bad pair during a Twin Plant construction predicts the existence of a critical path. This allows us to completely stop the construction of the current Twin Plant and learn at least one new bad pair (the one corresponding to the starting state of the Twin Plant). In our example, assuming that the bad unit $\{5n\}$ has been previously discovered and learned, testing $\{2n\}$ can be stopped just after the construction in the Twin Plant of $((2n,\emptyset),(2n,\emptyset)) \xrightarrow{b} ((5n,\emptyset),(5n,\emptyset))$ and the new bad unit $\{2n\}$ be learned. Many other bad pairs can actually be learned in general, corresponding to all non-ambiguous states in the critical path discovered. Moreover, we can avoid testing any candidate plan π that leads to a belief state that contains a known bad pair, i.e. such that $(\pi(I) \times \pi(I)) \cap \mathscr{B} \neq \emptyset$. In our example, knowing that $\{1n,2n\} \in \mathscr{B}$, it is useless to test $\pi = \beta$ and $\pi = \gamma$ as $\beta(I) = \{1n,2n,3n,4n\}$ and $\gamma(I) = \{1n,2n,4n,5n\}$.

The *Lazy Learning method* concentrates only on exploiting those bad pairs, however the *Eager Learning method* exploits also the good pairs generated after each Twin Plant construction to guide the plan generation and also to prune the further parts of Twin Plants construction. Thus, discovering a good pair means that the Twin Plant constructed starting from this pair, and using all possible output edges from each state in this pair, does not contain any critical path. This can be exploited in two ways. The first is that each state in the constructed Twin Plant represents a good pair, that can be learned. The second is that in any next Twin Plant, meeting a good pair allows pruning the construction of branches from this state (construction in other branches has to be continued). Moreover, for a candidate plan π, we can avoid testing all known good pairs in the belief state reached by π, i.e., all $\{q_1,q_2\} \in (\pi(I) \times \pi(I)) \cap \mathscr{G}$. In our example, suppose that when testing diagnosability from I, i.e. for $\pi = \emptyset$, and before finding the bad pair $\{1n,2n\}$, the good pairs $\{1n,1n\}$, $\{3n,3n\}$ and $\{1n,3n\}$ have been found and thus learned. Then, when later considering the candidate plan $\pi = \alpha\beta\beta$, as $\pi(I) = \{1n,3n\}$, it is useless to test it to conclude it is a diagnosability plan.

In Planning

We can use all discovered pairs in guiding a greedy algorithm for the plan generation. Let call $g(\pi)$ the cost of plan π , which is the sum of its elementary actions costs (e.g., its length if all actions have cost 1). We want to order the possible plans according not only to their cost in terms of actions but also to the chance of reaching a diagnosable belief state, heuristically evaluated by the ratio of good pairs in this belief state. For this, we classify the candidate

plans π (those obtained by adding one action to a previous failed candidate plan, but more generally one could consider extending failed candidate plans by actions up to a temporary maximal length) by partitioning $\pi(I) \times \pi(I)$ into three classes $\langle \mathscr{B}_\pi, \mathscr{G}_\pi, \mathscr{U}_\pi \rangle$, which represent respectively known bad, known good and non-tagged pairs. We keep those π with $\mathscr{B}_\pi = \emptyset$. The estimated cost function for the good pairs ratio is denoted by $h(\pi)$. Then, the best candidate plan, i.e. the one that optimizes the global cost function (combination of $g(\pi)$ and $h(\pi)$ minimizing the first and maximizing the second) is chosen (so, the best action for local optimization when extending plans by only one action). Then we test for diagnosability $\mathscr{U}_\pi$ for the plan π chosen, by the same previous iterative Twin Plant construction. The new bad and good pairs discovered at this occasion are used to update or compute the classes $\langle \mathscr{B}_\pi, \mathscr{G}_\pi, \mathscr{U}_\pi \rangle$ for the next candidate plans π considered in order to proceed to their ranking. The process is repeated until finding a diagnosability plan or proving its absence. The tagged pairs (good and bad) are propagated into all the classes, so any pair is never tested more than once with the *Eager Learning method*, and will be used to prune or stop the construction of the further Twin Plants.

One can learn three things by observing this structure after each failed plan; the first is the existence of diagnosability plans π that can be directly deduced if ($\mathscr{B}_\pi = \emptyset \ \wedge \ \mathscr{U}_\pi = \emptyset$) in any open belief state, the second is the absence of a diagnosability plan deduced if no more candidate plans exist and *all* classes are of the form ($\mathscr{B}_\pi \neq \emptyset$) and the third is that, if none of the two stopping conditions is satisfied, the next possible test is the one on the top of the priority queue.

6.4.3 General Algorithm

We propose the algorithm 1 which contains the general procedure that learns bad and good pairs and uses them to prune the search space, i.e., the *Eager Learning method*. This pruning is twofold: pruning the search space of the candidate plans and pruning the construction of the Twin Plants. Other procedures that add strategies for special exploitation of the bad units are also possible here.

The algorithm starts by generating a set of candidate plans by enriching the previous candidate plans not yet tested (none at the beginning) by the new ones obtained by extending in any way by one action the last chosen best plan candidate that just failed (so, at the beginning, this set is made up of the plans of length one), which is done by the procedure *genCandidatePlan*. This procedure prunes tests of candidate plans that lead to a belief state which is a superset of the belief state of a previously explored plan or of another candidate plan generated with a lower cost. The candidate plan with this superset belief state is then closed. Therefore no candidate plan will be generated if all nodes are closed, which is

Algorithm 1 General algorithm

procedure FINDDIAGNOSABILITYPLAN(G, I, g, h)
  $\pi \leftarrow \emptyset$; $\Pi \leftarrow \emptyset$; $EBS \leftarrow \{I\}$; $\mathscr{B} \leftarrow knownBad$; $\mathscr{G} \leftarrow knownGood$
  ▷ Initialization of the last candidate explored, the set of current candidates, the belief states explored, the bad and good pairs known from testing the diagnosability of G_I
  $\Pi \leftarrow genCandidatePlan(\Pi, \pi, EBS, G, I, g)$
  while $\Pi \neq \emptyset$ **do**
    $\pi \leftarrow getBestCandidate(\Pi, \mathscr{B}, \mathscr{G}, g, h)$
    $EBS \leftarrow EBS \cup \{\pi(I)\}$
    $Pairs \leftarrow getAllPossiblePairs(\pi(I))$
    while $(Pairs \cap \mathscr{B} = \emptyset) \wedge (Pairs \setminus \mathscr{G} \neq \emptyset)$ **do**
      $\{q_1, q_2\} \leftarrow chooseOne(Pairs \setminus \mathscr{G})$
      $DG^{\pi}_{(q_1,q_2)} \leftarrow getPreDiag(G_{\{q_1,q_2\}})$
      $TP^{\pi}_{(q_1,q_2)} \leftarrow getTP(DG^{\pi}_{(q_1,q_2)}, \Sigma_o, \mathscr{B}, \mathscr{G})$
      ▷ $\mathscr{B}$ and $\mathscr{G}$ are used to prune TP building
      $newBad \leftarrow CPPairs(TP^{\pi}_{(q_1,q_2)})$
      if $newBad \neq \emptyset$ **then**
        $\mathscr{B} \leftarrow \mathscr{B} \cup newBad$
        ▷ unambiguous states of the found critical path give bad pairs
      else
        $\mathscr{G} \leftarrow \mathscr{G} \cup StatesOf(TP^{\pi}_{(q_1,q_2)})$
        ▷ all TP states give good pairs
    if $Pairs \setminus \mathscr{G} = \emptyset$ **then**
      return π
      ▷ π is a diagnosability plan
    else
      $\Pi \leftarrow genCandidatePlan(\Pi, \pi, EBS, G, I, g)$
  return $\emptyset$
  ▷ there is no diagnosability plan
procedure GETBESTCANDIDATE($\Pi, \mathscr{B}, \mathscr{G}, g, h$)
  $\Pi \leftarrow sortByCostandGoodness(\Pi, \mathscr{B}, \mathscr{G}, g, h)$
  ▷ those $\rho \in \Pi$ with $(\rho(I) \times \rho(I)) \cap \mathscr{B} \neq \emptyset$ are the last in the sort; others are sorted in first according to cost function g and goodness-based heuristics h that uses $\mathscr{G}$
  $\rho \leftarrow removeTop(\Pi)$
  return ρ
procedure GENCANDIDATEPLAN(Π, π, EBS, G, I, g)
  $CurrentActions \leftarrow Actions(G)$; $new\Pi \leftarrow \emptyset$
  while $CurrentActions \neq \emptyset$ **do**
    $a \leftarrow removeOne(CurrentActions)$
    $\rho \leftarrow \pi a$
    if $\nexists BS \in EBS \mid BS \subseteq \rho(I)$ **then**
      ▷ if $\rho(I)$ contains an already explored belief state, the node ρ is closed
      $new\Pi \leftarrow new\Pi \cup \{\rho\}$
  while $new\Pi \neq \emptyset$ **do**
    $\rho \leftarrow removeOne(new\Pi)$
    if $\nexists \rho' \in new\Pi \mid \rho'(I) \subseteq \rho(I) \wedge g(\rho') \leq g(\rho)$ **then**
      ▷ if a plan candidate ρ' is better than the plan candidate ρ, then ρ is closed
      $\Pi \leftarrow \Pi \cup \{\rho\}$
  return Π

the stopping criterion for the algorithm, meaning the nonexistence of a diagnosability plan. Then, the procedure *getBestCandidate* ranks the candidate plans according to their costs (i.e., Breadth First Search for example if all elementary actions have the same cost) and optionally favoring also other heuristic criteria, like here the goodness (ratio of good pairs in the belief state), and ranking in last those whose belief space contains a bad pair. It returns the best candidate plan π for this ranking, which gives the current belief state $\pi(I)$ to be tested for diagnosability. Then if $\pi(I)$ does not contain any known bad pair and contains at least one non-tagged pair, iteration is done on these *non-tagged* pairs of states in $\pi(I)$ in order to check their goodness. That is, a pair $\{q_1, q_2\}$ is chosen and (part of, in the light of known tagged pairs) the Twin Plant $TP^{\pi}_{(q_1,q_2)}$ of the subsystem $G_{\{q_1,q_2\}}$ (having q_1 and q_2 as initial states) is built. If a critical path (which may be predicted just by meeting a bad pair) is found in $TP^{\pi}_{(q_1,q_2)}$, all the pairs of states corresponding to the new non-ambiguous states of this critical path (so, at least $\{q_1, q_2\}$) are added to the bad pairs list. In case $TP^{\pi}_{(q_1,q_2)}$ does not contain any critical path, all pairs represented by its states are learned as good pairs and the iteration continues. The iteration stops either because all pairs of $\pi(I) \times \pi(I)$ are found to be good, and then π is a diagnosability plan (necessarily optimal for the cost if no heuristics has been used), which ends the algorithm, or because a bad pair is found in $\pi(I)$. In this last case, the plan π is failed and its candidate successors are generated by the procedure *genCandidatePlan* and the next best candidate plan is chosen by the procedure *getBestCandidate*. So the algorithm always terminates and returns a diagnosability plan (guaranteed optimal if h is not used) if it exists.

6.4.4 Illustrative Example

We search an applicable sequence of actions that leads the system depicted in figure 6.1 to a diagnosable belief state. When we apply the algorithm 1 (*Eager Learning method* with equal elementary actions costs and without heuristics h, thus in BFS), we get the Figure 6.2 that shows the evolution of the candidate plans and their corresponding belief states before reaching the intended plan ($\pi = \alpha\beta\beta$) that leads the system to the diagnosable state $\pi(I) = \{1n, 3n\}$ (red node $N9$). In this figure, the blue node $N5$ represents a closed node as its belief state is equal to that of the previously explored node $N2$. Other explored nodes are not diagnosable, moreover the diagnosability test is not called for any node that contains at least one known bad pair and not called on the known good pairs of the belief state of any node. This is why the orange nodes $N3, N4, N6, N8$ and the red node $N9$ do not need to be tested according to the following scenario illustrated by the figure 6.1. We assume that, when testing $N1$ (i.e., the empty plan), the good pairs $\{1n, 1n\}$, $\{3n, 3n\}$ and $\{1n, 3n\}$ are found first before discovering the bad pair $\{1n, 2n\}$. And that, when testing $N2$, the bad

pair $\{3n, 4n\}$ is found. Then, $N3$ and $N4$ fail without having to be tested as their belief spaces contain $\{1n, 2n\}$. Later, it is the same for $N6$ whose belief space contains $\{3n, 4n\}$. When testing $N7$, the bad pair $\{4n, 5n\}$ is found. Again, $N8$ fails without having to be tested as its belief space contains $\{3n, 4n\}$. Finally $N9$ succeeds and provides the optimal diagnosability plan $\alpha\beta\beta$ without having to be tested as the three possible pairs of states issued from its belief state are all good pairs from the result of $N1$ test. Using the heuristics h based on the ratio of known good pairs in a belief state would allow in general better pruning of the plan search space (for example here, from the knowledge of the good pairs found in $N1$ to go directly to the solution node $N9$ via $N2$ and $N6$), even if the diagnosability plan found is not guaranteed to be optimal.

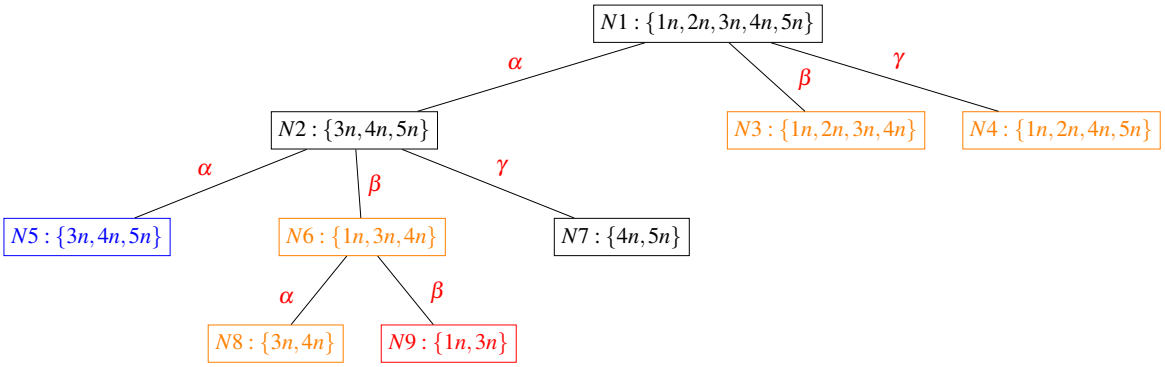

Fig. 6.2 Plan search space, with $N9$ diagnosable, $N5$ closed and other nodes not diagnosable.

6.5 Experimental Results

In order to test our proposed approaches on a benchmark, we created a rectangular grid of components by repeating the active model (i.e., without its actions) of our running example in figure 6.1 and we reconfigured the actions in all components and added global actions between components. We defined two actions models that are applied to a component according to its position in the grid (given by line index i and column index j, with the origin at the left top corner). The first one is adopted for all top and bottom border components and allows the planner, for an initial belief state chosen in one of these components, to find a short plan, of length two or three, made up of local actions inside this component. The second one is adopted for all other (internal) components and does not allow reaching a diagnosable belief state inside this component, but allows moving from this component to the one just below or above it by using global actions that connect the actions models of any internal component and those of its two neighbors on the same column. We designed the diagnosability plan to be, starting from any internal component belief state, the one obtained

by moving to its below neighbors until reaching the bottom line of the grid. Moving upward is also always an option for the planner but will not give a diagnosable plan. The tested benchmark does not contain any horizontal action between components in different columns. In order to connect actively by event transitions each component with its existing (at most four) neighbors, we have added an observable communication event c that connects each state $3n$ (resp. $1n$) at the position (i, j) in the grid to the corresponding state $2n$ (resp. $4n$) at the position $(i+1, j)$. Similarly, we connect faulty state $3f$ (resp. $4f$) at the position (i, j) to $1f$ (resp. $2f$) at the position $(i, j+1)$, and faulty state $1f$ (resp. $2f$) at the position (i, j) to $3f$ (resp. $4f$) at the position $(i, j-1)$.

We have tested three search algorithms of the intended diagnosability plan. The first algorithm uses the *Normal method*, i.e. without any recycling. The second algorithm uses the *Lazy Learning method* which concentrates on learning only the bad pairs while not being able to exploit any good pair. The difference with the first algorithm is that this one learns about bad pairs and later uses them to prune another plan test or to stop another Twin Plant construction if it meets such a pair again. The third algorithm uses the *Eager Learning method* and is the one described in Algorithm 1.

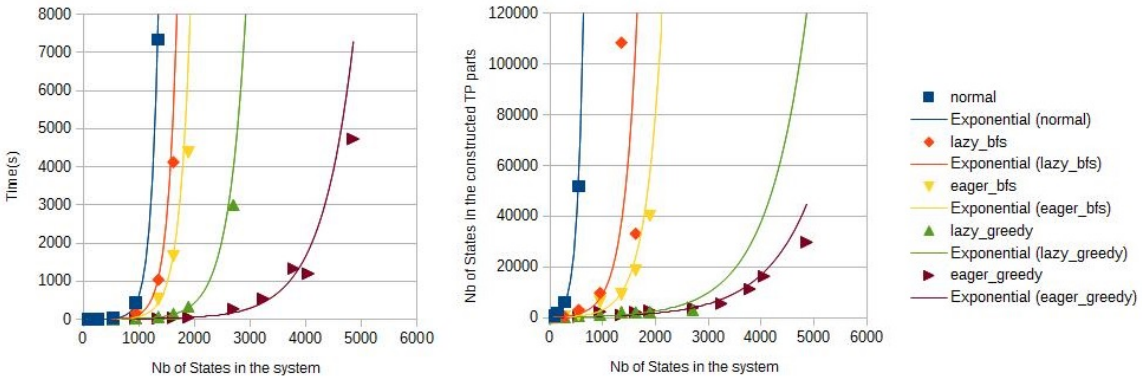

Fig. 6.3 I contains the normal states of one central component.

Concerning the search strategy, firstly we applied to the planner Breadth First Search strategy (i.e., without h and with equal elementary actions costs in g) which guarantees optimal length of the plan, and tested the three methods. Secondly we applied our greedy strategy, limited here to the use of the heuristic function h (without g), computed from the learned pairs and aiming at maximizing the percentage of "goodness" of a belief state reached by each candidate plan. This approach is applicable to the third method, the only one to have the information about good pairs; we adapt it to the second method by minimizing the number of non-tagged pairs, i.e. the size of the belief space.

Test Id	**Matrix**	$\|\mathscr{B}\|$	$\|\mathscr{G}\|$	**|TPs|**	**|tested π|**	$\|\pi\|$	**Time**
1N	3x3	0	0	898	38	5	0s
1L	3x3	22	3	204	18	5	0s
1E	3x3	8	6	59	17	5	0s
2N	5x3	0	0	1980	99	8	1s
2L	5x3	36	3	231	29	8	0s
2E	5x3	12	6	128	20	8	0s
3N	10x3	0	0	6127	335	14	3s
3L	10x3	113	3	444	101	14	1s
3E	10x3	33	24	612	95	14	1s
4N	20x3	0	0	51846	2350	29	27s
4L	20x3	994	3	3093	950	29	12s
4E	20x3	89	60	1227	554	29	8s
5N	50x3	0	0	1046545	38259	74	2h2m26s
5L	50x3	11336	3	108466	11252	74	17m17s
5E	50x3	219	156	9207	4320	74	8m45s

Table 6.1 The three methods tested on different grid heights (with width 3) using BFS, where I is made up of the normal states of the central component.

Test Id	**Matrix**	$\|\mathscr{B}\|$	$\|\mathscr{G}\|$	**|TPs|**	**|tested π|**	$\|\pi\|$	**Time**
1N	3x5	0	0	7385	148	6	2s
1L	3x5	64	9	1619	56	6	1s
1E	3x5	9	12	148	24	6	0s
2N	5x5	0	0	40207	806	11	10s
2L	5x5	214	6	4429	199	11	4s
2E	5x5	25	15	529	133	11	1s
3N	10x5	0	0	325385	7118	20	1m57s
3L	10x5	2048	6	32820	2019	20	56s
3E	10x5	63	57	7518	947	20	55s
4N	15x5	0	0	3238594	43570	29	2h9m26s
4L	15x5	12087	6	4653	12051	29	20m6s
4E	15x5	105	96	56323	4260	29	10m6s
5L	17x5	34597	6	13449	34533	35	4h45m20s
5E	17x5	119	108	165193	13691	35	1h21m1s

Table 6.2 The three methods tested on different grid heights (with width 5) using BFS, where I is made up of the normal states of the components at $(|lines|/3,1)$ and $(2|lines|/3,3)$.

Test Id	**Matrix**	$\|\mathscr{B}\|$	$\|\mathscr{G}\|$	**|TPs|**	**|tested π|**	$\|\pi\|$	**Time**
1L	3x3	17	3	93	14	5	0s
1E	3x3	5	3	35	6	5	0s
2L	5x3	22	3	119	16	8	0s
2E	5x3	10	6	101	12	9	0s
3L	10x3	63	3	277	50	14	1s
3E	10x3	25	21	362	39	16	1s
4L	20x3	390	3	617	353	29	5s
4E	20x3	57	48	453	149	36	3s
5L	50x3	2406	3	2255	2309	74	1m3s
5E	50x3	146	141	1119	661	96	22s

Table 6.3 The two learning methods tested on different grid heights (with width 3) using the greedy heuristics, where I is made up of the normal states of the central component.

Test Id	**Matrix**	$\|\mathscr{B}\|$	$\|\mathscr{G}\|$	**|TPs|**	**|tested π|**	$\|\pi\|$	**Time**
1L	3x5	18	6	300	12	6	0s
1E	3x5	4	6	47	7	6	0s
2L	5x5	44	6	743	33	12	1s
2E	5x5	14	9	189	24	13	0s
3L	10x5	151	6	1166	133	21	3s
3E	10x5	47	54	1678	228	24	6s
4L	15x5	285	6	1634	257	30	7s
4E	15x5	52	66	1156	112	35	3s
5L	17x5	689	6	2039	653	36	18s
5E	17x5	57	75	2381	205	40	5s
6L	20x5	1005	6	2045	951	42	25s
6E	20x5	84	87	1784	215	46	5s
7L	50x5	10725	6	26226	10558	102	49m6s
7E	50x5	160	258	13413	1613	116	54s
8E	100x5	488	498	87236	15413	243	1h35m58s

Table 6.4 The two learning methods tested on different grid heights (with width 5) using the greedy heuristics, where I is made up of the normal states of the components at $(|lines|/3,1)$ and $(2|lines|/3,3)$.

Test Id	**|Comps. of belief |**	$\|\mathscr{B}\|$	$\|\mathscr{G}\|$	**|TPs |**	**|tested π |**	$\|\pi\|$	**Time**
1N	2	0	0	153473	4904	23	1m5s
1L	2	1500	12	15916	1463	23	37s
1E	2	44	49	1384	562	23	12s
2N	3	0	0	423034	6652	27	2m14s
2L	3	1446	14	36424	1400	27	52s
2E	3	60	81	6121	750	35	36s
3N	4	0	0	594293	6254	28	2m47s
3L	4	1230	17	48385	1177	28	52s
3E	4	75	113	18524	1332	35	64s
3N	5	0	0	774272	6174	29	3m52s
3L	5	1470	20	62346	1413	29	1m1s
3E	5	74	134	15015	833	35	46s
3N	6	0	0	692229	5181	28	3m36s
3L	6	1212	23	55982	1156	28	54s
3E	6	73	139	7721	414	35	23s
4N	7	0	0	821782	5601	28	5m26s
4L	7	967	26	49440	909	28	1m
4E	7	74	144	4788	358	30	21s
5N	8	0	0	1002066	6333	28	8m1s
5L	8	1344	29	46194	1289	28	1m21s
5E	8	67	130	2795	267	29	17s
9N	9	0	0	1331933	8263	28	14m50s
9L	9	1612	32	62676	1554	28	2m6s
9E	9	56	115	3023	245	29	12s
10E	100	21	121	6340	113	35	18s

Table 6.5 The Normal method and the two learning methods with the greedy heuristics tested on a fixed 10×10 grid, where I is made up of the normal states of an increasing number of components on the diagonal of the grid (and for the last line all components).

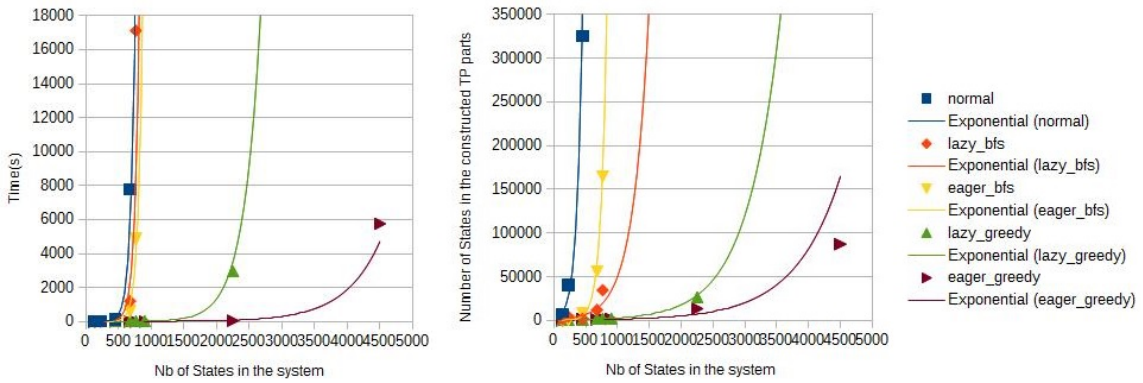

Fig. 6.4 *I* contains the normal states of two scattered internal components.

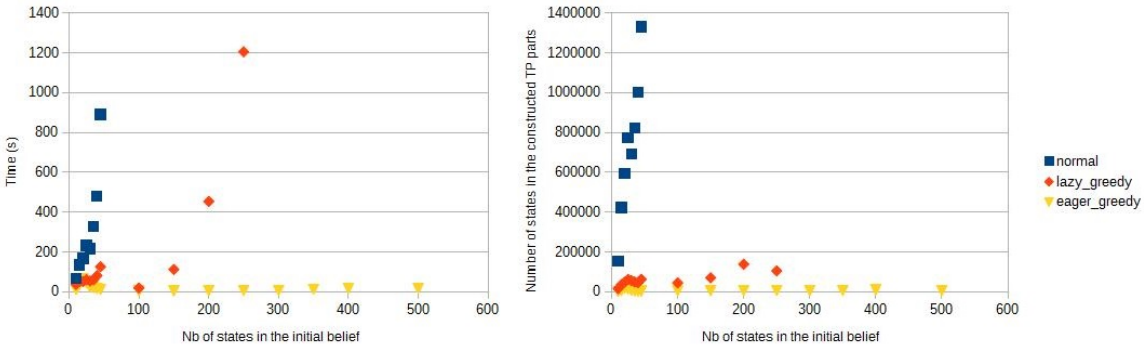

Fig. 6.5 Changing initial belief state size in a fixed system of (10×10) components.

Successive columns in the tables show the test number along with the letter (N, L or E) identifying the method used, the grid size (except for Table 6.5 where it is the number of components on which the initial belief state spreads), the number of learned bad pairs, the number of learned good pairs, the states number of the Twin Plants constructed, the number of tested plans, the size of the diagnosability plan computed and the CPU time.

We have first tested the 5 different methods on increasing grid heights (with width 3) with the initial belief state made up of the five normal states in the central component of the grid. Using a Breadth First Search algorithm for the planner, Table 6.1 shows that the *Eager Learning method* is promising and is 15 times faster for a grid of height 50 (the length of the optimal diagnosability plan being 74) than the classical approach that tests diagnosability without any learning. This shows the efficiency of recycling learned pairs even if they are not exploited in guiding the planning step to solve the problem. But the benefit of this exploitation by the greedy heuristics in the two learning methods appears clearly in Table 6.3, even if the *Eager Learning method* does not give an optimal diagnosability plan (actually it

scales well up to a 180×3 grid with a plan of length 355). Figure 6.3 shows all the five tests on this bench graphically.

The tests shown in Tables 6.2 and 6.4 are more challenging as we suppose an initial belief state composed of the normal states of two scattered internal components. The largest the height of the grid (with width 5), the farthest they are. The search space of the plan is much bigger, but still, even without guiding the plan search by learned pairs, the performances of the two learning methods are better than the performance of the *Normal method* which explodes for the height 17 (see Table 6.2). However, results improve dramatically when exploiting the learned pairs in guiding the planning search as shown in Table 6.4, where the *Eager Learning method* scales well up to a 100×5 grid. However the greedy heuristics used here does not return an optimal length of the intended plan as does BFS strategy. But we can notice that it is very close to the optimal one for the *Lazy Learning method* (which is not the case for the *Eager Learning method*, although a better heuristics could be used according to the studied system structure). Figure 6.4 shows all the five tests on this bench graphically.

In order to show how changing the size of the initial belief state can affect the results, we fixed the size of the system to a 10×10 grid and incrementally increased the size of the initial belief state by adding at each increment i the five normal states of the component at the position (i,i), so up to 45 states. Table 6.5 shows the interest of using learned good pairs during the search for the diagnosability plan, especially in the last lines (belief state spread on 9 components). Then, the *Normal method* explodes and we continued adding randomly normal states from other components to the initial belief state to compare the *Eager* and *Lazy Learning methods*. The first one scales up to the maximum of 500 states (i.e., the normal states of all the 100 components of the grid, representing complete uncertainty on the belief state). These results are clear in figure 6.5 that shows all the three tests on this bench graphically. Another point concerning tests on this bench is that it does not construct long critical paths in terms of states, so learning bad pairs may be faster in other systems.

Although the motivation of our approach is to avoid possibly intrusive active diagnosability test in running systems, this approach is also useful at the design stage. For example, if after the search of a diagnosability plan, either one is found but with an expensive cost and thus not favored, or no one exists, the actions should be reconfigured to achieve it. Therefore in both cases the information about the sets of good and bad pairs can be used to reconfigure the actions in order to get a plan or improve an existing one. This information can be also used as initialization for the sets of bad and good pairs to plan diagnosability of the system in the operation stage.

6.6 Related Works

Traditional approaches of diagnosability analysis return only the information about whether the systems are diagnosable or not, and can do nothing for non-diagnosable systems, except providing a counter-example. Moreover, diagnosability is a quite strong property that is often not satisfied in real systems, which complicates the diagnosis process. As we mentioned in the beginning of this chapter the active diagnosability problem is a more strict version than our diagnosability planning problem and can affect the plan of production of the system, in addition to the difference with our approach that constructs on the demand parts of the Twin Plants and recycle learned knowledge to prune the search of the plan and the next Twin Plants constructions.

On the other hand, planning techniques have been developed over the last several decades, whose idea is to find a plan satisfying the desired goals (e.g. given properties). In [Barbeau *et al.* 1998], a synthesis method was presented that automatically generates controllers on timed transition graphs, where the specification of control requirements is expressed by metric temporal logic (MTL) formulas. Precisely, during searching the space of all possible paths, MTL formulas are verified over these paths to determine points at which controllable actions should be disabled in order to get a conditional plan. However, the authors assumed the full observability, i.e., every state variable can be observed at each step. Considering that planning domains are often partially observable and non-deterministic, new approaches for planning under partial observability, dealing with uncertainty about the state in which the actions will be executed, were proposed in [Bertoli *et al.* 2001, Bertoli *et al.* 2006]. The search space is no longer the set of states of the domain, but its power-set. The problem is addressed by using BDDs, which can be used to represent and efficiently manipulate the power-set of states. However, BDDs are limited to propositional representations. A close work was presented in [Ciré & Botea 2008], where a goal state represented in LTL is calculated and verified on the fly. In their proposed planning model, all transitions are deterministic. However, in our case, the transitions are non-deterministic. Furthermore, their online learning component has, for each step, to construct a Boolean formula that can explain the violation of the considered property, which could be quite complex since different rules have to be applied at the atomic level. While in our case it suffices to check whether the current explored path is a critical path. The work in [Grastien 2015] addressed the self healing as a combination between conformant planning and diagnosis steps to repair the system state without explicitly computing the system belief state. Given the goal states, an optimal plan is computed for a sample of the belief state that leads to a goal state, then the plan is refined depending on the result of a special diagnoser that tries to find a behavior of the system which contradicts the current plan. Once a behavior is found, this can enrich the

sample to recompute a better plan. The process is repeated until the failure of the diagnoser in its mission, which means that the computed plan is correct for all the belief state members. Our work can be seen as a continuation of this work, where the goal states are described by the diagnosability property.

6.7 Conclusion and Future Work

In this chapter, we formally defined the problem of Diagnosability Planning before demonstrating that it is PSPACE-COMPLETE. Then an algorithm was provided to search for an optimal diagnosability plan. This is done by incrementally constructing the Twin Plants for the belief states produced by the candidate plans, during which the set of learned bad and good pairs is updated that helps in pruning next Twin Plants construction and in defining an heuristic function to guide the plan search. Experimental results demonstrate the efficiency of this approach by exploiting the different learning strategies. Considering more informative heuristic functions is our current work. Similar approaches are also possible for planning other properties that use the Twin Plant structure in their verification, like predictability for example. In a further step, a probabilistic version of this problem will be studied. Another line of research, which has already been started, is to encode the Diagnosability Planning problem into SAT and compare the new experimental results with the current ones.

Chapter 7

Conclusion

7.1 Thesis overview

We have presented in the first part of this thesis our study of the diagnosability and predictability properties in the centralized and distributed discrete event systems using the SAT solver technology. One can informally reduce these properties to asking questions of the form: given a sufficient set of observations, can we tell with certainty that a specific unobservable event has occurred (so it is diagnosable)? or will occur later (so it is predictable)?

7.1.1 Diagnosability

We showed that when diagnosability property is violated, our SAT-based approach can answer such question efficiently, i.e., with a NO answer even for systems with very large number of states. Actually it is equivalent to satisfying the proposed SAT formula and it returns a possible satisfying assignment, i.e. a counter-example to diagnosability property. This assignment can be translated into a pair of infinite trajectories in the system, synchronized on their observations and with only one of them containing the fault. Such information can be used by the system designer to update the sensors placement and re-test the SAT formula until it is UNSAT in all the reachable space, proving thus diagnosability. This performance of the SAT solver is due to the fact that, contrary to the traditional approaches, it avoids building a Twin Plant structure to answer the question. Thus it exploits its freedom to start assigning randomly the different variables aiming at satisfying the formula. In fact its choices are not really random as many heuristics are employed to choose the best branching variable during the search. The important virtue of the modern SAT solvers is that they learn from their "bad" choices! Actually they can do more and even learn form some partial assignments so they are caching their "good" choices as well. Indeed, they are powered with several

heuristics and data structures that allow them to learn new constraints efficiently. Another very important factor to this efficiency is the succinct transition system modeling which allows firing simultaneously non interfering events, an obvious example is between two sets of local events in two different components, thus the counter-example search is reduced to searching in each component separately and only synchronizing their shared events.

From another side, answering this diagnosability question with YES is more difficult using our approach. It is equivalent to returning an UNSAT answer of the SAT formula after browsing *all possibilities* of satisfying assignments in all the search space. Especially that we are not adding currently any heuristic on how to search this space to check the satisfiability of the formula, i.e., we are using a brute force SAT solving. Actually we can observe a similar effect even if the answer is NO (SAT) but with a long counter-example. This weakness is already known about SAT-based approaches, e.g., in model checking. We proposed the usage of incremental SAT mode to mitigate this problem. Thus it allows exploiting the learned knowledge about the system behavior in one call during its next calls, so we adopted our encoding to this mode. Our incremental SAT-based approach improved the CPU time by more than 25% in the centralized structure in comparison with non-incremental approach and by more than 10% in the distributed case. These improvements are despite the fact that we have done the tests on an artificial benchmark, which we created on purpose to have a simple and sparse behavior of the system, i.e., that there are not much constraints describing the behavior; so less constraints can be learned and exploited in the incremental mode than for a denser example. From another side, the communication events in this benchmark are designed only to increase the overhead of communications, thus the critical pair does not contain any other transitions from these neighbors than communication ones. This means also that during the incremental SAT calls there will be a few amount of new information to learn about the behaviors in the global system. In other words we expect larger gain on a real benchmark. We present in the future works section two propositions that would improve this incremental SAT-based approach.

Concerning the communication events, we exploited the flexibility of the propositional logic formalism to handle a mixed set of observable and unobservable communication events simultaneously. Our encoding of these events allows one to push the synchronization process overhead to the SAT solver which facilitates encoding the components separately.

7.1.2 Predictability

Regarding the predictability property, which is a stronger property than diagnosability, we have encoded its verification as a SAT problem using the same formalism of succinct transition systems. The main difference with the diagnosability verification is the way to

synchronize the observations. Thus predictability test requires observations synchronization only before the fault occurrence and checking the non-finiteness only for the correct trajectory. In order to test this property we encoded the fault occurrence in an incremental way allowing us to determine symbolically when a fault occurrence takes place in order to control the synchronization of observations. Here also we got results similar to the diagnosability tests, i.e., scalability of the size of non-predictable systems and lower efficiency if the fault is predictable. In comparison with non-diagnosability tests we can notice that non-predictability is faster, in terms of needed number of steps, to be proven. This is because any model that proves non-diagnosability of a fault can be cast to represent a witness of non-predictability, which is consistent with the fact that predictability is stronger than diagnosability. Another observation from our results is that the incremental encoding of the fault occurrence improved dramatically the efficiency for large formulas. Finally, the flexibility of our communication events encoding allowed us to handle the case of observable communication events by adding some logical constraints, when the traditional approaches do not deal with this case. Thus, an observable communication event must be synchronized before the fault qua observable and communication event while only qua communication event after the fault occurrence. This would not be so straightforward to handle in the predictor structure and would require special adaptation for such cases.

7.1.3 Pattern Diagnosability

We generalized our formalism to handle diagnosability of pattern of events. This pattern may represent a faulty sequence or just important sequences of events which we want to be able to detect with certainty in a finite number of system's steps after its occurrence. This pattern is represented by a complete deterministic finite state machine with a stable set of final states. Thus we can study the diagnosability of any formal extension-closed rational language of events. We proposed to encode it as a succinct transition system just like any other component of the system and to synchronize it with the concerned components. We introduced the encoding in SAT but we have not yet tested on examples what we also implemented. The motivation to handle this problem in SAT is similar to above cases where we seek the scalability and we expect a similar result also. Thus this problem was issued in the literature because of the cost of building the whole diagnoser structure before testing the property which contains an important overhead of synchronization. In order to reduce the size of our encoding, we restricted the synchronization between the concerned component events and the pattern events by considering only significant events in the pattern for this synchronization, i.e., those events that change at least one of the pattern states.

7.1.4 Diagnosability Planning

In a controllable discrete event system where the system is equipped with a set of elementary actions, we studied the problem of restoring the diagnosability of the system through reducing it to a planning problem. Thus we introduced formally the diagnosability planning problem which consists in finding a plan (a sequence of these actions) that, when applied from a given uncertain belief state of the system, guarantees that the system is surely in a diagnosable state and can be left to run freely. We introduced this problem motivated by several needs like ensuring diagnosability from a potential current belief state of the system (after an accidental intervention or a repair plan), reducing the diagnostic interaction with a running system (for costs or quality reasons), mitigating the high complexity (EXPTIME) of building an active diagnoser. Then we analyzed the problem and proved that its decision problem is PSPACE-COMPLETE. Then we have used a similar spirit that exists in the SAT solvers, i.e., building on-the-demand structure and exploiting previous tries, by proposing three methods which exploit these ideas differently. Actually the first method does not use any of them at all so it always builds blindly a complete diagnosis structure and does not learn from this construction, the second one learns from conflicts so just like the SAT solvers, the third one exploits not only its failed tries but also its partially successful tries to construct the intended diagnosable belief state, thus it recycles the constructed parts of the Twin Plants from the different possible states.

Roughly speaking, we have studied the event diagnosability, event predictability, pattern diagnosability in centralized and distributed discrete event systems and introduced the diagnosability planning problem using on-the-fly structures and we provided experimental results about all of them which is not common when studying these properties.

7.2 Future Works

Several extensions to our work presented in this thesis are still under development, some of them are formally encoded in SAT but not yet implemented and others are implemented but not yet tested, and many others have not yet been investigated. We will mention them here.

- Testing the implemented encoding of pattern diagnosability on a benchmark.
- Implementing our incremental SAT encoding for the predictability checking.
- Implementing a general benchmark generator. We have designed a general algorithm to generate benchmarks with several parameters in order to test our encoding and also make it available for the research community, as it happens that, 20 years after the

introduction of the diagnosability problem, there is still no general benchmark available for comparing scientific results experimentally. The parameters that we want to pass to the algorithm contain: number of states, maximum output degree, observability ratio, density of transitions, faulty sequences ratio, distance of the first fault from the initial state, whether the fault is diagnosable or not, cycle length in a critical path in case of non-diagnosability, ratio of communication events, their observability ratio, number of components in the system. We have constructed a first draft of the algorithm and started its implementation but we faced some problems in managing the observable cycles in diagnosable instances.

- Taking into account the specificity of our problem by adding heuristics that favor decision firstly on the most informative variables related to our problem should improve the results. Such variables here are the fault occurrences and the reference variables that are responsible of the synchronization of observable events and of communication events.

- Testing other SAT solvers can be also interesting, in particular circuit-based SAT solvers as they can be more efficient in verifying the synchronization formulas.

- Using more efficiently control hypotheses to guide the SAT-based search. We know that in both diagnosability and predictability analyses the fault occurrence is essential to do the verification, thus a cycle detection is not important before the fault, so we think searching the cycles can be "delayed" until the fault is detected. However, in our encoding of the incremental SAT approach we adopted only one hypothesis to control the two constraints. We think that using two control hypotheses would allow us to deactivate the cycle detection search until the first fault occurrence is found then start the search for the cycle detection. Maybe we can tell the solver to test at each step firstly the fault occurrence and continue (without stopping the search) in case of positive answer else stop. It could be something just like, when all clauses are satisfied, adding a new set of clauses to be satisfied rather than answering SAT directly.

- Exploring bounded versions of diagnosability and predictability, which we have not done in this thesis but that can also be dealt with SAT solvers. Those properties are stronger than the unbounded version discussed here, and they very important in an industrial context. Thus in diagnosability analysis one would want to know how many observable events should have to be waited for in order to decide if the fault has occurred or not. Similarly, in predictability analysis the time interval where the fault will occur can be interesting to know, thus the interval predictability versions can

answer this need (a recent work has been done about this but without using SAT in the verification).

- Solving the diagnosability planning problem with SAT solvers. Actually, planning problems have been translated to SAT problems since [Kautz & Selman 1992] but only when they are defined with a set of explicit goal states. Thus producing a partial or full SAT-based version of diagnosability planning is one of our short term future works as we have already encoded the recycling of bad pairs in SAT, although this is not presented in this thesis.

- Searching a diagnosability plan under additional constraints like being of length smaller than K, introducing probabilities on the fault occurrences, favoring special goal belief states in this light can be also other interesting work to do.

- Studying planning of other properties is also an interesting future work, in particular those based on Twin Plant like structures like predictability, as in general planning problems are proposed regarding to explicit goal states and not regarding to a formal property.

- Studying such properties planning for Distributed DES.

- Dealing with distributed observations. We look forward to investigate the case of distributed observations among autonomous components. Although our representation can be easily extended to deal with local observations (i.e., observable events in one component are observed only by this component), we know that in general diagnosability checking becomes then undecidable, e.g., when communication events are unobservable (obviously it remains decidable when communication events are observable in all their owners) [Ye & Dague 2013]. A future work will be to study decidable cases of diagnosability checking in DDES with local observations, e.g., assuming some well chosen communication events being observable or at least diagnosable in some context!

- Studying an optimal sensor placement as a MAXSAT problem is also in our todo list, this is an important problem that consists in arranging observation in the system so the system is diagnosable with the minimum number of observations being measured from it.

- Studying diagnosability as an information problem, using the logical entropy, can give us ideas on how systems can be diagnosable and maybe give optimal ways to test this diagnosability and can be also useful to build our random benchmark generator.

All these above extensions and many others are interesting to be explored. However being always a human I tend to control my work, discretize it and do these extensions incrementally, so I will stop here, hoping that humans will always hold the control!

References

[Audemard & Simon 2009] Gilles Audemard et Laurent Simon. *Predicting Learnt Clauses Quality in Modern SAT Solvers.* In Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09), pages 399–404, 2009.

[Barbeau *et al.* 1998] M. Barbeau, F. Kabanza et R. St-Denis. *A Method for the Synthesis of Controllers to Handle Safety, Liveness, and Real-Time Constraints*. IEEE Transactions on Automatic Control (TAC), vol. 43, no. 11, pages 1543–1559, 1998.

[Bavishi & Chong 1994] Sanjiv Bavishi et Edwin KP Chong. *Automated fault diagnosis using a discrete event systems framework*. In Proceedings of the 1994 IEEE International Symposium on Intelligent Control, pages 213–218, 1994.

[Bertoli *et al.* 2001] P. Bertoli, A. Cimatti, M. Roveri et P. Traverso. *Planning in Nondeterministic Domains Under Partial Observability via Symbolic Model Checking*. In Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01), pages 473–478, 2001.

[Bertoli *et al.* 2006] P. Bertoli, A. Cimatti, M. Roveri et P. Traverso. *Strong Planning Under Partial Observability*. Artificial Intelligence, vol. 170, no. 4-5, pages 337–384, 2006.

[Biere *et al.* 1999] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Masahiro Fujita et Yunshan Zhu. *Symbolic model checking using SAT procedures instead of BDDs*. In Proceedings of the 36th annual ACM/IEEE Design Automation Conference, pages 317–320, 1999.

[Bjesse *et al.* 2001] Per Bjesse, Tim Leonard et Abdel Mokkedem. *Finding bugs in an alpha microprocessor using satisfiability solvers*. In Proceedings of the 13th Conference on Computer Aided Verification (CAV'01), pages 454–464, 2001.

[Boussif *et al.* 2015] Abderraouf Boussif, Mohamed Ghazel et Kais Klai. *Combining Enumerative and Symbolic Techniques for Diagnosis of Discrete-Event Systems*. In Proceedings of the 9th International Workshop on Verification and Evaluation of Computer and Communication Systems (VECoS'15), page 23, 2015.

[Briones *et al.* 2008] Laura Brandán Briones, Alexander Lazovik et Philippe Dague. *Optimizing the system observability level for diagnosability*. In Proceedings of the 3rd International Symposium On Leveraging Applications of Formal Methods, Verification and Validation (ISoLA'08), pages 815–830, 2008.

[Buss *et al.* 1990] Samuel R. Buss, Christos Papadimitriou et John Tsitsiklis. *On the predictability of coupled automata: An allegory about Chaos*. In Proceedings of the 31st Annual Symposium on Foundations of Computer Science (FOCS'90), pages 788–793, 1990.

[Bylander 1994] T. Bylander. *The computational complexity of propositional STRIPS planning*. Artificial Intelligence, vol. 69, no. 1, pages 165–204, 1994.

[Cao 1989] X-R Cao. *The predictability of discrete event systems*. IEEE Transactions on Automatic Control, vol. 34, no. 11, pages 1168–1171, 1989.

[Cassandras & Lafortune 2009] Christos G Cassandras et Stephane Lafortune. Introduction to discrete event systems. Springer Science & Business Media, 2009.

[Chanthery & Pencolé 2009] Elodie Chanthery et Yannick Pencolé. *Monitoring and active diagnosis for discrete-event systems*. Proceedings of the 7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (Safeprocess'09), vol. 42, no. 8, pages 1545–1550, 2009.

[Ciré & Botea 2008] A. Ciré et A. Botea. *Learning in Planning with Temporally Extended Goals and Uncontrollable Events.* In Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008), pages 578–582, 2008.

[Cook 1971] Stephen A Cook. *The complexity of theorem-proving procedures*. In Proceedings of the third annual ACM symposium on Theory of computing, pages 151–158, 1971.

[Davis & Putnam 1960] Martin Davis et Hilary Putnam. *A computing procedure for quantification theory*. Journal of the ACM, vol. 7, no. 3, pages 201–215, 1960.

[Davis *et al.* 1962] Martin Davis, George Logemann et Donald Loveland. *A machine program for theorem-proving*. Communications of the ACM, vol. 5, no. 7, pages 394–397, 1962.

[Debouk *et al.* 2000] Rami Debouk, Stéphane Lafortune et Demosthenis Teneketzis. *Coordinated decentralized protocols for failure diagnosis of discrete event systems*. Discrete Event Dynamic Systems, vol. 10, no. 1-2, pages 33–86, 2000.

[Eén & Biere 2005] Niklas Eén et Armin Biere. *Effective preprocessing in SAT through variable and clause elimination*. In Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT'05), pages 61–75. Springer, 2005.

[Eén & Sörensson 2005] Niklas Eén et Niklas Sörensson. *MiniSat v1.13 – A SAT Solver with Conflict-Clause Minimization*. In Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT'05), 2005.

[Fadel & Holloway 1999] Hisham Khalil Fadel et Lawrence E Holloway. *Using SPC and template monitoring method for fault detection and prediction in discrete event manufacturing systems*. In Proceedings of the 1999 IEEE International Symposium on Intelligent Control/Intelligent Systems and Semiotics, pages 150–155, 1999.

[Fourdrinoy *et al.* 2007] Olivier Fourdrinoy, Eric Grégoire, Bertrand Mazure et Lakhdar Saïs. *Reducing hard SAT instances to polynomial ones.* In Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI'07), pages 18–23, 2007.

[Genc & Lafortune 2006a] S. Genc et S. Lafortune. *Diagnosis of patterns in partially-observed discrete-event systems*. In Proceedings of the 45th IEEE Conference on Decision and Control (CDC'06), pages 422–427, 2006.

[Genc & Lafortune 2006b] Sahika Genc et Stéphane Lafortune. *PREDICTABILITY IN DISCRETE-EVENT SYSTEMS UNDER PARTIAL OBSERVATION 1*. IFAC Proceedings Volumes, vol. 39, no. 13, pages 1461–1466, 2006.

[Genc & Lafortune 2009] Sahika Genc et Stéphane Lafortune. *Predictability of event occurrences in partially-observed discrete-event systems*. Automatica, vol. 45, no. 2, pages 301–311, 2009.

[Gomes *et al.* 2000] Carla P Gomes, Bart Selman, Nuno Crato et Henry Kautz. *Heavy-tailed phenomena in satisfiability and constraint satisfaction problems*. Journal of automated reasoning, vol. 24, no. 1-2, pages 67–100, 2000.

[Grastien *et al.* 2007] Alban Grastien, Anbu Anbulagan, Jussi Rintanen et Elena Kelareva. *Diagnosis of discrete-event systems using satisfiability algorithms*. In Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI'07), pages 305–310, 2007.

[Grastien 2015] Alban Grastien. *Self-healing as a combination of consistency checks and conformant planning problems*. In Proceedings of the 26th International Workshop on Principles of Diagnosis (DX'15), pages 105–112, 2015.

[Haar *et al.* 2013] Stefan Haar, Serge Haddad, Tarek Melliti et Stefan Schwoon. *Optimal constructions for active diagnosis*. In Proceedings of the 33rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'13), pages 527–539, 2013.

[Haddad *et al.* 2004] Serge Haddad, Jean-Michel Ilié et Kais Klai. *Design and evaluation of a symbolic and abstraction-based model checker*. In Proceedings of the 2nd International Symposium on Automated Technology for Verification and Analysis (ATVA'04), pages 196–210, 2004.

[Haim & Heule 2014] Shai Haim et Marijn Heule. *Towards ultra rapid restarts*. arXiv preprint arXiv:1402.4413, 2014.

[Huang 2007] Jinbo Huang. *The Effect of Restarts on the Efficiency of Clause Learning*. In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07), volume 7, pages 2318–2323, 2007.

[Jéron *et al.* 2006] T. Jéron, H. Marchand, S. Pinchinat et M.-O. Cordier. *Supervision Patterns in Discrete Event Systems Diagnosis*. In Proceedings of the 8th International Workshop on Discrete Event Systems (WODES'06), 2006.

[Jiang & Kumar 2004] Shengbing Jiang et Ratnesh Kumar. *Failure diagnosis of discrete-event systems with linear-time temporal logic specifications*. IEEE Transactions on Automatic Control, vol. 49, no. 6, pages 934–945, 2004.

[Jiang *et al.* 2001] S. Jiang, Z. Huang, V. Chandra et R. Kumar. *A polynomial algorithm for testing diagnosability of discrete-event systems.* IEEE Transactions on Automatic Control, vol. 46, no. 8, pages 1318–1321, 2001.

[Kautz & Selman 1992] H. Kautz et B. Selman. *Planning as Satisfiability*. In Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92), pages 359–363, 1992.

[Le Berre & Parrain 2010] D. Le Berre et A. Parrain. *The Sat4j library, release 2.2*. Journal on Satisfiability, Boolean Modeling and Computation, vol. 7, pages 59–64, 2010.

[Liu *et al.* 2014] Baisi Liu, Mohamed Ghazel et Armand Toguyéni. *Toward an efficient approach for diagnosability analysis of DES modeled by labeled Petri nets*. In Proceedings of the 13th European Control Conference (ECC'14), pages 1293–1298, 2014.

[Mé & Michel 2001] Ludovic Mé et Cédric Michel. *Intrusion detection: A bibliography*. Tech. Rep. SSIR-2001-01, Supélec, Rennes, France, 2001.

[Moskewicz *et al.* 2001] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang et Sharad Malik. *Chaff: Engineering an efficient SAT solver*. In Proceedings of the 38th annual Design Automation Conference, pages 530–535, 2001.

[Nadel & Ryvchin 2012] A. Nadel et V. Ryvchin. *Efficient SAT Solving under Assumptions*. In Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing (SAT'12), 2012.

[Pencolé 2004] Y. Pencolé. *Diagnosability Analysis of Distributed Discrete Event Systems.* In Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04), 2004.

[Ribot *et al.* 2007] Pauline Ribot, Yannick Pencolé et Michel Combacau. *Characterization of requirements and costs for the diagnosability of distributed discrete event systems*. In Proceedings of the 5th Workshop on Advanced Control and Diagnosis (ACD'07), Grenoble, 2007.

[Rintanen & Grastien 2007] J. Rintanen et A. Grastien. *Diagnosability testing with satisfiability algorithms.* In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07), pages 532–537, 2007.

[Rintanen 2007] Jussi Rintanen. *Diagnosers and Diagnosability of Succinct Transition Systems.* In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07), pages 538–544, 2007.

[Sampath *et al.* 1995] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen et D. Teneketzis. *Diagnosability of discrete-event systems.* IEEE Transactions on Automatic Control, vol. 40, no. 9, pages 1555–1575, 1995.

[Sampath *et al.* 1996] Meera Sampath, Raja Sengupta, Stephane Lafortune, Kasim Sinnamohideen et Demosthenis Teneketzis. *Failure diagnosis using discrete-event models*. IEEE transactions on control systems technology, vol. 4, no. 2, pages 105–124, 1996.

[Sampath *et al.* 1998] Meera Sampath, Stephane Lafortune et Demosthenis Teneketzis. *Active diagnosis of discrete-event systems*. IEEE Transactions on Automatic Control, vol. 43, no. 7, pages 908–929, 1998.

[Schumann & Pencolé 2007] Anika Schumann et Yannick Pencolé. *Scalable Diagnosability Checking of Event-Driven Systems.* In Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07), volume 7, pages 575–580, 2007.

[Silva & Sakallah 1997] João P Marques Silva et Karem A Sakallah. *GRASP—a new search algorithm for satisfiability*. In Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design, pages 220–227, 1997.

[Tseitin 1970] Gregory Tseitin. *On the complexity of proofs in propositional logics*. In Seminars in Mathematics, volume 8, pages 466–483, 1970.

[Ushio *et al.* 1998] Toshimitsu Ushio, Isao Onishi et Koji Okuda. *Fault detection based on Petri net models with faulty behaviors*. In Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, volume 1, pages 113–118, 1998.

[Velev & Bryant 2003] Miroslav N Velev et Randal E Bryant. *Effective use of Boolean satisfiability procedures in the formal verification of superscalar and vliw microprocessors*. Journal of Symbolic Computation, vol. 35, no. 2, pages 73–106, 2003.

[Yan 2004] Yuhong Yan. *Sensor placement and diagnosability analysis at design stage*. In MONET Workshop on Model-Based Systems (ECAI'04), 2004.

[Ye & Dague 2010] L. Ye et P. Dague. *An optimized algorithm for diagnosability of component-based systems*. In Proceedings of the 10th International Workshop on Discrete Event Systems (WODES'10), 2010.

[Ye & Dague 2012] L. Ye et P. Dague. *A general algorithm for pattern diagnosability of distributed discrete event systems*. In Proceedings of the 24th International Conference on Tools with Artificial Intelligence (ICTAI'12), 2012.

[Ye & Dague 2013] L. Ye et P. Dague. *Undecidable Case and Decidable Case of Joint Diagnosability in Distributed Discrete Event Systems*. International Journal On Advances in Systems and Measurements, vol. 6, no. 3-4, pages 287–299, 2013.

[Ye *et al.* 2010] L. Ye, Y. Yan et P. Dague. *Diagnosability for patterns in distributed discrete event systems*. In Proceedings of the 21st International Workshop on Principles of Diagnosis (DX'10), 2010.

[Ye *et al.* 2013] L. Ye, P. Dague et F. Nouioua. *Predictability Analysis of Distributed Discrete Event Systems*. In Proceedings of the 52nd IEEE Conference on Decision and Control (CDC-13), pages 5009–5015, 2013.

[Ye *et al.* 2015] Lina Ye, Philippe Dague et Farid Nouioua. *A Predictability Algorithm for Distributed Discrete Event Systems*. In International Conference on Formal Engineering Methods (ICFEM'15), pages 201–216. Springer, 2015.

[Ye 2011] Lina Ye. Optimized diagnosability of distributed discrete event systems through abstraction. PhD thesis, Université Paris-Sud, 2011.

[Yoo & Lafortune 2002] Tae-Sic Yoo et Stéphane Lafortune. *Polynomial-time verification of diagnosability of partially observed discrete-event systems*. IEEE Transactions on automatic control, vol. 47, no. 9, pages 1491–1495, 2002.

université PARIS-SACLAY | ÉCOLE DOCTORALE Ondes et Matière (EDOM)

Titre : Analyse à base de SAT de la diagnosticabilité et de la prédictabilité des systèmes à événements discrets centralisés et distribués

Mots clefs : SED,Diagnosticabilité,Prédictabilité, SAT, Planification

Résumé : Les systèmes complexes sont omniprésents dans nos vies, mais sujet à des pannes qu'il est important de détecter ou de prédire. Leur modélisation comme des systèmes à événements discrets (SED) est un moyen naturel de les représenter pour les étudier formellement. Ainsi, un système peut être décrit par un ensemble d'états tels que son état actuel est obtenu après avoir déclenché une séquence d'événements. Ces événements sont prédéfinis dans un ensemble fini et peuvent être déclenchés spontanément dans le système. Tous ces événements ne sont pas observables / mesurables et certains d'entre eux sont considérés comme fautifs, donc ils modélisent un changement anormal entre deux états du système.
Le processus de diagnostic de SED a pour but de déterminer avec certitude si le système est actuellement dans un état défectueux ou dans un état normal, c'est-à-dire si un changement anormal d'un état du système s'est produit ou non. À cette fin, un observateur de système ne dispose que de la séquence d'événements observables pour décider le diagnostic de l'état actuel du système. Cependant, cet état peut être ambigu (normal ou défectueux) en fonction des observations disponibles. En outre, il peut être définitivement ambigu ! La possibilité de le désambiguïser en utilisant un nombre fini d'observations est appelée la diagnosticabilité d'une occurrence d'événement fautif. La faute est diagnosticable si toutes ses occurrences le sont et le système est diagnosticable si toutes ses fautes le sont. De même, la possibilité de prédire une occurrence future d'une faute en utilisant les événements observables la précédant est appelée la prédictabilité d'un événement fautif. Les deux problèmes de la diagnosticabilité et de la prédictabilité d'un événement peuvent être généralisés pour étudier celles d'un motif d?événements, soit un langage clos par extension représenté par une machine à états finis.
Cette thèse considère dans sa première partie les problèmes de vérification de la diagnosticabilité et de la prédictabilité d'un événements fautif et de la diagnosticabilité d'un motif d'événements dans les systèmes à événements discrets centralisés et distribués (avec des événements de communication synchrone observables ou non), à l'aide de solveurs SAT. Ainsi, nous les avons encodés comme des problèmes SAT, avons étudié des variantes incrémentales et fourni des résultats expérimentaux qui prouvent le passage à l?échelle et la flexibilité de cette approche. Dans la deuxième partie, nous avons introduit le problème de la planification de la diagnosticabilité. Ce problème consiste à trouver un plan d'actions (une séquence d'événements intentionnels prédéfinis à la conception) qui garantit, lorsqu'il est appliqué à un ensemble donné d'états potentiels du système actuel appelé état courant de croyances, de conduire le système dans un état de croyances diagnosticable d?où on peut le laisser s'exécuter librement (sans les actions de contrôle). Ce problème peut survenir après une intervention externe sur le système, comme par exemple l'application d'un plan de réparation après la détection d'une faute. Ainsi, cette approche peut garantir la possibilité de détecter d'autres futures fautes du système. Nous avons analysé ce problème et prouvé qu?il est PSpace-complet puis nous avons proposé trois méthodes pour engendrer un plan diagnosticable, que nous avons comparées sur un banc d?essais créé à cette fin.

Title : SAT-Based Diagnosability and Predictability Analysis in Centralized and Distributed Discrete Event Systems

Keywords : DES, Diagnosability, Predictability, SAT, Planning

Abstract : Complex systems are omnipresent in our lives, but subject to failures that it is important to detect or predict. Discrete event system (DES) modeling is a natural way to represent and study such systems formally. Thus a system can be described by a set of states such that its current state is obtained after firing a sequence of events. These events are predefined in a finite set and can be fired spontaneously in the system. Not all these events are observable (measurable) and some of them are considered faulty, thus they model an abnormal change between two system states.
The diagnosis process in DES aims at determining with certainty if the system is currently in a faulty state or in a normal one, i.e., if an abnormal change of a system state has occurred or not. To this end, a system observer has only the sequence of observable events to decide the current status of the system state. However this state might be currently ambiguous (normal or faulty) according to the available observations. Moreover it can be permanently ambiguous! The possibility to disambiguate it using a finite number of observations is called the diagnosability of a faulty event occurrence. The fault is diagnosable if all its occurrences are diagnosable and the system is diagnosable if all its faults are diagnosable. Similarly, the possibility to predict a future occurrence of a fault using its preceding observable events is called the predictability of a faulty event occurrence. Both problems of diagnosability and predictability can be generalized to study the diagnosability or the predictability of a pattern of events, i.e., an extension-closed language represented by a finite state machine.
This thesis considers in its first part the problems of checking event diagnosability, event predictability and pattern diagnosability in centralized and distributed (with observable or unobservable synchronous communication events) discrete event systems, using SAT solvers. Thus we have encoded them as SAT problems, studied incremental SAT variants and provided experimental results that prove the scalability and flexibility of this approach. In the second part, we have introduced the diagnosability planning problem. This problem consists in finding a plan of actions (intentional/designful predefined events) that ensures, when applied on a set of potential current system states (called a current belief state), to drive the system in a diagnosable belief state from which it can be left to run freely (without control actions). This problem can arise after an external intervention on the system, like the application of a repair plan after a fault detection.Thus this approach can ensure the possibility to detect the system further faults. We analyzed this problem, proved its PSpace-completeness and proposed three methods to find the intended plan that we compared on a benchmark created for this purpose.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France