Time Decomposition for Diagnosis of Discrete Event Systems

Xingyu Su

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

August 2016

© Xingyu Su 2016

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

Xingyu Su
8 August 2016

To everyone who kindly offers their help.

Acknowledgments

I would like to thank my PhD supervision panel, which consists of Alban Grastien, Yannick Pencolé, and Patrik Haslum. I appreciate this golden opportunity of post-graduate study, and I would like to thank everyone who kindly offers their help.

Abstract

Artificial intelligence diagnosis is a research topic of knowledge representation and reasoning. This work addresses the problem of on-line model-based diagnosis of Discrete Event Systems (DES). A DES model represents state dynamics in a discrete manner. This work concentrates on the models whose scales are finite, and thus uses finite state machines as the DES representation. Given a flow of observable events generated by a DES model, diagnosis aims at deciding whether a system is running normally or is experiencing faulty behaviours.

The main challenge is to deal with the complexity of a diagnosis problem, which has to monitor an observation flow on the fly, and generate a succession of the states that the system is possibly in, called belief state. Previous work in the literature has proposed exact diagnosis, which means that a diagnostic algorithm attempts to compute a belief state at any time that is consistent with the observation flow from the time when the system starts operating to the current time. The main drawback of such a conservative strategy is the inability to follow the observation flow for a large system because the size of each belief state has been proved to be exponential in the number of system states. Furthermore, the temporal complexity to handle the exact belief states remains a problem. Because diagnosis of DES is a hard problem, the use of faster diagnostic algorithms that do not perform an exact diagnosis is often inevitable. However, those algorithms may not be as precise as an exact model-based diagnostic algorithm to diagnose a diagnosable system.

This Thesis has four contributions. First, Chapter 3 proposes the concept of simulation to verify the precision of an imprecise diagnostic algorithm w.r.t. a diagnosable DES model. A simulation is a finite state machine that represents how a diagnostic algorithm works for a particular DES model. Second, Chapter 4 proposes diagnosis using time decomposition, and studies window-based diagnostic algorithms, called Independent-Window Algorithms (IWAs). IWAs only diagnose on the very last events of the observation flow, and forget about the past. The precision of this approach is assessed by constructing a simulation. Third, Chapter 5 proposes a compromise between the two extreme strategies of exact diagnosis and IWAs. This work looks for the minimum piece of information to remember from the past so that a window-based algorithm ensures the same precision as using the exact diagnosis. Chapter 5 proposes Time-Window Algorithms (TWAs), which are extensions to IWAs. TWAs carry over some information about the current state of the system from

one time window to the next. The precision is verified by constructing a simulation. Fourth, Chapter 6 evaluates IWAs and TWAs through experiments, and compares their performance with the exact diagnosis encoded by Binary Decision Diagrams (BDD). Chapter 6 also examines the impact of the time window selections on the performance of IWAs and TWAs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Thesis Statement

Under the research theme of *future energy systems*, an electricity network is often tolerant to failures but may collapse after encountering a certain number of failures that take place within a short period of time. The *smart grid* is the upcoming generation of electricity networks. Challenges faced in the smart grid include prompt, reliable, and informative alarm processing to detect the points of failure. Another aspect is inferring the root cause for failures. Being able to accurately and promptly determine the occurrence of faults in a network allows rapid intervention, leading to faster restoration, and lower vulnerability to multiple faults.

This Thesis focuses on those systems that are modelled by a Discrete Event System (DES). A DES is a system model that represents state dynamics in a discrete manner [Cassandras and Lafortune, 2008]. It provides a common modelling framework for diagnosis problems such that a problem can be modelled as a DES while retaining the properties with a discrete nature in a system. Such abstraction gives the benefit of abstraction to focus on the logic of a system.

Diagnosis in Artificial Intelligence (AI) refers to the detection and identification of faults in a system. Research of diagnosis in AI offers the capacity to address some of the challenges mentioned above while research questions still remain in the context of on-line diagnosis for DES. This Thesis focuses on on-line diagnosis of DES and the quality of diagnosis is measured by *diagnosability*, as Section 1.2 explains the details.

1.2 Research Questions and Thesis Contributions

This Thesis addresses the problem of on-line diagnosis of a DES, which was initially proposed by Sampath et al. [1995]. Given a flow of observable events generated by the underlying system, the problem consists in determining whether the DES is operating normally or not, based on a behavioral model of it. On-line diagnosis means

diagnosing a system on the fly and in real time such that constraints on computational time and memory space are imposed.

Many previous works address this problem by the use of a diagnoser automaton [Sampath et al., 1995; Rozé and Cordier, 2002a], automata unfoldings [Baroni et al., 1999] that can also be distributed as in Pencolé and Cordier [2005]; Su and Wonham [2005], or encoded with binary decision diagrams (BDD) as in Schumann et al. [2010], and finally by the use of a satisfiability (SAT) solver as in Grastien et al. [2007].

The research challenge is to deal with the complexity of a diagnostic algorithm that monitors on the fly the observable flow and generates a succession of belief states that is consistent with the flow. The difficulty is the size of each belief state that is exponential w.r.t. the number of system states in the worst case [Rintanen, 2007]. Any of the precited work, except the SAT approach, proposes diagnostic algorithms that attempt to compute at any time a belief state that is consistent with the observable flow from the time when the system starts operating to the current time. The main drawback of such a conservative strategy is the inability to *follow* the observable flow for a large system due to the exponential size of the generated belief states and therefore the temporal complexity to handle them. Although diagnosis using SAT computes one trace in the system for an observation sequence, the complexity of a SAT problem is exponential to the number of propositional variables, which is linear to the number of state variables [Grastien et al., 2007].

This section defines the research questions, and provides an overview for the four contributions of this Thesis, i.e. verifying the precision of a diagnostic algorithm w.r.t. a DES model, Independent-Window Algorithms (IWAs), Time-Window Algorithms (TWAs), as well as the implementations and the experiments of IWAs and TWAs.

1.2.1 Verifying the Precision of a Diagnostic Algorithm w.r.t. a DES Model

The first contribution is to study the precision of a diagnostic algorithm w.r.t. a DES model in Chapter 3. It also proposes a novel approach to verify the precision of a diagnostic algorithm w.r.t. a DES model.

Because diagnosis of DES is a hard problem, the use of faster diagnostic algorithms is inevitable. Such algorithms include the IWAs proposed by Su and Grastien [2013], which do not carry all of the historical information about the system execution, and chronicle-recognition diagnostic algorithms by Dousson [1996], which uses pattern recognition techniques for diagnosis. However, these algorithms may be less precise to diagnose a diagnosable system than using an exact model-based diagnostic algorithm, e.g. Sampath et al. diagnosis [Sampath et al., 1995]. Faults are very harmful to a system and expensive to recover from if not correctly diagnosed. Hence,

it is essential to examine how to measure the quality of using a potentially imprecise diagnostic algorithm w.r.t. a diagnosable DES model.

In the literature, diagnosability of DES is an important property to measure the capability of a diagnostic system to identify faults and the quality of diagnosis. Diagnosability is well-known criterion of DES, which is initially proposed by Sampath et al. [1995]. Diagnosability of DES holds if using the model, a fault can always be diagnosed after it occurs. Furthermore, diagnosability testing has been a well-studied problem. Jiang et al. [2001] showed that proving non-diagnosability amounts to finding a critical witness, which is a pair of infinite executions on the model that are indistinguishable, i.e. they produce the same observations where one of them is faulty while the other one is nominal [Jiang et al., 2001]. Thus, diagnosability is proved by showing that there is no such witness. This approach is known as the *twin plant* method. [Yoo and Lafortune, 2002] also proposed their approach to test the diagnosability by constructing a *verifier*.

Chapter 3 proposes a novel approach to verify the precision of a diagnostic algorithm w.r.t. a DES model by constructing a *simulation*. A diagnostic algorithm is defined as *precise* if the algorithm will diagnose the fault after it occurs. As proposed in Chapter 3, precision can be verified using the known methods, such as the twin plant method by Jiang et al. [2001], on the condition that a simulation is built. A simulation is a modified model that *simulates* how a diagnostic algorithm runs on a given DES model. The precision holds iff there is no critical witness in the synchronisation of the DES model and the simulation. Finally, this work illustrates how to construct the simulation for chronicle-recognition diagnosis in Chapter 3, and for IWAs in Chapter 4.

1.2.2 Independent-Window Algorithms (IWAs)

The second contribution is to propose a new class of on-line DES diagnostic algorithms in Chapter 4, called Independent-Window Algorithms (IWAs). The on-line DES diagnosis problem was initially studied by Sampath et al. [1995]. Given a flow of observable events generated by a system, the problem consists in determining whether the DES is operating normally or not, based on the behavioural model of the system. This work uses the term *belief state* of the system to describe this computation. A belief state represents the set of global states that the system is possibly in after the given observations [Pencolé and Cordier, 2005; Rintanen, 2007]. Chapter 2 reviews the existing work of on-line diagnosis in the literature. It concludes that the main challenge is to deal with the complexity of a diagnostic algorithm that monitors the observable flow on the fly, and generates a succession of belief states

that are consistent with the flow. In fact, the difficulty lies in the size of each belief state, which is proved to be exponential w.r.t. the number of system states [Rintanen, 2007]. The existing diagnostic algorithms attempt to compute at any time a belief state that is consistent with the observable flow from the time when the system starts operating to the current time. The main drawback of such a conservative strategy is the inability to *follow* the observable flow for a large system due to the exponential size of the generated belief states. Also, the temporal complexity to handle all of the belief states remains a challenge.

Chapter 4 proposes a new diagnostic strategy, called window-based diagnosis. This strategy differs from the conservative strategy by proposing diagnostic algorithms that are only applied on the very last events of the observable flow, and *forget* about the past. Chapter 4 presents four Independent-Window Algorithms (IWAs), namely Al_p, Al_1, Al_2, and Al_3. IWAs slice an observation sequence into time windows so that each time window is diagnosed independently. IWAs diagnose a specified number of observations for one time window, and move to another time window without keeping any information. The four IWAs differ only in the time window selections.

IWAs offer an array of benefits. First, IWAs improve the flexibility and feasibility of diagnosis by computing diagnosis independently on separate time windows. Second, IWAs avoid the overhead of maintaining a precise tracking of the system state, and thus reduce the computational complexity of diagnosis. Third, IWAs are able to handle intermittent loss of communication, whereby the state of the system becomes unknown. This is because at the beginning of one time window, IWAs reset to every nominal state, or every faulty state, depending on the result at the end of the previous time window. In other words, diagnosis using IWAs does not require a complete observation sequence that is strictly compatible with the given DES model.

On the other hand, IWAs may cause imprecise diagnosis. Since IWAs diagnose time windows independently, imprecision happens when both current and past observations are necessary to understand the system behaviour. Chapter 3 presents the theory to verify the precision of a diagnostic algorithm w.r.t. a DES model by constructing a simulation. A simulation is a modified DES model that describes how a diagnostic algorithm computes diagnosis on the given DES system model. Chapter 4 illustrates the construction of a simulation in order to verify the precision of an IWA w.r.t. a DES model.

1.2.3 Time-Window Algorithms (TWAs)

The third contribution is to propose Time-Window Algorithms (TWAs) in Chapter 5. The strategy of TWAs is proposing a compromise between the two extreme strategies of exact diagnosis and imprecise diagnosis, e.g. a compromise between Sampath et al. diagnoser [Sampath et al., 1995] and IWAs [Su and Grastien, 2013]. Such a compromise is achieved by looking for the minimum piece of information to *remember* from the past, called *abstracted belief state*, so that a window-based algorithm will certainly ensure the same precision as using an exact diagnostic algorithm.

Chapter 5 formally presents two TWAs, namely, Al_5 and Al_6. TWAs are extensions to IWAs as presented in Chapter 4. TWAs carry over some information about the current state of the system from one time window to the next. Chapter 5 also describes how the precision of the two new TWAs is verified w.r.t. a DES model by constructing a simulation. Finally, Chapter 5 proposes a formal procedure to minimise the amount of information that a TWA needs to carry over to ensure no precision loss.

1.2.4 Implementations and Experiments of IWAs and TWAs

The fourth contribution is the implementations and the evaluations of the window-based diagnostic algorithms through experiments. Chapter 6 firstly describes the implementations, which read a DES model with multiple components and an observation sequence to diagnose using window-based diagnostic algorithms, i.e. IWAs and TWAs. The implementations also include the exact diagnostic algorithm Al_0 encoded using BDD as proposed by Schumann [2007]. Chapter 6 then describes the experimental models and scenarios before evaluating Al_p, Al_1, Al_2, Al_3, Al_5, and Al_6. The performance is measured by the precision of diagnosis, computational time, maximum memory use, average memory use, and diagnostic distance. Diagnostic distance is defined as the number of observations between a fault occurrence and the fault diagnosis of a diagnostic algorithm. Chapter 6 compares the above aspects of IWAs and TWAs with the performance of the exact diagnostic algorithm Al_0 encoded by BDD. It also examines the impact of the time window size on the performance of a window-based diagnostic algorithm.

1.3 Publications and Thesis Outline

This Thesis is organised as follows.

- Chapter 2 reviews the existing work of on-line diagnosis of DES in the literature, and examines the definition of the important property of DES diagnosabil-

ity. This chapter also examines the existing approaches to test the diagnosability of a DES model.

- Chapter 3 defines the precision of a diagnostic algorithm w.r.t. a DES model, and proposes to verify the precision by constructing a simulation. This theoretical contribution has been published at Su and Grastien [2014a].
- Chapter 4 proposes a class of windows-based diagnostic algorithms, called Independent-Window Algorithms (IWAs). The contribution of this chapter has been published at [Su and Grastien, 2013].
- Chapter 5 proposes a class of window-based diagnostic algorithms, called Time-Window Algorithms (TWAs). The contribution of this chapter has been published at [Su and Grastien, 2014b].
- Chapter 6 reports the implementations and the experiments of the IWAs of Chapter 4 and the TWAs of Chapter 5.
- Chapter 7 concludes this work, and provides an outline for the future directions.

Chapter 2

Background and Related Work

This chapter examines the state of the art in the research of diagnosis and diagnosability. Fault detection and isolation is widely used in automatic control of large and complex systems. Given a system and a set of observations, the essence of a diagnostic task is to find whether any fault has happened, and identify which one. Diagnosis is categorised into two approaches: (1) rule-based, and (2) model-based [Russell and Norvig, 2010].

Rule-based diagnosis is also known as an expert system, which requires an expertise and relies on direct mapping from the observations to diagnostic results, i.e.

$$\text{From } observations \rightarrow detection\ and\ identification\ of\ faults$$

However, rule-based diagnosis generally cannot adapt to system changes, which means that it frequently requires new expertise.

Model-based diagnosis overcomes the problem of rule-based diagnosis. A model provides the descriptions of a system, and the explanation for a fault will be obtained by the comparing the observations with the system model [de Kleer and Williams, 1987; Reiter, 1987]. This work focuses on model-based diagnosis. A system model can be represented by Boolean logic, equations, or Discrete Event Systems (DES). This work concentrates on DES, which is a system model that represents state dynamics in a discrete manner [Cassandras and Lafortune, 2008]. DES provides a common modelling framework for diagnosis problems. Using DES as a model for a diagnosis problem retains the properties with a discrete nature. Such abstraction has the benefit to focus on the logic of a system. Notice that DES is also useful for other modelling framework such as *hybrid system*. A hybrid system is a system model that involves both discrete and continuous dynamics. The study on hybrid system by [Bayoudh et al., 2008] is related to DES.

Fig. 2.1 shows a very simple DES model, which is to model the entrances to a café and a library [Jéron et al., 2006].

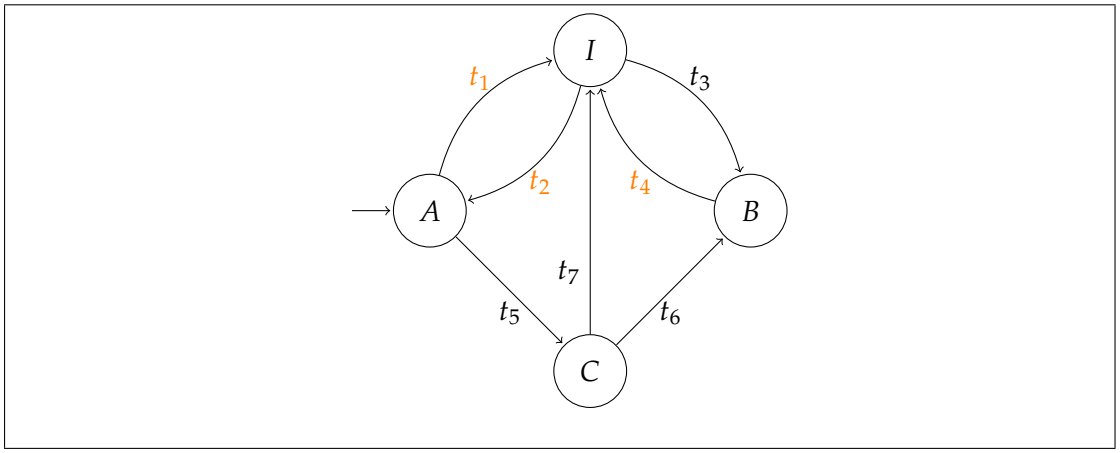

Figure 2.1: A simple DES system model

- the four states are the *reception* (A), the *office* (I), the *library* (B), and the *café* (C);
- A is the initial state;
- t_1, t_2, t_4 are the observable events;
- t_3, t_5, t_6, t_7 are the unobservable events.

Based on Fig. 2.1, the diagnoser should detect the fault if one goes to the café (C) twice without going to the library (B) at least once.

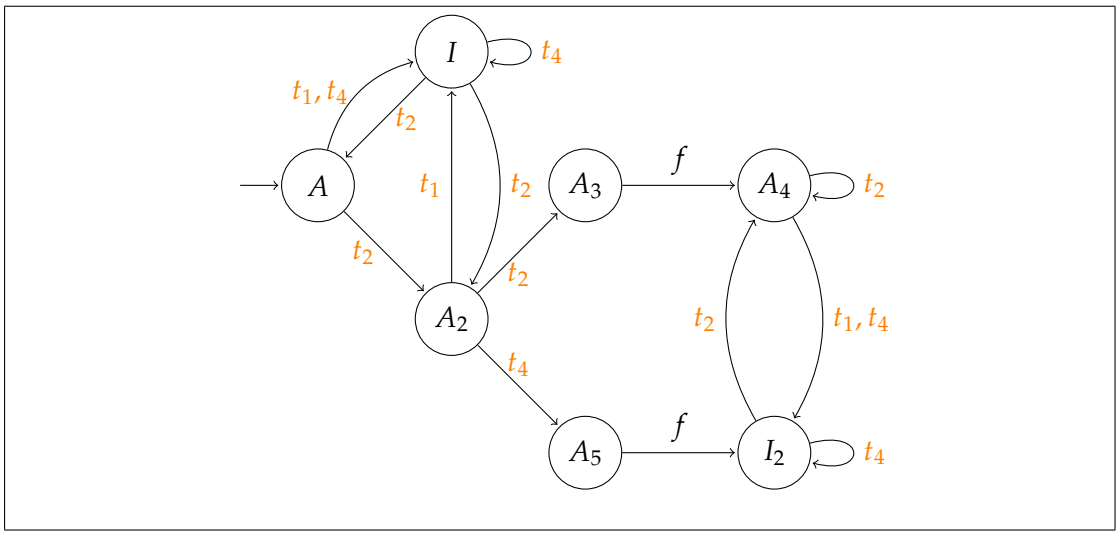

Figure 2.2: A diagnosis model for the café and library model described in Fig. 2.1

Fig. 2.2 shows the diagnosis model for the café and library model described in Fig. 2.1. A_4 and I_2 are faulty states while the other states are nominal. Given an observation sequence, if the diagnoser starts from the initial state A and the observation sequence ends in a faulty state, then the system is faulty.

Section 2.1 reviews the existing work regarding on-line diagnosis of DES. Section 2.2 reviews the existing work on DES diagnosability. Section 2.3 reviews the existing work on sensor minimisation and dynamic observers, which is related to diagnosability.

2.1 Existing Work of On-line Diagnosis of DES

Baroni et al. [1999] proposed diagnosis using the model of a network of communicating automata. The diagnostic process is to generate a representation of the behaviours of a decomposed system using observable events. It then combines a series of interpretations, and the diagnostic output is the faulty events incorporated within the reconstructed behaviour.

This approach starts with the definitions for the model of a DES component. A *component model* is a finite state machine denoted as M_c. $M_c = (\mathcal{S}, \mathcal{E}^{in}, \mathcal{I}, \mathcal{E}^{out}, \mathcal{O}, \mathcal{T})$ where

- $\mathcal{S}$ is the set of states;
- $\mathcal{E}^{in}$ is the set of input events;
- $\mathcal{I}$ is the set of input terminals;
- $\mathcal{E}^{out}$ is the set of output events;
- $\mathcal{O}$ is the set of output terminals;
- $\mathcal{T}$ is the transition function such that $\mathcal{T} : \mathcal{S} \times \mathcal{E}^{in} \times I \times 2^{\mathcal{E}^{out} \times \mathcal{O}} \mapsto 2^{S}$.

Baroni et al. [1999] also defined *link model* L as a 4-tuple. $L = (I, O, B, S)$ where

- I is the input terminal;
- O is the output terminal;
- B is the capacity;
- S is the saturation mode.

Furthermore, a *link* is an instantiation of a link model, which is a directed communication channel between two different components C and C' such that the output terminal of C and the input terminal of C' coincide with the input and output terminal of L. Based on components and links, an active system A is a set of components which are connected with each other by means of links among terminals. Within the link, E is the dangling event of L. The capacity B of L is the maximum number of

dangling events in L. The cardinality of L is denoted by $|L|$. If $|L| = B$, then L is *saturated*. $||L||$ is a sequence of dangling events in which the i-th generated event is denoted by $L[i]$. When a candidate event is consumed, it is dequeued from $||L||$.

Baroni et al. [1999] proposed a diagnostic algorithm to find a set of faulty components, and presented a reconstruct function to generate a graph-based representation. The aim is to reconstruct system behaviour based on the system observation. The challenge is the possibly huge size of the search space. Their approach does not generate any explicit search space, but incrementally makes up the overall behaviour of a system using a reconstruction plan starting from partial behaviours relevant to sub-systems. Eventually, diagnostic information at different abstraction levels is drawn from the reconstructed behaviour. In summary, using communicating automata saves the reconstruction of a global diagnoser.

The following five sub-sections reviews the existing techniques of on-line diagnosis including:

- Sampath et al. diagnosis,
- symbolic diagnosis using Binary Decision Diagrams,
- diagnosis using Satisfiability,
- distributed diagnosis, and
- incremental diagnosis.

2.1.1 Sampath et al. Diagnosis

Sampath et al. diagnoser is an accurate and exact diagnoser taking into account all information available from a DES model [Sampath et al., 1995]. *Belief state* is a set of states that a system is possibly in at a given time. In a Sampath et al. diagnoser, all potential belief states are pre-computed. It is a diagnoser that has off-line compilation of a model such that each state in the diagnoser is a set of pairs $\langle s, fm \rangle$ where s is a state of the system, and $fm \subseteq \{F_1, \ldots, F_f\}$ is a fault mode.

The initial state of a Sampath et al. diagnoser is $\{\langle s_0, \emptyset \rangle\}$. For each state in $s = \{\langle s_1, fm_1 \rangle, \ldots, \langle s_k, fm_k \rangle\}$, and for each observable event o from the given DES model, a Sampath et al. diagnoser creates a transition between s and s' labelled by o where s contains a pair $\langle s_i, fm_i \rangle$, and s' contains a pair $\langle s'_j, fm'_j \rangle$ such that

- there exists a path p labelled with unobservable events from the state s_i to the state s''; and,
- there exists a path from s'' to s'_j labelled by o; and,

- the fault mode consists of the previous fault mode, and the faults in the path, i.e. $fm'_j = fm_i \oplus p$.

For instance, $\{\langle s_1, fm_1\rangle, \langle s_2, fm_2\rangle\}$ from a Sampath et al. diagnoser means two possible situations. The first possibility is that the state after the last observation is s_1, and the fault that has occurred is fm_1. The second possibility is that the state after the last observation is s_2, and the fault that has occurred is fm_2.

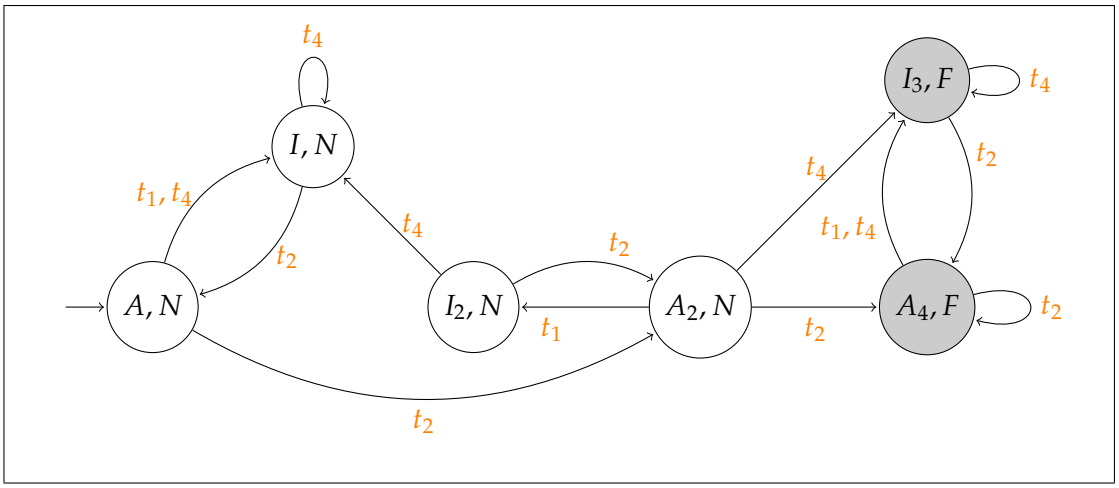

Figure 2.3: The Sampath et al. diagnoser for the café and library model described in Fig. 2.1

Sampath et al. diagnosis is an accurate and exact DES diagnostic approach, which pre-computes all belief states for a DES model. Fig. 2.3 shows the computed Sampath et al. diagnoser for the café and library model described in Fig. 2.1 [Jéron et al., 2006]. In each state, N means that the system is nominal; F means that the system is faulty, which represents the faulty behaviour of not going to the library between two visits to the café. The Sampath et al. diagnoser is used for a diagnostic task when given a sequence of observations. If a sequence ends in a state with a fault mode, then the system is faulty. Otherwise, the system is nominal or ambiguous. Once a Sampath et al. diagnoser is built and given a sequence of observations, a diagnostic task is to follow the states in the diagnoser to determine whether the system is nominal or faulty.

However, a Sampath et al. diagnoser has its disadvantages. First, it requires that the observation sequence must originate from the initial states, which is often infeasible for a real-world application. Second, the number of belief states has been proved to be exponential w.r.t the number of system states in the worst case [Rintanen, 2007]. It can be argued that the exponential blow-up of Sampath et al. diagnosis arises from the fact that the diagnoser maintains all the information it can about the observations

it received so far. For instance, the current belief state could be $\mathcal{B}'$ instead of $\mathcal{B}$ because of an early observation, and a Sampath et al. diagnoser needs to remember both $\mathcal{B}$ and $\mathcal{B}'$.

2.1.2 Symbolic Diagnosis using Binary Decision Diagrams

Schumann et al. [2010] studied the symbolic approaches for diagnosis where the representation is propositional logic, e.g. Binary Decision Diagrams (BDD). BDD enables encoding and manipulation of states and transitions without enumeration. Symbolic diagnosis aims to improve the efficiency of an exact diagnoser, which uses BDD to represent a DES model and a diagnoser.

Symbolic diagnosis involves encoding for states, events, and transitions. The symbolic representation for states is a set of Boolean variables denoted as V, i.e. a propositional formula. Then, the disjunction of propositional representation of states accurately describes a set of states, e.g. a Boolean variable for each state to represent whether a system is in that state or not, the combination of which will formulate which states that the system is in. Additionally, the "next" properties, denoted as V', are defined for the states, which are used for encoding the ending state of a transition. There is also a set of event variables to represent the events denoted as V_Σ. Finally, each transition is formulated as a conjunction of a beginning state from V, an event from V_Σ, and an ending state from V'.

State	State Encoding	"Next" State Encoding
A	$\neg v_1 \wedge \neg v_2$	$\neg v_3 \wedge \neg v_4$
B	$v_1 \wedge \neg v_2$	$v_3 \wedge \neg v_4$
C	$\neg v_1 \wedge v_2$	$\neg v_3 \wedge v_4$
I	$v_1 \wedge v_2$	$v_3 \wedge v_4$

Event	Event Encoding
t_1	$\neg v_5 \wedge \neg v_6$
t_2	$v_5 \wedge \neg v_6$
t_3	$\neg v_5 \wedge v_6$
t_4	$v_5 \wedge v_6$

Transition	Transition Encoding
$A \xrightarrow{t_1} I$	$(\neg v_1 \wedge \neg v_2) \wedge (\neg v_5 \wedge \neg v_6) \wedge (v_3 \wedge v_4)$

Table 2.1: Example of BDD encoding for the states, events, and part of the transitions of the DES model in Fig. 2.2: four states require two BDD variables, i.e. $V = \{v_1, v_2\}$; the "next" property of the states also requires two BDD variables, i.e. $V' = \{v_3, v_4\}$; four events require two BDD variables, i.e. $V_\Sigma = \{v_5, v_6\}$; this table shows encoding for one transition; the rest transitions are encoded accordingly.

Example of BDD Encoding Tab. 2.1 shows the BDD encoding for the DES diagnosis model in Fig. 2.2. Four states require two BDD variables, i.e. $V = \{v_1, v_2\}$. The "next" property of the states also requires two BDD variables, i.e. $V' = \{v_3, v_4\}$. Also, four events require two BDD variables, i.e. $V_\Sigma = \{v_5, v_6\}$. Notice that Tab. 2.1 shows the encoding for one transition. The rest transitions are encoded accordingly.

□

BDD has been widely used as a data structure. One of the benefits of symbolic representation is that a set of transitions does not require enumeration for each transition. In fact, a set of transitions is written in one formula ($\phi_\mathcal{T}$), which is the disjunction of every transition with the same event label.

Operations on a set of states include union, intersection, complement, emptiness, inclusion, and overlap. Furthermore, BDD operations provide existential quantification and substitution. Firstly, an occurrence of a variable is *free* if it is not within the scope of a quantifier. The BDD *substitution* operation ($t[t'/x]$) means replacing all free occurrence of x in t by t'. Secondly, the BDD *exists* operation means a variable can be replaced with 0 or 1, i.e.

$$\exists x.t = t[0/x] \vee t[1/x] \tag{2.1}$$

Therefore, symbolic representation of transitions derives the targets of a set of transitions ($\mathcal{T}$), i.e.

$$(\exists V.\exists V_\Sigma.\phi_\mathcal{T})[V'/V] \tag{2.2}$$

The set of states reached from states S through a single transition labelled by any event from Σ is written as:

$$(\exists V.\exists V_\Sigma.\phi_\mathcal{T} \wedge (\bigvee_{e\in\Sigma} e) \wedge \phi_S)[V'/V] \tag{2.3}$$

An important BDD operation, called *explore states*, computes the set of states reached from states S through any number of transitions labelled by events of Σ, as outlined in Algorithm 1.

Finally, given an observation sequence o and the initial state I, symbolic diagnosis tests against the fault set F. If the output is $false$, then there is a fault occurrence. Otherwise, there is no fault. Algorithm 2 outlines the procedure where Σ_u represents the set of unobservable events.

In summary, symbolic diagnosis offers advantages for encoding a DES model and operations of diagnosis. Given a sequence of observations, symbolic diagnosis computes the belief states on-the-fly, in contrast to pre-computation as in [Sampath et al., 1995]. Symbolic diagnosis improves the efficiency of the exact diagnosis because BDD

Algorithm 1: explore_states

Input: ϕ_S, Σ, V, V'
Output: the set of states that can be reached from the ϕ_S through any number of transitions labelled by events of Σ

$\phi := \phi_S$
$\phi' := \phi$
repeat
  $\phi := \phi'$
  $\phi' := \phi \vee (\exists V.\exists V_\Sigma.\phi_\mathcal{T} \wedge (\bigvee_{e\in\Sigma} e) \wedge \phi)[V'/V]$
until $\phi = \phi'$
return ϕ

Algorithm 2: test_fault_set

Input: ϕ_I, Σ_u, Σ, ϕ_T, o, V, V'
Output: $false$ if there is a fault occurrence; $true$ otherwise

$\phi := \phi_I$
foreach *observation fragment* o_i *in the observation sequence* o **do**
  $\phi := explore\_states(\phi, \Sigma_u, V, V')$
  $\phi := (\exists V.\exists V_\Sigma.\phi_T \wedge o_i \wedge \phi)[V'/V]$
return $\phi \wedge \phi_F \not\equiv \bot$

enables encoding and manipulation of states and transitions without enumeration. However, the symbolic approach alone has the limitation because it is still subject to exponential blow-up due to the amount of states in a large system.

2.1.3 Diagnosis using Satisfiability

Grastien et al. [2007] studied modelling and solving diagnosis problems of DES using satisfiability (SAT). Diagnosis of a DES is translated to a propositional SAT problem in order to be solved by the state-of-the-art SAT algorithms. The results show that SAT algorithms are able to efficiently explore the search space. The related work is using SAT in the context of planning [Kautz and Selman, 1992; Rintanen, 2009].

In SAT, a *literal* is a state variable or its negation. The set of all literals is $L = A \cup \{\neg a \mid a \in A\}$. The language $\mathcal{L}$ over A are all formulae that can be formed from A, or using the connectives $\vee$ and $\neg$. The standard definitions for connectives are adapted:

- $\phi \wedge \psi \equiv \neg(\neg\phi \vee \neg\psi)$
- $\phi \rightarrow \psi \equiv \neg\phi \vee \psi$
- $\phi \leftrightarrow \psi \equiv (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$

In the approach of diagnosis using SAT, the states of a DES model are represented by the assignment of a finite set of two-valued Boolean variables in the finite domains. Φ_{SD} is a formula for the system description $SD = \langle A, \Sigma_u, \Sigma_o, \delta, s_0 \rangle$. A SAT model consists of:

- a set of state variables A,
- a set of unobservable events Σ_u,
- a set of observable events Σ_o,
- $\delta \subseteq \Sigma_o \cup \Sigma_u \rightarrow 2^{\mathcal{L} \times 2^L}$, which assigns each event a set of *event instances* $\langle \phi, c \rangle$,
- an initial state s_0.

A *state* $s : A \rightarrow \{0, 1\}$ is a total function from the state variables to the constants 0 and 1 where 0 means $false$, and 1 means $true$. Grastien et al. [2007] used an *event instance* to describe an event e as a pair $\langle \phi, c \rangle \in \delta(e)$ meaning that e is associated with changes, denoted as c, in states that satisfy the condition of ϕ. When e takes place in s, one of the pairs $\langle \phi, c \rangle \in \delta(e)$ satisfying $s \models \phi$ is chosen, and the effect of the event is that the literals in c become true. The propositional variables with superscript of time step t are the following:

- a^t for all $a \in A$ where $t \in \{0, 1, 2, \ldots, n\}$
- e^t for all $e \in \Sigma_u \cup \Sigma_o$ where $t \in \{0, 1, 2, \ldots, n-1\}$
- ω^t for all $e \in \Sigma_u \cup \Sigma_o$ where $\omega \in \delta(e) \wedge t \in \{0, 1, 2, \ldots, n-1\}$

The successor state s' of a state s is denoted as $succ(s, c)$. An event sequence $e_0, e_1, e_2, \ldots, e_{n-1}$ leads to states $s_0, s_1, s_2, \ldots, s_n$ such that $\forall i \in \{0, 1, 2, \ldots, n-1\}$, $\exists \langle \phi, c \rangle \in \delta(e_i) \mid s_i \models \phi \wedge s_{i+1} = succ(s_i, c)$. However, a state s_i and an event e_i may not uniquely determine the successor state s_{i+1}. When an event occurs, the event must be possible in the current state, i.e.

$$\text{for every } \omega = \langle \phi, c \rangle \in \delta(e), \omega^t \rightarrow \phi^t.$$

The effects of the transitions are:

$$\text{for every } \omega = \langle \phi, c \rangle \in \delta(e), \omega^t \rightarrow \bigwedge_{l \in c} l^{t+1}.$$

For a diagnostic task, observations OBS is a set of pairs $\langle e, t \rangle$ where $e \in \Sigma_o$ is an observable event, and $t \in \mathbf{N}^+$ is a positive integer. Hence, observations are regarded

as a collection of timed observable events. A sequence of states and events is called a *trajectory*, which models the behaviour of a system. The diagnosis label of a trajectory says a trajectory is *faulty* if it contains any faulty event. Otherwise, a trajectory is *nominal*. A system status is *ambiguous* when nominal and faulty behaviours cannot be distinguished from a sequence of observations. The assumption is that it is sufficient to find any one nominal behaviour that is consistent with the observations, and the diagnostic output is that the system status is nominal. Formally, a system is nominal if there exists a sequence $\mathcal{E} = E_0, E_1, E_2, \ldots, E_{n-1}$ of events on the system model consistent with OBS such that $\mathcal{E}$ is nominal. Furthermore, Grastien et al. [2007] proposed a theorem that the solutions to a SAT problem $\Phi_{SD} \wedge \Phi_{OBS}$ represent the set of traces on SD consistent with the observations OBS and ending with the last observation of OBS.

In summary, diagnosis using SAT has advantages and disadvantages. Firstly, the formulation of a SAT problem enables a SAT solver to run efficiently. Diagnosis using SAT takes the advantage that once a trace has been computed, it can be reused for different diagnostic queries. Therefore, SAT diagnosis is beneficial when utilising the set of all computed traces. Secondly, Section 2.1.1 has reviewed that the construction of a Sampath et al. diagnoser may be extremely expensive because the size of a Sampath et al. diagnoser is exponential to the number of states in the system. The SAT approach has the advantage that there is no such direct dependency between run time and the number of states [Grastien et al., 2007].

On the other hand, diagnosis using SAT has a potential problem that the run time in the worst case grows exponentially to the length of the observation sequence. Another disadvantage is that the complexity of a SAT problem is exponential to the number of propositional variables, which is linear to the number of state variables, and linear to the number of time steps [Grastien et al., 2007].

In conclusion, diagnosis using SAT is still a difficult problem when the number of observations increases. Also, it is an open question which SAT solver is the best for a particular diagnosis problem. It has been recommended that incremental diagnosis and temporal windows have advantages to deal with this problem such that the diagnosis of a temporal window is ensured to be consistent with the next one. Notice that Section 2.1.5 will review the existing work on incremental diagnosis. Then, this work will propose time-window-based diagnostic algorithms in Chapter 4 and 5.

2.1.4 Distributed Diagnosis

Distributed diangosis focuses on a component-based system. It aims to compute diagnosis using relevant components of a system, and then merge to a final diagnosis.

This approach improves the capability of a centralised diagnoser, e.g. Sampath et al. diagnosis, to diagnose a component-based large-scale system.

Pencolé and Cordier [2005] proposed on-line distributed diagnosis for a large DES. The principle is *divide and conquer* where local diagnosis computes on each component, and the global diagnosis is produced for the entire system. A large DES is decomposed to multiple components. A *component* is an entity that has a finite set of internal states. Components communicate with each other using communication channels. The system is event-driven, and evolves with the occurrence of events on the components.

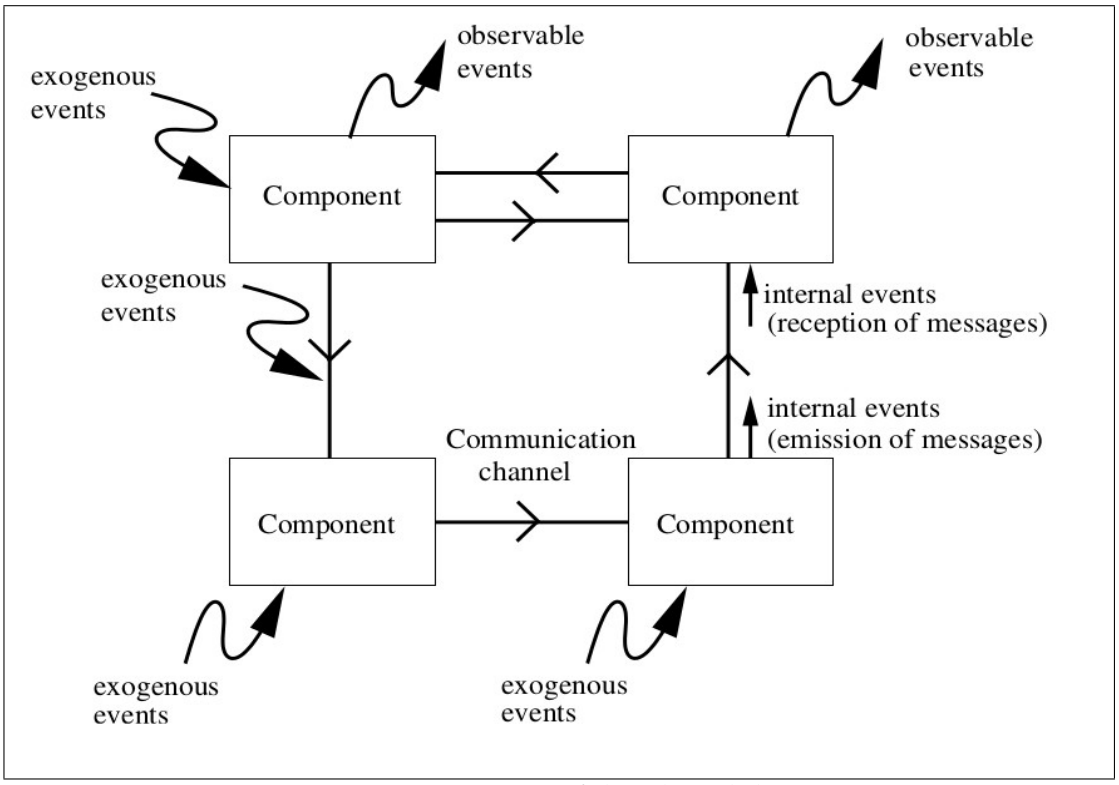

Figure 2.4: Event types of distributed diagnosis

Fig. 2.4 illustrates various event types and their data flow [Pencolé and Cordier, 2005]. An *exogenous event*, from the set Σ_{exo}, is an event produced by the environment of a system. An exogenous event will trigger a change of state in one component. During a state change, the affected components may produce communication events, from the set Σ_{com}, towards its neighbourhood by emitting messages via its communication channels. Those affected components may also produce observable events, from the set Σ_{obs}, towards the environment by emitting observable messages. The reception of a message from a communication channel is an internal event, and may change the internal state of a component. In that case, the components affected by

this event may also emit observable events. Pencolé and Cordier [2005] defined that a component c_i receives the following two kinds of events:

- exogenous events Σ^i_{obs}, which are events from the environment such that $\Sigma^i_{obs} \subseteq \Sigma_{exo}$;
- communication events $\Sigma^i_{com\_rcv}$, which are messages received from other components of a system such that $\Sigma^i_{com\_rcv} \subseteq \Sigma_{com}$.

A component c_i also emits the following two kinds of events:

- observable events Σ^i_{obs}, which are emitted messages that are observed by a supervision system such that $\Sigma^i_{obs} \subseteq \Sigma_{obs}$;
- communication events $\Sigma^i_{com\_emit}$, which are messages emitted to other components of the system such that $\Sigma^i_{com\_emit} \subseteq \Sigma_{com}$.

Thus, a component model c_i is a communicating finite state machine $\Gamma_i = (\Sigma^i_{com\_rcv}, \Sigma^i_{com\_emit}, Q_i, E_i)$ where

- Q_i is the set of component states,
- $E_i \subseteq (Q_i \times \Sigma^i_{com\_rcv} \times 2^{(\Sigma^i_{com\_emit})} \times Q_i)$ is the set of transitions.

Furthermore, components are connected by bounded communication channels. A message emitted from component c_1 to component c_2 corresponds to the reception of this message by a communication channel between c_1 and c_2. Finally, a system model Γ consists of the following:

- a set of component models $\{\Gamma_1, \ldots, \Gamma_n\}$;
- a set of exogenous events Σ_{exo};
- a set of observable events Σ_{obs}; and,
- a set of internal events Σ_{int} such that:
  - $\{\Sigma^1_{obs}, \ldots, \Sigma^n_{obs}\}$ is a partition of Σ_{obs};
  - $\{\Sigma^1_{exo}, \ldots, \Sigma^n_{exo}\}$ is a partition of Σ_{exo};
  - $\{\Sigma^1_{intreceived}, \ldots, \Sigma^n_{intreceived}\}$ and $\{\Sigma^1_{intemitted}, \ldots, \Sigma^n_{intemitted}\}$ are partitions of Σ_{int};
  - $\forall e \in \Sigma_{int},\ (\exists \Gamma_i \mid e \in \Sigma^i_{intreceived}) \wedge (\exists \Gamma_j \mid e \in \Sigma^j_{intemitted}) \wedge (i \neq j)$.

Distributed diagnosis computes local diagnosis on each component, and then builds a global diagnosis of the whole system from local diagnoses. A local diagnosis $\Delta_{\Gamma_i}(X^{\Gamma_i}_{init}, \mathcal{O}_{\Gamma_i})$ of Γ_i according to the sequence $\mathcal{O}_{\Gamma_i}$ is a finite state machine $(\Sigma_i, 2^{(\Sigma^i_{out})}, Q_{\Delta_i}, E_{\Delta_i})$ where

- Σ^i_{in} is a set of input events $\Sigma^i_{in} = \Sigma^i_{exo} \cup \Sigma^i_{intreceived}$;
- Σ^i_{out} is a set of output events $\Sigma^i_{out} = \Sigma^i_{obs} \cup \Sigma^i_{intemitted}$;
- $Q_{\Delta_i} \subseteq Q_i \times (\Sigma^i_{obs})^*$ is a set of states;
- $E_{\Delta_i} \subseteq (Q_{\Delta_i} \times \Sigma^i_{in} \times 2^{\Sigma^i_{out}} \times Q_{\Delta_i})$ is a set of transitions.

As local diagnoses are represented by automata, a global diagnosis $(\Delta_\Gamma(X^\Gamma_{init}, \mathcal{O}))$ is built from local diagnoses. Suppose that the sequence of observations $\mathcal{O} \mid \Sigma^i_{obs}$ received by the supervision system from each component Γ_i corresponds exactly to the sequence of local observations $\mathcal{O}_{\Gamma_i}$ emitted by Γ_i, then $\mathcal{O} \mid \Sigma^i_{obs} = \mathcal{O}_{\Gamma_i}$. The global diagnosis is computed by applying the classical composition operation between the communicating finite state machines so that they are synchronised on the internal events exchanged between the local diagnoses, i.e.

$$\Delta_\Gamma(X^\Gamma_{init}, \mathcal{O}) = \bigodot_{i=1}^{n} \Delta_{\Gamma_i}(X^{\Gamma_i}_{init}, \mathcal{O} \mid \Sigma^i_{obs})$$

In summary, distributed diagnosis aims to analyse a flow of observations on-line in a more efficient way since centralised diagnosis is not practical for a large-scale component-based system. One of the benefits of distributed diagnosis is that this approach does not require the computation of a global model. Also, it is feasible for distributed diagnosis to be extended to incremental diagnosis, which provides on-line diagnosis and assists supervision. However, distributed diagnosis does not completely solve the problem of keeping track of all belief states, the size of which is exponential to the number of states.

2.1.5 Incremental Diagnosis

Grastien and Anbulagan [2009] proposed incremental diagnosis with the aim to update diagnosis when new observations arrive. Incremental diagnosis is an open research problem of on-line diagnosis. The incremental approach updates diagnostic results regularly. Grastien and Anbulagan [2009] proposed *preferred diagnosis* to deal with the situation when it is not possible to retrieve precisely what behaviour generated these observations. For systems that are not diagnosable, the sequence of faults that actually occurred on the system cannot always be precisely deduced from the

sequence of observations. Hence, several diagnoses can be returned to explain the observations. Grastien and Anbulagan [2009] defined the notion of *preference* such that the expected output of a diagnoser is the preferred diagnosis. The diagnosis preference is an ordering relation $\preceq$ on a set of diagnoses. $\Delta \preceq \Delta'$ denotes that diagnosis Δ is preferred to the diagnosis Δ'. Accordingly, $p \preceq p'$ denotes that a path p is preferred to path p', which means that the diagnosis of p is preferred to the diagnosis of p'. A Non-Exhaustive Diagnosis Engine (NEDE) is defined as a diagnostic engine such that a diagnoser only returns the preferred diagnosis, and this engine should return an explanation of diagnosis. Therefore, preferred diagnosis saves computation, which avoids exponential numbers of all diagnoses.

Grastien and Anbulagan [2009] further argued that NEDE would be sufficient for a diagnoser since it only needs to consider the preferred diagnoses. This is because it is possible to turn an exhaustive diagnoser that returns explanations into an NEDE by determining the preferred diagnosis and extracting one of its explanations. Efficient computation is feasible if the search space is pruned as soon as the initial diagnosis is computed. For instance, SAT-based diagnoser is an instance of NEDE.

Incremental diagnosis means given a DES A, a sequence of observation $obs_j = [o_1, o_2, o_3, \ldots, o_j]$ and a continuation $obs_{j'} = [o_1, o_2, o_3, \ldots, o_{j'}]$ of obs_j such that $j' \geq j$, the incremental diagnosis is the computation of the diagnosis of $obs_{j'}$ using the diagnosis of obs_j. Grastien and Anbulagan [2009] proposed Algorithm 3 to compute a path and the associated diagnosis.

Algorithm 3: Algorithm of Incremental Diagnosis with a NEDE

Input: prediction window μ, NEDE Ω, observation flow $\mathcal{OF}$
Output: p_u

$S_0 := I$
$obs := []$
$p := \emptyset$
$p_u := \emptyset$
while $\mathcal{OF}$ *generates new_obs* **do**
  $obs := obs \oplus new\_obs$
  $p' := \Omega(obs, S_0)$
  if p' *not found* **then**
    $p' := \Omega(obs, Q)$
  $p_u := p \oplus p'$
  $p := p \oplus$ remove_last (p', μ)
  $obs :=$ keep_last (obs, μ)
  $S_0 :=$ { last_state (p)}
return p_u

A prediction window μ indicates how many observations are required before a diagnosis is established. The input of NEDE, denoted as Ω, consists of a set of initial states and a sequence of observations. The output is a preferred explanation. An observation flow $\mathcal{OF}$ provides an ordered sequence of observations. This flow does not return all observations at once. Starting from Line 1, the established explanation of observations is stored in p and its final state is stored in S_0. The path p_u represents the current explanation of all observations, which can be returned any time if the procedure is on-line. *obs* contains a list of observations that are not yet explained in p and its size is bounded by μ. Whenever new observations, denoted as *new_obs*, become available, new observations are added to the observations not explained and a path p' is computed from a state of S_0. Both paths p_u and p are updated, and the last μ observations are kept. Finally, the set of states S_0 is updated.

This algorithm allows a diagnoser to backtrack until the μ previous observations. The backtracking is limited to this position to avoid complexity explosion. However, this algorithm may return an incorrect diagnosis if the value μ is set poorly, and the algorithm may return a diagnosis that is not the preferred one. One way to overcome this problem is to limit the backtracking until a bounded number of observations, and return a failure message. The alternative approach is to run the diagnostic algorithm from any system state Q in Line 9, which is called *diagnosis reset*. Diagnosis reset means that the state at the beginning of a prediction window abruptly jumps for no sensible reason. If the supervision system is able to recover the system state, the diagnosis will be incorrect around the diagnosis reset, but should be correct before and after. Finally, the incremental algorithm should warn a human agent of any incorrectness.

In summary, incremental diagnosis does not compute all diagnoses but only the preferred one. The benefit of the incremental approach is to simplify the problem as the number of possible diagnoses and their explanations is usually exponential to the number of system states. Furthermore, a human agent is usually more interested in the preferred diagnosis. Therefore, incremental diagnosis is advantageous when not all observations are known in the first place. On the other hand, the NEDE algorithm cannot ensure the completeness or consistency of diagnosis. Suggestions for future work include bounded preferred diagnoses, reusing conclusions from previous computations, and decomposition into sub-systems for diagnosis.

2.1.6 Diagnosis using Supervision Pattern

Jéron et al. [2006] proposed supervision pattern for DES diagnosis. A supervision pattern is a model that is general enough to capture past occurrences of particular

trajectories of a DES model.

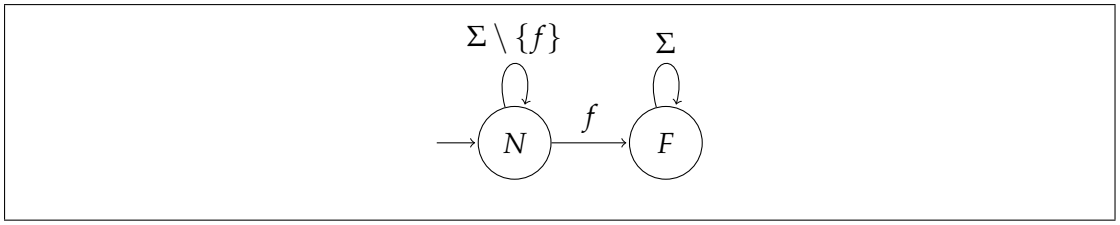

Figure 2.5: Example 1 of supervision pattern for one fault f

For example, Fig. 2.5 shows a supervision pattern to diagnose the fault f. At the language level, a trajectory $s \in \Sigma^*$ is faulty if $s \in \Sigma^*.f.\Sigma^*$. The supervision pattern Ω_f exactly recognises the language of $\Sigma^*.f.\Sigma^*$.

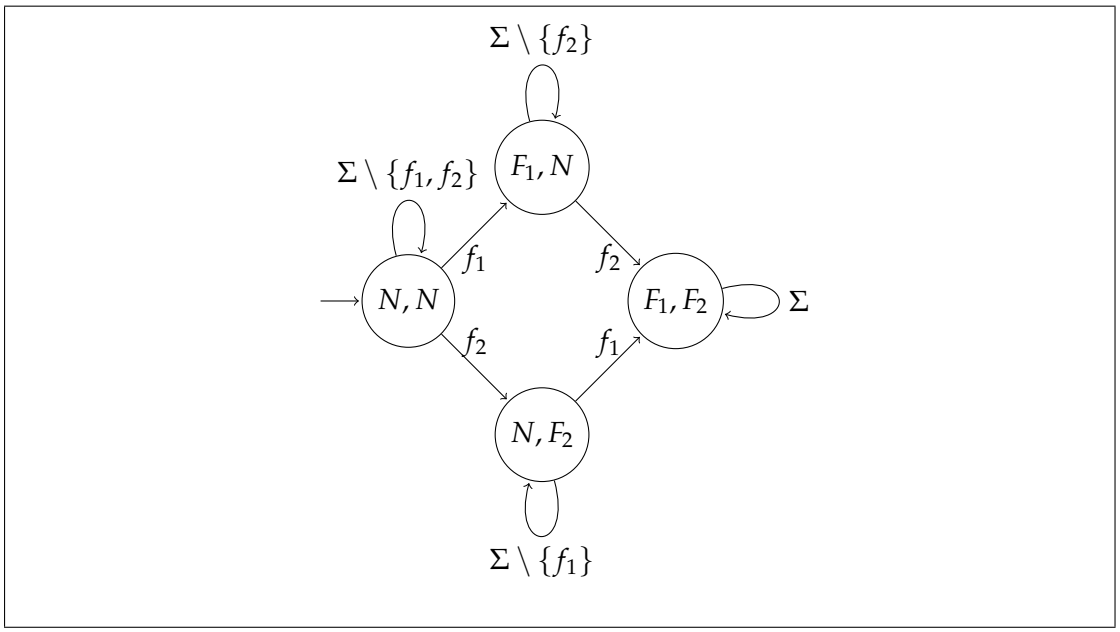

Figure 2.6: Example 2 of supervision pattern for two faults f_1 and f_2

Fig. 2.6 shows a supervision pattern to diagnose two faults, i.e. f_1 and f_2. Diagnosing the occurrence of these two faults in an trajectory means deciding the membership of this trajectory in $\Sigma^*.f_1.\Sigma^* \cap \Sigma^*.f_2.\Sigma^* = \mathcal{L}_{F_1}(\Omega_{f_1}) \cap \mathcal{L}_{F_2}(\Omega_{f_2})$ where $\Omega_{f_i}, i \in \{1,2\}$ are isomorphic to the supervision pattern Ω_f as described in the example of Fig. 2.5. Finally, the supervision pattern is the product $\Omega_{f_1} \times \Omega_{f_2}$, which accepts language in $F_1 \times F_2$, i.e.

$$L_{F_1 \times F_2}(\Omega_{f_1} \times \Omega_{f_2}) = \mathcal{L}_{F_1}(\Omega_{f_1}) \cap \mathcal{L}_{F_2}(\Omega_{f_2}).$$

In summary, supervision patterns allow to generalise the properties to be diagnosed, and render them independently, which does not require the system description.

2.1.7 State-based Diagnosis

Hashtrudi Zad et al. [2003] proposed state-based approach for on-line DES diagnosis where system states are partitioned according to the system condition, and a diagnoser uses measurements, e.g. sensor readings, to determine the nominal and/or faulty partition that the state belonged to, i.e. the system condition, when the last measurement was received. The time complexity of state-based diagnosis proposed by Hashtrudi Zad et al. [2003] is polynomial after *model reduction scheme*, which reduces the cardinality of the system states. In contrast to event-based diagnosis, state-based diagnosis does not assume or require simultaneous initialisation of the system and the diagnoser; also, no knowledge about the system state or condition prior to the diagnosis initiation is required [Hashtrudi Zad et al., 2003], which allows to diagnose faults that may have occurred before diagnosis starts. Finally, an event-based diagnosis problem can be formally transformed to state-based diagnosis [Hashtrudi Zad et al., 2003].

2.2 Existing Work on DES Diagnosability

After reviewing the existing work of on-line diagnosis, this section reviews the existing work on DES diagnosability. Diagnosability of an observable system holds when using the model, a fault can be diagnosed after it occurs Sampath et al. [1995]. This property is desirable when designing a diagnostic system because it should be able to determine that a fault has occurred whenever it did. Furthermore, diagnosability is helpful to analyse a system as there is often a delay between the occurrence of a fault and the time when it is diagnosed. If such a delay is always finite, then diagnosability is guaranteed. This section reviews the definition of DES diagnosability, and the existing methods to test the diagnosability of a DES model.

2.2.1 Definition of DES Diagnosability

Sampath et al. [1995] formally defined the diagnosability of a DES model. Firstly, they defined that diagnosability of a language L holds if it is possible to detect fault occurrences using observed events within a finite delay. A prefix-closed and live language L is diagnosable w.r.t. the projection P, and w.r.t. the partition Π_f on the

faulty events Σ_f if the following equation holds.

$$(\forall i \in \Pi_f)(\exists n_i \in \mathbf{N})[\forall s \in \Psi(\Sigma_{f_i})](\forall t \in L/s)[||t|| \geq n_i \Rightarrow D] \tag{2.4}$$

$$\text{where } D = \omega \in P_L^{-1}[P(st)] \Rightarrow \Sigma_{f_i} \in \omega;\ ||t|| \text{ represents the length of a trace;} \tag{2.5}$$

$$\text{and } L/s = \{t \in \Sigma^* | st \in L\} \text{ is the post-language of L.} \tag{2.6}$$

$$\text{Erasing unobservable events from a trace, and the result is:} \tag{2.7}$$

$$P_L^{-1}(y) = \{s \in L | P(s) = y\}; \tag{2.8}$$

$$\text{A trace whose final event belongs to } \Sigma_{fi}\text{: } \Psi(\Sigma_{f_i}) = \{s\sigma_f \in L | \sigma_f \in \Sigma_{f_i}\}. \tag{2.9}$$

In their definition of diagnosability, s is any trace in a system that ends with a faulty event from the set Σ_{f_i}. t is a sufficiently long continuation of s. Condition D requires that every trace in a language that produces the same observable events as the trace st should contain a faulty event from the set Σ_{f_i} [Sampath et al., 1995]. Therefore, fault occurrences can be detected within a finite delay, i.e. at most n_i transitions after s.

2.2.2 Verification of DES Diagnosability in Polynomial Time

The original diagnosability verification algorithm proposed by Sampath et al. [1995] relies on the construction of a diagnoser. The computational complexity of such a construction requires exponential time w.r.t. the number of system states [Jiang et al., 2001; Yoo and Lafortune, 2002]. Jiang et al. [2001] proposed diagnosability testing of DES that avoids the construction of a diagnoser for the system, and runs in polynomial time. This approach is known as the *twin plant* method. It uses a model and a copy of the model, called the bad twin and the good twin. A bad twin is allowed to contain faulty behaviours while a good twin can only include nominal behaviours that will produce the same observations as faulty behaviours. Then, the twins are synchronised to generate a twin plant. A path is defined as ambiguous if the path contains a fault, and the path is valid in both bad twin and good twin. Finally, a model is diagnosable if there is no loop on any ambiguous path. Otherwise, it is not diagnosable. This is because if there is an ambiguous path that is infinite, then it will not be possible to distinguish between the nominal and faulty behaviours. In other words, a system is diagnosable if there is no two infinite paths in the model that consists of the same observations such that one contains a fault while the other one does not.

Grastien [2009] implemented the twin plant method. A DES model is defined as $\langle Q, E, T, I \rangle$ where

- Q is a set of states;

- E is a set of events;
- $T \subseteq Q \times E \times Q$ is a set of transitions; and,
- $I \subseteq Q$ is a set of initial states.

The set Q of states is partitioned in two subsets Q_N of nominal states, and Q_F of faulty states. E is partitioned in two subsets E_{obs} of observable events, and E_u of unobservable events.

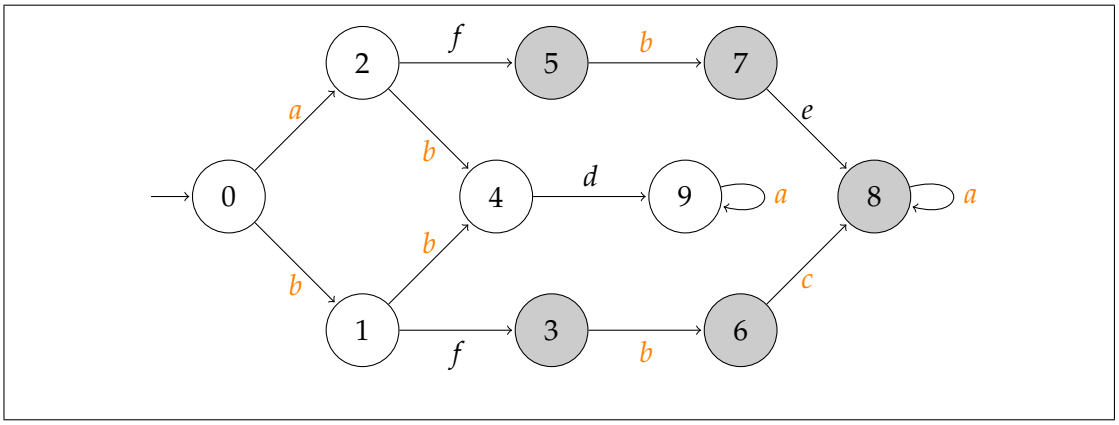

Figure 2.7: A DES model to build a twin plant

Fig. 2.7 shows a DES model that is used to build to a twin plant in this section [Brandán-Briones et al., 2008; Grastien, 2009]. a, b, c are observable events while d, e, f are unobservable events. States $5, 7, 8, 3, 6$ are faulty while the other states are nominal.

Grastien [2009] formally defined the twin plant of the given DES model as a finite state machine $\langle \mathcal{Q}, \mathcal{E}, \mathcal{T}, \mathcal{I} \rangle$ such that

- $\mathcal{Q} = Q_N \times Q$
- $\mathcal{E} = E_{obs} \cup ((E \backslash E_{obs}) \times \{1, 2\})$
- $\mathcal{T} = \{\langle q, e, q' \rangle | q = \langle q_1, q_2 \rangle \wedge q' = \langle q'_1, q'_2 \rangle \wedge$
  $((e \in E_{obs} \wedge \langle q_1, e, q'_1 \rangle \in T \wedge \langle q_2, e, q'_2 \rangle \in T) \vee (e \in (E \times \{i\}) \wedge \langle q_i, e, q'_i \rangle \in T \wedge$
  $(q_j = q'_j) \wedge (j \neq i)))$;
- $\mathcal{I} = I \times I$

Fig. 2.8 shows the twin plant for the DES model in Fig. 2.7. Shaded states are ambiguous because they are combinations of nominal and faulty states. There is at least one ambiguous path that contains a loop, e.g. $00' \xrightarrow{a} 22' \xrightarrow{f_2} 25' \xrightarrow{b} 47' \xrightarrow{e_2} 48' \xrightarrow{d_1}$

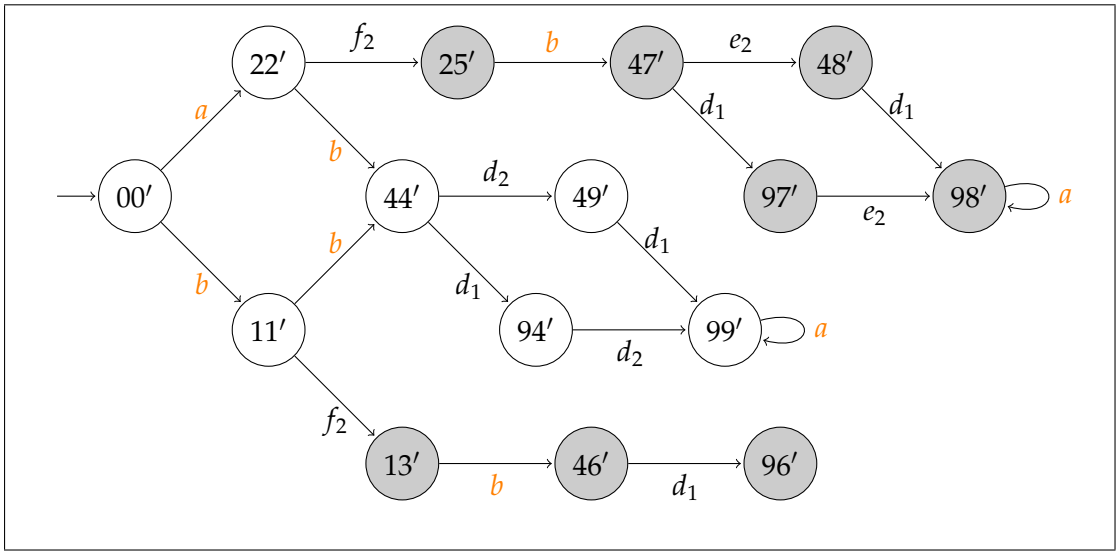

Figure 2.8: Twin plant for the DES model in Fig. 2.7

$98' \xrightarrow{a} 98' \ldots$. Consequently, this model is not diagnosable since nominal and faulty behaviours cannot always be distinguished.

Yoo and Lafortune [2002] proposed the *verifier* method to test the diagnosability of a DES model that runs in polynomial time. Their approach constructs a non-deterministic automaton, called a verifier. Given an automaton $M = \langle Q, \Sigma, T, I, L \rangle$, and fault type F_i, Yoo and Lafortune [2002] defined verifier as a non-deterministic automaton, denoted as V_{F_i}. They also defined a set of labels for the faulty event of type F_i such that $Label_i = \{N, F_i\}$. V_{F_i} is defined as F_i-confused if there is at least one state in the verifier whose meaning is that the nominal and faulty behaviours cannot be distinguished, and there is a cycle on that state. Yoo and Lafortune [2002] proposed a theorem that a system is diagnosable if the verifier for the system is never F_i-confused.

2.2.3 Symbolic Model Checking for Diagnosability Testing

Model checking is a formal verification technique [Clarke et al., 1999]. Model checking is applied to the design of industrial digital systems. It verifies whether the behaviours of a system satisfy a given property. A system is presented in Kripke structure while properties are expressed in temporal logic. Model checking algorithms are based on the exhaustive exploration of the Kripke structure. If a property is violated, it will produce a counterexample. This technique is completely automatic. Symbolic model checking is based on the manipulation of sets of states described by Boolean formulae. Efficient engines, such as BDD and SAT solvers, can be used to explore of the search space.

Cimatti et al. [2003] studied diagnosability testing using symbolic model checking. They proposed how to verify at design time whether the diagnosis system is able to infer the required information at run time. Their work showed how to reduce diagnosability testing to a model checking problem. The finite state machine modelling the dynamic system is replicated to construct pairs of scenarios. The diagnosability conditions are formally expressed in temporal logic so that the diagnosability testing is to solve a model checking problem.

Grastien [2009] also studied symbolic testing of diagnosability, and proposed two algorithms implemented using symbolic tools, namely the *forward algorithm* and the *backward algorithm*. Grastien [2009] explained how to combine distributed diagnosis with symbolic diagnosis, and examined the minimal set of sensors to ensure the diagnosability. Algorithm 4 is the forward algorithm to test the diagnosability of a DES model. $\mathcal{T}$ represents the set of transitions on the twin plant. $Next_{\mathcal{T}}(X)$ is the set of states reached from X by one transition of $\mathcal{T}$. $\mathcal{W}_k$ is the set of pairs $\langle q, q' \rangle (q' \in Q_N)$ reachable in k or less transitions. $\mathcal{R}_i$ is the set of pairs $\langle q, q' \rangle (q' \in Q_F)$ reachable in $(i+1)$ or less transitions from a pair of states of $\mathcal{W}_\infty$. S_j is the set of pairs reachable from $(j+1)$ or more transitions from a pair of states of $\mathcal{W}_\infty$. In summary, a twin plant, when implemented in the symbolic approach, is able to test the diagnosability of large DESs because the symbolic method avoids enumeration.

2.2.4 Diagnosability Testing using SAT

Rintanen and Grastien [2007] encoded the diagnosability testing as a SAT problem, and they used SAT solvers to test diagnosability. Rintanen and Grastien [2007] applied the twin plant method, and introduced definitions using SAT. States are pairs $(s, \hat{s})$ of states of the original transition system. Events represent unobservable events in one or both of the components of these state pairs, or observable events that are shared by both components. If there is an event sequence from $(s_0, \hat{s}_0)$ to some $(s, \hat{s})$, which includes a faulty event in the first component but not in the second, and there is a non-empty event sequence back to $(s, \hat{s})$ with no faults in the second component, then a pair of infinite event sequences witnessing non-diagnosability exists.

Rintanen and Grastien [2007] then combined the twin plant method and the formulation as a SAT problem in the classical propositional logic. The satisfiable valuations of state sequences are: $[(s_0, \ldots, s_n), (\hat{s}_0, \ldots, \hat{s}_n)]$ where $s_0 = \hat{s}_0$. Also, for $i \in \{0, 1, 2, \ldots, n-1\}$, $s_n = s_i$ and $\hat{s}_n = \hat{s}_i$. The corresponding event sequences are $[(E_0, \ldots, E_{n-1}), (\hat{E}_0, \ldots, \hat{E}_{n-1})]$ such that

$$\pi(E_0, \ldots, E_{n-1}) = \pi(\hat{E}_0, \ldots, \hat{E}_{n-1}).$$

Algorithm 4: Forward algorithm to test diagnosability

```
k := 0
W_k := I × I
repeat
    k := k + 1
    W_k := W_{k−1} ∪ ((Q_N × Q_N) ∩ Next_T(W_{k−1}))
until W_k := W_{k−1};
i := 0
R_i := (Q_N × Q_F) ∩ Next_T(W_k)
repeat
    i := i + 1
    R_i := R_{i−1} ∪ Next_T(R_{i−1})
until R_i = R_{i−1};
j := 0
S_j := R_i
repeat
    j := j + 1
    S_j := Next_T(S_{j−1})
until S_j := S_{j−1};
if S_j = ∅ then
    return diagnosable
else
    return not diagnosable
```

The formula for a given event sequence is satisfiable if and only if it is not possible to detect the occurrence of a faulty event. Following diagnosis using SAT as in Section 2.1.3, the diagnosability test is written as:

$$\Phi^{T}_{m,n} = \mathcal{T}_0 \wedge \mathcal{T}(0,1) \wedge \cdots \wedge \mathcal{T}(n-1,n) \wedge \bigvee_{t=0}^{n-1} \bigvee_{e \in \Sigma_f} \bigvee_{o \in \delta(e)} e^t_o \wedge \bigwedge_{a \in A} ((a^n \leftrightarrow a^m) \wedge (\hat{a}^n \leftrightarrow \hat{a}^m)) \tag{2.10}$$

In summary, the twin plant method proposed by Jiang et al. [2001] has relied on explicit enumeration of the states and are only feasible for a small number of states. Although SAT has the advantage to solve diagnosability problems, it is still subject to SAT-based techniques that the presence of short paths in transition graphs can be easily detected, but the absence of paths is often much more difficult to detect without length restrictions.

2.3 Sensor Minimisation and Dynamic Observers for Timed Systems

In the case of static observers, a set of observable events do not change during the execution of a system. This in turn determines the set of unobservable events. Thus, observing an event usually requires some detection mechanism, which brings the design questions of which sensors to use, how many of them, and where to place them. They are often difficult to answer, especially without knowing what these sensors are to be used for.

However, using a fixed set of observable events, observing events is costly in terms of time and energy. Firstly, it consumes time to read and process the information provided by the sensors. Secondly, it requires power for the operation of the sensors and computation of the data from the sensors. Consequently, it is essential to ensure that the sensors in use are necessary, and provide useful information. It is also important to discard any information given by a sensor that is not needed. In the case of a fixed set of observable events, not all sensors always provide useful information. Also, energy for sensor operation and data computation is sometimes spent for nothing.

The sensor minimisation problem focuses on dynamic observers for fault diagnosis of timed systems [Brandán-Briones et al., 2008; Cassez, 2010]. In particular, Cassez [2010] studied dynamic observers for a timed automaton. The sensor minimisation problem does not consider the problems of sensor placement, or choosing between different types of sensors. It is reasonable to follow the DES setting where the behaviours of a plant are known and a model is available as a finite state automaton over $\Sigma \cup \{\epsilon, f\}$ where Σ is a set of observable events of a DES, and ϵ is a single unobservable event to represent invisible actions. f is a special unobservable event that corresponds to a fault. Finally, the aim is to test diagnosability in polynomial time for a given plant and a fixed set of observable events.

2.3.1 Definitions and Examples

This section reviews the definitions of Cassez [2010] for timed word, un-timed word, clock valuation and timed automaton. Firstly, a *timed word*, denoted as w, is one or more events with time descriptions. For example, $0.4\,a\,1.0\,b\,2.7\,c$ is a timed word. The numbers are the time elapsed between two events. Secondly, $Unt(w)$ represents the un-timed version of w obtained by erasing all durations. For example, $Unt(0.4\,a\,1.0\,b\,2.7\,c) = abc$. Thirdly, *clock valuation* means adding a time constraint to each state and transition in an automaton. Finally, a *timed automaton* (TA) is a finite

automaton extended with real-valued clocks to specify timing constraints between occurrences of events. A timed automaton TA is a tuple $(L, l_0, X, \Sigma_{\mathcal{T}}, E, Inv, F, R)$ where

- L is a finite set of locations;
- l_0 is the initial location;
- X is a finite set of clocks;
- $\Sigma_{\mathcal{T}}$ is a finite set of actions;
- $E \subseteq L \times C(X) \times \Sigma_{\mathcal{T}} \times 2^X \times L$ is a finite set of transitions where $C(X)$ is a set of conjunctions of constraints of the form $x \bowtie c$ with $c \in \mathbf{Z}$ and $\bowtie \in \{\leq, <, =, >, \geq\}$; for $(l, g, a, r, l') \in E$,
  - g is a guard;
  - a is an action;
  - r is a reset value;
- $Inv \subseteq C(X)^L$ associates an invariant with each location;
- $F \subseteq L$ is the final set of locations;
- $R \subseteq L$ is the repeated set of locations.

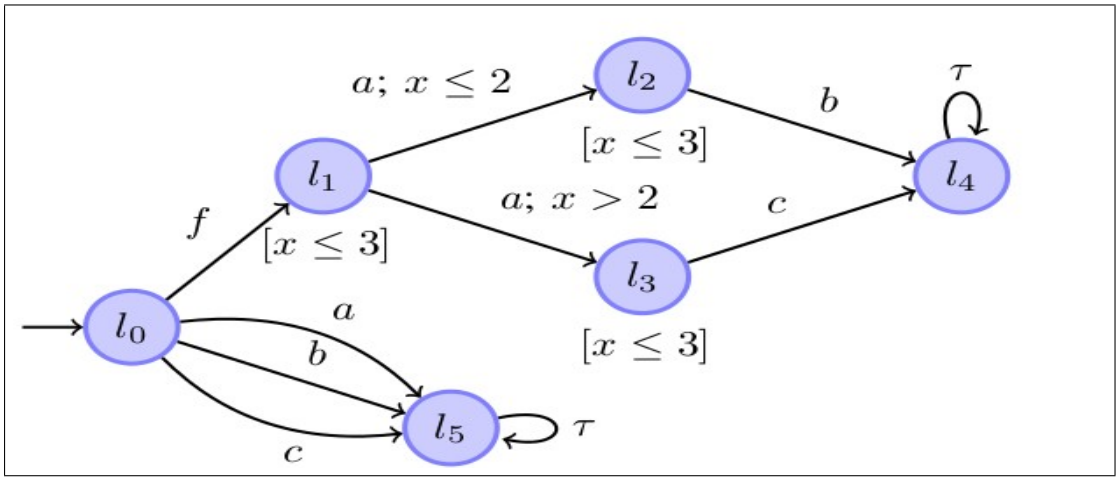

Figure 2.9: A timed automaton TA

Fig. 2.9 is an example of timed automaton. $[x \leq 3]$ is an invariant for the state l_1. The transition from l_1 to l_2 has a guard $x \leq 2$ and an action a. Although reset is not shown in this example, the clock can be re-initialised to a certain value.

A state of A is a pair $(l, v) \in L \times R^X_{\geq 0}$. $R_{\geq 0}$ is the set of positive rational numbers. A run r of A from (l_0, v_0) is a finite or infinite sequence of alternating delay δ and discrete move a, i.e.

$$r = (l_0, v_0) \xrightarrow{\delta_0} (l_0, v_0 + \delta_0) \xrightarrow{a_0} (l_1, v_1) \ldots \xrightarrow{a_{n-1}} (l_n, v_n) \xrightarrow{\delta_n} (l_n, v_n + \delta_n) \ldots \tag{2.11}$$

$$\text{such that for every } i \geq 0 : v_i + \delta \models Inv(l_i) \text{ for } 0 \leq \delta \leq \delta_i \tag{2.12}$$

$$\text{there is some transition } (l_i, g_i, a_i, r_i, l_{i+1}) \in E \tag{2.13}$$

$$\text{such that } v_i + \delta_i \models g_i \text{ and } v_{i+1} = (v_i + \delta_i)[r_i] \tag{2.14}$$

A finite timed word w is accepted by TA if the trace of a run of TA ends in a final location. A timed automaton TA is deterministic if there is no ϵ labelled transition in TA, and when the same action may occur, the guard must be different. ϵ means no event is observed, but a new state is entered.

2.3.2 Sensor Minimisation

The sensor minimisation problem has three aspects. First, it requires to decide whether there is a subset of observable events such that faults can be detected by observing only events in Σ_o. It also needs to find an optimally minimum Σ_o.

Second, the sensor minimisation problem allows a diagnoser to raise a fault not necessarily immediately after it occurs, but possibly some time later, as long as this time is bounded by some $k \in \mathbf{N}$. Time is modelled by counting the moves that a plant makes including observable and unobservable ones.

Third, it needs to address the masking issue. There are cases where two events are observable but not distinguishable. For example, a and b are observable; when a diagnoser knows that a or b occurred, it does not know which of the two. This is not the same as considering a and b to be unobservable, since in that case a diagnoser would not be able to detect the occurrence. Distinguishability of events is captured by the notion of a mask. A *mask* (M, n) over Σ is a total and surjective function:

$$M : \Sigma^* \to \{1, 2, 3, \ldots, n\} \cup \{\epsilon\}.$$

Then, morphism is defined as:

$$M^* : TW^*(\Sigma) \rightarrow TW^*(\{1, 2, 3, \ldots, n\})$$

where $TW^*(\Sigma)$ is the set of finite timed words over Σ.

For example,

$$\text{if } \Sigma = \{a, b, c, d\}, n = 2, M(a) = M(b) = 1, M(c) = 2, M(d) = \epsilon,$$

$$\text{then } M^*(a.b.c.b.d) = 1.1.2.1 = M^*(a.a.c.a).$$

2.3.3 Dynamic Observers

A dynamic observer modifies the set of events that it wishes to observe during a system execution.

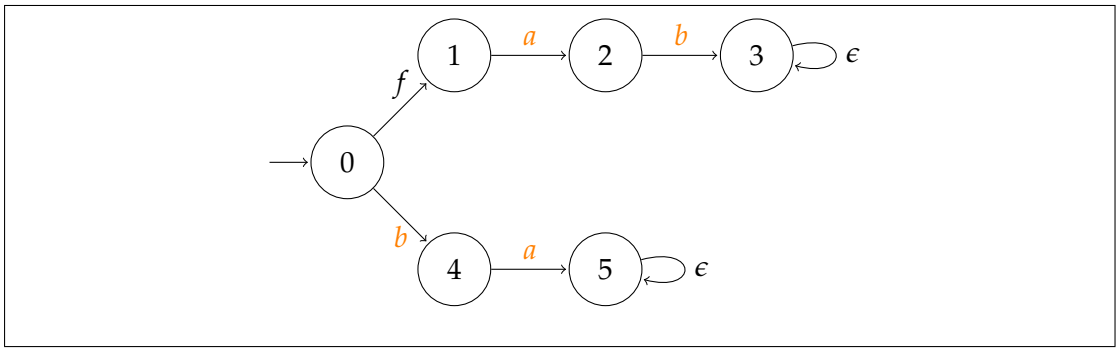

Figure 2.10: DES model B

Fig. 2.10 show the DES model B. Suppose the target is to detect faults in B. A static diagnoser that observes $\Sigma = \{a, b\}$ works. However, no proper subset of Σ can be used to diagnose the faults in B. Thus, the minimum cardinality of the set of observable events for diagnosing B is 2., i.e. a static observer will have to monitor two events during the execution of this DES model. If a mask is used, the minimum cardinality for a mask is 2 as well. Therefore, an observer will have to be receptive to at least two inputs at each point in time to detect a fault in B.

In contrast, it is more efficient to use a dynamic observer, which only switches on sensors when needed. In the DES model B, a diagnoser only switches on the a-sensor at the beginning; once a occurs, the a-sensor is switched off, and the b-sensor is switched on. Compared to the static diagnoser, a dynamic observer uses half as much energy. In general, switching sensors on and off leads to energy saving.

2.3.4 Complexity of Sensor Minimisation and Dynamic Observers

Cassez [2010] showed that the complexity of minimising the number of observable events is NP-complete. Membership in NP can be derived by reducing these problems to the standard diagnosability problem, once a candidate minimal solution is chosen non-deterministically. Additionally, Cassez [2010] defined dynamic observer synthesis problem, and showed the reduction of computing a dynamic observer for a given DES model to a game problem. Cassez et al. [2007] explained how to reduce dynamic diagnosability to a Büchi Game Problem, which is a trace-based winning strategy with partial observations. Therefore, game theory is an approach to solve the problem of partial observations.

2.4 Summary

Section 2.1 reviews the existing work of on-line diagnosis of DES. Section 2.2 reviews the definition of an important property of a DES model, i.e. diagnosability, and examines the established approaches to test the diagnosability of a DES model. Section 2.3 reviews the existing work on the sensor minimisation problem and dynamic observers for fault diagnosis on a timed system. Sensor minimisation problem is to minimise observers in operation using a fixed set of observable events. Dynamic observers aim to switch sensors on or off, in order to dynamically change the set of events to observe. The complexity of both problems are analysed and proved by Cassez [2010].

Faults can be defined at the event level, or the state level. Traditionally faults are defined as specific events. Jéron et al. [2006] defined faults using event patterns. Both formalisms are equally expressive, and this work considers faults at the state level for simplicity.

The on-line DES diagnosis problem was initially studied by Sampath et al. [1995]. Given a flow of observable events generated by a system, the problem consists in determining whether the DES is operating normally or not, based on the behavioural model of the system. This work uses the term *belief state* of the system to describe this computation. A belief state represents the set of global states that the system is possibly in after the given observations. The main challenge of on-line diagnosis is to deal with the complexity of a diagnostic algorithm that has to monitor the observable flow on the fly, and generate a succession of belief states that are consistent with the flow. In fact, the difficulty is that the number of the belief states has been proved to be exponential w.r.t. the number of system states [Rintanen, 2007]. The existing diagnostic algorithms attempt to compute at any time a belief state that is consistent

with the observable flow from the time when the system starts operating to the current time. The main drawback of such a conservative strategy is the inability to *follow* the observable flow for a large system due to the exponential size of the generated belief states. Also, the temporal complexity to handle all of the belief states remains a challenge.

Because diagnosis of DES is a hard problem, the use of faster diagnostic algorithms is inevitable. Such algorithms include the IWAs [Su and Grastien, 2013], which do not carry all of the historical information about the system execution, and chronicle-recognition diagnostic algorithms [Dousson, 1996], which uses pattern recognition techniques for diagnosis. However, these algorithms may be imprecise to diagnose a diagnosable system than using an exact model-based diagnostic algorithm, e.g. Sampath et al. diagnosis [Sampath et al., 1995]. Faults are very harmful to a system and expensive to recover from if not correctly diagnosed. Hence, it is essential to examine how to measure the quality of using a potentially imprecise diagnostic algorithm w.r.t. a diagnosable DES model.

In the literature, diagnosability of DES is an important property to measure the capability of a diagnostic system to identify faults. Diagnosability is well-known criterion of DES, which was initially proposed by Sampath et al. [1995]. Diagnosability of DES holds if using the model, a fault can always be diagnosed after it occurs. Furthermore, diagnosability testing has been a well-studied problem. Jiang et al. [2001] showed that proving non-diagnosability amounts to finding a critical witness, which is a pair of infinite executions on the model that are indistinguishable, i.e. they produce the same observations, one of which is faulty, and the other one is nominal [Jiang et al., 2001]. Thus, diagnosability is proved by showing that there is no such witness. This approach is known as the twin plant method. [Yoo and Lafortune, 2002] also proposed their approach to test the diagnosability by constructing a verifier.

Chapter 3 of this work extends the diagnosability of a DES model, and studies the precision of an imprecise diagnostic algorithms w.r.t. a DES model. A diagnostic algorithm is defined as *precise* w.r.t. a DES model if it always diagnoses the fault after it occurs. Chapter 3 proposes a novel approach to verify the precision of a diagnostic algorithm w.r.t. a DES model by constructing a *simulation*. Precision can be verified using known methods, such as the twin plant method by Jiang et al. [2001], on the condition that a simulation is built, which is a modified model that *simulates* how a diagnostic algorithm runs on a given DES model. The precision holds iff there is no critical witness in the synchronisation of the DES model with the simulation. This work also illustrates how to construct the simulation for chronicle-recognition diagnosis in Chapter 3, and for IWAs in Chapter 4.

As one is able to verify the precision of a diagnostic algorithm w.r.t. a DES model by building a simulation, this work proposes a new class of diagnostic algorithms called Independent-Window Algorithms (IWAs) in Chapter 4. IWAs are window-based diagnostic algorithms, which slice a sequence of observations into time windows, and diagnose them independently. This approach differs from the conservative strategy by proposing diagnostic algorithms that are only applied on the very last events of the observable flow, and *forget* about the past. IWAs slice an observation sequence into time windows so that each time windows are diagnosed independently. IWAs diagnose a certain number of observations for one time window, and move to another time window without keeping any information.

On the other hand, IWAs may cause imprecise diagnosis. Since IWAs diagnose time windows independently, imprecision happens when both current and past observations are necessary to understand the system behaviour. Chapter 4 demonstrates the construction of a simulation in order to verify the precision an IWA w.r.t. a DES model.

This work further proposes Time-Window Algorithms (TWAs) in Chapter 5. TWAs are inspired by IWAs. Chapter 5 formally presents two TWAs, namely, Al_5 and Al_6. TWAs remember a certain knowledge between the time windows so that the precision of diagnosis is preserved. The strategy of TWAs is compromising between the two extreme strategies of exact and imprecise diagnosis, e.g. a compromise between Sampath et al. [1995] and IWAs [Su and Grastien, 2013]. Such a compromise is achieved by looking for the minimum piece of information to *remember* from the past, called *abstracted belief state*, so that a window-based algorithm will certainly ensure the same precision as using an exact diagnostic algorithm. Chapter 5 also demonstrates how to verify the precision of each TWA w.r.t. a DES model by constructing a simulation.

Window-based diagnosis, i.e. IWAs and TWAs, are inspired by the sensor minimisation problem studied by Cassez [2010]. The intuition of the sensor minimisation problem is that the information from the sensors is not always useful. This work studies window-based diagnosis such that a diagnoser does not have to maintain all the information it has about a system. In particular, Al_6 is inspired by the preference of diagnosis as defined by Grastien and Anbulagan [2009], and involves refinement of diagnostic results of the time windows.

Chapter 3

Verifying the Precision of a Diagnostic Algorithm for DES

Model-based diagnosis of DES aims at deciding whether a system is running normally or is experiencing faulty behaviours. A diagnoser uses a DES model to search for system executions that are consistent with the observations. This work concentrates on finite state machines as the representation for a DES model. Section 2.1 of Chapter 2 examines that many previous works address the problem of DES diagnosis by the use of a diagnoser automaton [Sampath et al., 1995; Rozé and Cordier, 2002a], automata unfoldings [Baroni et al., 1999] that can also be distributed as in Pencolé and Cordier [2005]; Su and Wonham [2005], or encoded with Binary Decision Diagrams (BDD) as in Schumann et al. [2010], and finally by the use of a satisfiability (SAT) solver as in Grastien et al. [2007].

This work uses the term *belief state* to describe a set of states that a system is possibly in at a given time. The research challenge is to deal with the complexity of a diagnostic algorithm that has to monitor the observable flow on the fly, and generate a succession of belief states consistent with the flow. However, the difficulty is to maintain the belief state for the system. In fact, the size of a belief state in a Sampath et al. diagnoser has been proved to be exponential w.r.t. the number of system states [Rintanen, 2007].

Any of the precited work, except diagnosis using SAT, proposes diagnostic algorithms that attempt to compute at any time a belief state that is consistent with the observable flow from the time when the system starts operating to the current time. The main drawback of such a conservative strategy is the inability to *follow* the observable flow for a large system due to the exponential size of the generated belief states, and therefore the temporal complexity to handle them. Although diagnosis using SAT computes one trace in the system for an observation sequence, the complexity of a SAT problem is exponential to the number of propositional variables, which is linear to the number of state variables [Grastien et al., 2007].

Because diagnosis of DES is a hard problem, the use of faster diagnostic algorithms is often inevitable. Such algorithms include the IWAs [Su and Grastien, 2013], which do not carry all of the historical information about the system execution, and the chronicle-recognition diagnostic algorithm [Dousson, 1996], which uses pattern recognition techniques for diagnosis. Notice that, as opposed to the original work on chronicles by Dousson [1996], this work defines the time constraints in terms of the number of observations between two observed events, rather than the actual time between the event occurrences.

However, those algorithms may be imprecise to diagnose a diagnosable system than using an exact model-based diagnostic algorithm, e.g. Sampath et al. diagnosis [Sampath et al., 1995]. Faults are very harmful to a system and expensive to recover from if not correctly identified. Therefore, it is essential to measure the quality of using a potentially imprecise diagnostic algorithm w.r.t. a diagnosable DES model. This work focuses on a well-known criterion of *diagnosability*. Diagnosability is the property of an observable system, which states that, using the diagnosis model, a fault can be diagnosed after it occurs [Sampath et al., 1995]. Therefore, diagnosability is a property to measure the quality of diagnosis and the capacity of a diagnostic system to identify faults.

Verifying the diagnosability of DES has been a well-studied problem. Section 2.2 of Chapter 2 examines the two methods to verify the diagnosability of DES in polynomial time. Jiang et al. [2001] showed that proving non-diagnosability amounts to finding a critical witness, which is a pair of infinite executions on the diagnosis model that are indistinguishable, i.e. they produce the same observations, one of which is faulty, and the other one is nominal [Jiang et al., 2001]. Thus, diagnosability is proved by showing that there is no such witness. This approach is known as the *twin plant* method. Also, Yoo and Lafortune [2002] proposed the *verifier* method to verify the diagnosability of DES in polynomial time.

This chapter is inspired by DES diagnosability, and defines the *precision* of a diagnostic algorithm w.r.t. a DES diagnosis model. A diagnostic algorithm is *precise* if the algorithm will diagnose the fault after it occurs. Precision can be verified using known methods, such as the twin plant method by Jiang et al. [2001], on the condition that a *simulation* is built, i.e. a modified model that *simulates* how a diagnostic algorithm works on a given DES diagnosis model. This chapter proposes a method for deciding whether an imprecise algorithm preserves the precision of diagnosing a diagnosable DES model by constructing a simulation. A simulation is a finite state machine that represents how a diagnostic algorithm works for a DES diagnosis model. Precision is preserved iff no critical witness exists in the synchronisation of the system and the simulation. Finally, this work illustrates the construction of the

simulation for chronicle-recognition diagnosis in this chapter, and the simulation for IWAs in Chapter 4.

This chapter is organised as follows. Section 3.1 provides the definitions of DES model, exact diagnosis, and diagnosability of DES. Section 3.2 defines the generic notion of a diagnostic algorithm and the question of proving that an algorithm is precise. Section 3.3 reviews chronicle-recognition diagnosis, and illustrates its simulation. Section 3.4 summarises the impact of the simulation method, and concludes this chapter.

3.1 Diagnosis and Diagnosability of DES

This section provides the definitions for diagnosis and diagnosability of DES. It also reviews how to verify the diagnosability of a DES diagnosis model.

3.1.1 Definition for Diagnosis of DES

This work is based on the model-based diagnosis framework of *DES* [Cassandras and Lafortune, 2008]. In general, a state of a DES is a description for the system behaviour at an instant in time. A state in a DES may be changed by an occurrence of a discrete event depending on the DES specification. This work uses an *automaton* as a representation for DES, which is defined as follows.

Definition 1 (Automaton) *Let* $\mathbb{L} = \{N, F\}$ *be a set that represents the two modes of nominal (N), or faulty (F). An* automaton *is a tuple* $M = \langle Q, \Sigma, T, I, L \rangle$ *where*

- Q *is a finite set of* states*;*
- Σ *is a finite set of* events*;*
- $T \subseteq Q \times \Sigma \times Q$ *is a set of* transitions*;*
- $I \subseteq Q$ *is the set of* initial states*;*
- $L : Q \to \mathbb{L}$ *is a function that assigns to each state a* mode *that is either N, or F.*

The diagnosis model has a set of observable events $\Sigma_o \subseteq \Sigma$*, and a set of unobservable events* $\Sigma \setminus \Sigma_o$*.*

$L^{-1} : \mathbb{L} \to 2^Q$ *denotes the reverse function of L.*

Fig. 3.1 depicts a DES diagnosis model M_1, which is a visualisation for the states, the initial state, the transitions, the events, and the results of the mode label function L: $L(H) = L(F) = L(G) = F; L(B) = L(A) = L(C) = L(D) = N$. In other words,

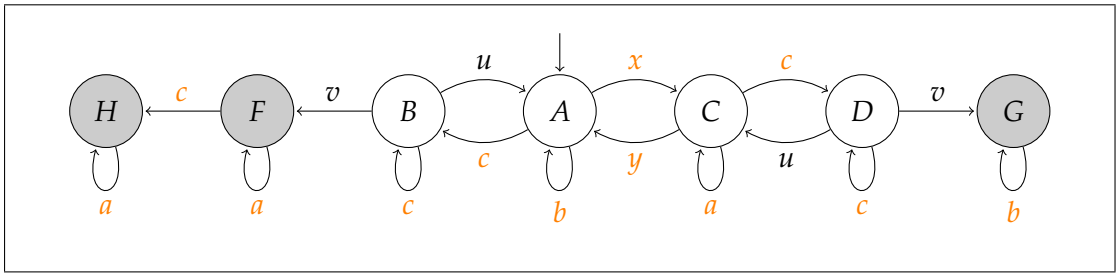

Figure 3.1: A simple DES model M_1 where a, b, c, x, y are observable events, and u, v are unobservable events

states H, F, G are faulty, and the other states are nominal. State A is the unique initial state. Finally, a, b, c, x, y are observable events, and u, v are unobservable events.

This work makes the following assumptions on the DES diagnosis models that will be studied.

Assumption 1 (No Cycle of Unobservable Events) *A DES studied in this work does not contain any cycle of unobservable events.*

Assumption 2 (Live System) *A DES studied in this work is live, i.e. a DES never reaches a point where no transition can take place.*

Assumption 3 (Permanent Fault) *Once a system is faulty, it remains faulty, i.e.*

$$\big(\langle q, e, q' \rangle \in T \wedge L(q) = F\big) \Rightarrow L(q') = F.$$

Because the diagnosis of a fault relies on the observations of a system, Assumption 1 enforces that a system does not generate a sequence of unobservable events that is infinitely long. Furthermore, Assumption 2 removes the limitation on the length of an observation sequence since an observable event is always possible take place in a system.

Faults can be defined at the event level, or the state level. Traditionally faults are defined as specific events. Jéron et al. [2006] defined faults using event patterns. For example, based on the café and library system model described in Fig. 2.1 of Chapter 2, Fig. 3.2 shows the event-based diagnosis model, and Fig. 3.3 shows the state-based diagnosis model. Both formalisms are equally expressive. For simplicity, this work considers faults at the state level. The implication of using faulty states is that a fault is permanent after its occurrence, as stated in Assumption 3.

Definition 2 (Trace) *A* trace *is a sequence, denoted as* $q_0 \xrightarrow{e_1} q_1 \ldots q_{k-1} \xrightarrow{e_k} q_k$, *such that* $\forall i \in [1, k]$, $\langle q_{i-1}, e_i, q_i \rangle \in T$. $\sigma = e_1 \ldots e_k$ *is a sequence of events. Any system behaviour is represented by a trace such that* $q_0 \in I$.

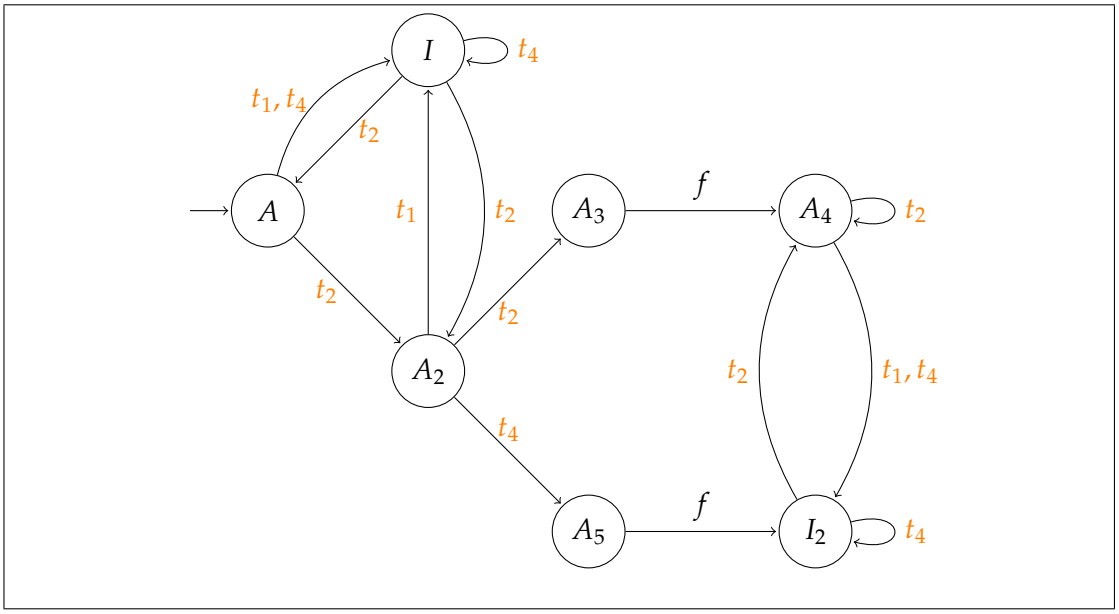

Figure 3.2: Event-based diagnosis model for the café and library model described in Fig. 2.1 of Chapter 2

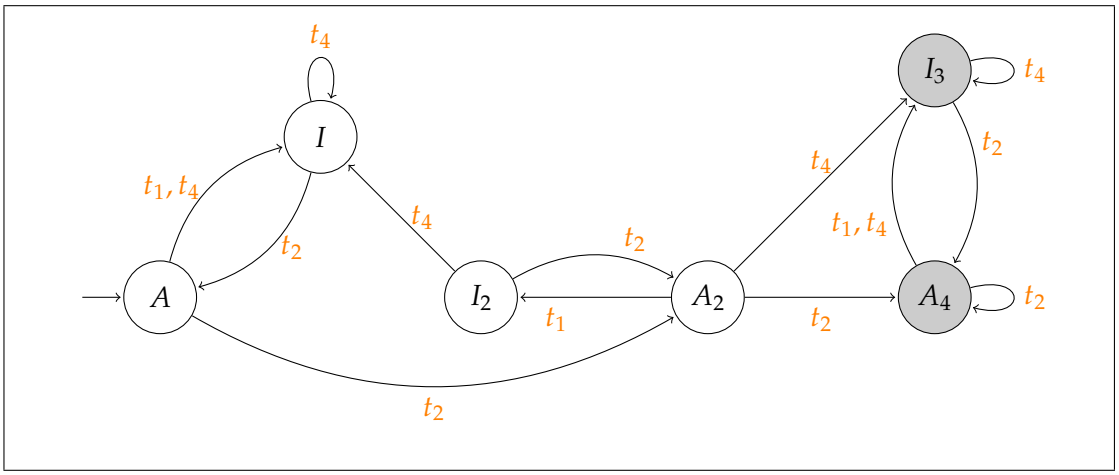

Figure 3.3: State-based diagnosis model for the café and library model described in Fig. 2.1 of Chapter 2

Definition 3 (Observation) *The observation* $obs(\sigma)$ *of* $\sigma \in \Sigma^*$ *is defined as the restriction of* σ *to the set of observable events, i.e.*

$$obs(e_1, e_2, e_3, \ldots, e_k) = \begin{cases} \varepsilon & \text{if } k = 0 \\ e_1, obs(e_2, e_3, e_4, \ldots, e_k) & \text{if } e_1 \in \Sigma_o \\ obs(e_2, e_3, e_4, \ldots, e_k) & \text{if } e_1 \in \Sigma \setminus \Sigma_o \end{cases}$$

This work also views a diagnosis model as a pair of languages $\mathcal{L} = \mathcal{L}_N \cup \mathcal{L}_F$ such that $e_1, e_2, e_3, \ldots, e_k \in \mathcal{L}_l$ ($l \in \{N, F\}$) iff there exists a trace $q_0 \xrightarrow{e_1} \ldots \xrightarrow{e_k} q_k$ where

$q_0 \in I$ and $L(q_k) = l$.

Definition 4 (Language $\mathcal{L}$) *The prefix-closed language $\mathcal{L} = \mathcal{L}_N \cup \mathcal{L}_F$ generated by the system is the set of sequences $\sigma = e_1, \ldots, e_k$ where $q_0 \xrightarrow{e_1} \ldots \xrightarrow{e_k} q_k, q_0 \in I$ is a trace. If $L(q_k) = N$, then $\sigma \in \mathcal{L}_N \subseteq \mathcal{L}$; otherwise, $\sigma \in \mathcal{L}_F \subseteq \mathcal{L}$.*

The diagnosis problem is defined based on the notion of *consistent trace*, which will be useful to define Time-Window Algorithms in Chapter 5.

Definition 5 (Consistent Trace) *The trace $q \xrightarrow{\sigma} q'$ is* consistent *with a sequence of observations θ if $obs(\sigma) = \theta$.*

Let $Q_0, Q_1, Q_2, \ldots, Q_p \subseteq Q$ be a collection of $(p+1)$ subsets of states, and let $\theta_1, \theta_2, \theta_3, \ldots, \theta_p$ be a collection of p sequences of observations. The consistent trace predicate, denoted as ct, is defined as follows.

Definition 6 (Consistent Trace Predicate ct) *The predicate $ct([Q_0, Q_1, Q_2, \ldots, Q_p], [\theta_1, \theta_2, \theta_3, \ldots, \theta_p])$ holds iff there exists a trace $q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \ldots \xrightarrow{\sigma_p} q_p$ such that $q_0 \in Q_0$, and for any $i \in \{1, 2, 3, \ldots, p\}$, it holds that $q_i \in Q_i$, $\sigma_i \in \Sigma^*$, and $obs(\sigma_i) = \theta_i$.*

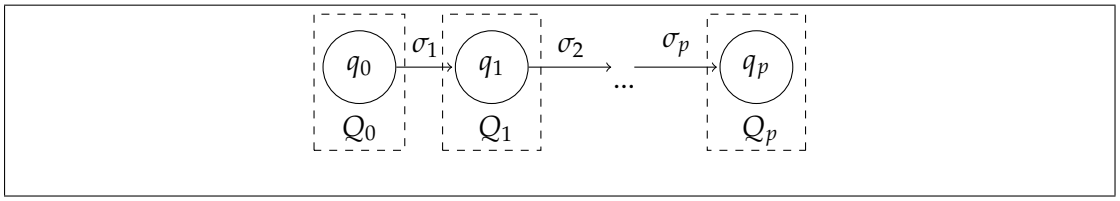

Figure 3.4: Visual representation for the ct predicate

Fig. 3.4 is a visual representation for the ct predicate. For any $i \in \{1, 2, 3, \ldots, p\}$, the system takes a trace $q_{i-1} \xrightarrow{\sigma_i} q_i$, which emits observation $\theta_i = obs(\sigma_i)$. The goal is to decide whether q_i is faulty. Notice that in Fig. 3.4, Q_i may contain one or more states.

A diagnosis problem is defined on a diagnosis model and an observation sequence. In general, a diagnoser should indicate whether the system is in the nominal mode, in the faulty mode, or it cannot decide, i.e. the system status is ambiguous. A diagnostic result is ambiguous if there are at least two explanations such that one is nominal, and the other is faulty. In this work, the diagnosis policy does not distinguish between the nominal mode and ambiguous mode, i.e. the diagnoser assumes that the system is not faulty unless proved otherwise.

Assumption 4 (Optimistic Diagnosis) *This work adopts an optimistic view for diagnosing a system such that as long as one nominal explanation of the observations can be found, the diagnoser assumes the system as nominal.*

Indeed, if the system is faulty, diagnosability ensures that a finite number of future observations is required to assert that no nominal explanation exists any more, i.e. the diagnoser will certainly assert that the system is faulty.

Exact diagnosis Δ by *language* is defined as follows.

Definition 7 (Exact Diagnosis Δ by *language*) *Given a diagnosis model M and a sequence of observations Θ, the* exact diagnosis $\Delta(M, \Theta)$ *is defined as follows.*

- $\Delta(M, \Theta) = N$ *if* $\exists \sigma \in \mathcal{L}_N.\ obs(\sigma) = \Theta$;
- $\Delta(M, \Theta) = F$ *otherwise.*

$\Delta(M, \Theta)$ *returns N if there exists a nominal trace whose observation is Θ. Otherwise, $\Delta(M, \Theta)$ returns F.*

Property 1 (Diagnosis by ct predicate) $\Delta(M, \Theta) = N \Leftrightarrow ct([I, L^{-1}(N)], [\Theta])$

Property 1 can be proved by Definition 6 and 7. This work proposes the definitions for the ct predicate and the exact diagnosis Δ, as they will be useful to define Time-Window Algorithms in Chapter 5.

Observation	Diagnoser	Diagnostic Result
x, y, x, y, b, c	Δ	N (precise)
b, x, c, c, c, b	Δ	F (precise)
c, c, c, c, c, c	Δ	N (precise)

Table 3.1: Example of diagnostic results for M_1 in Fig. 3.1

Tab. 3.1 shows the diagnostic results of M_1 in Fig. 3.1 with three observation sequences.

Example 1 of Δ Diagnosis Given the first observation sequence x, y, x, y, b, c, there exists only one trace from the initial state A that is consistent with the given observation sequence, i.e. $A \xrightarrow{x} C \xrightarrow{y} A \xrightarrow{x} C \xrightarrow{y} A \xrightarrow{b} A \xrightarrow{c} B$. Since $L(B)$ evaluates to N, this trace terminates in a state with a nominal label. Therefore, the diagnostic result for the first observation sequence is nominal.

□

Example 2 of Δ Diagnosis Given the second observation sequence, b, x, c, c, c, b, there exists only one trace from the initial state A that is consistent with the given observation sequence, i.e. $A \xrightarrow{b} A \xrightarrow{x} C \xrightarrow{c} D \xrightarrow{c} D \xrightarrow{c} D \xrightarrow{v} G \xrightarrow{b} G$. Since $L(G)$ evaluates to F, this trace terminates in a state with a faulty label. There is one faulty

explanation and no nominal explanation for this observation sequence. Therefore, the diagnostic result for the second observation sequence is faulty.

□

Example 3 of Δ Diagnosis Given the third observation sequence, c, c, c, c, c, c, there are two traces from the initial state A that are consistent with the given observation sequence. One trace is: $A \xrightarrow{c} B \xrightarrow{c} B \xrightarrow{c} B \xrightarrow{c} B \xrightarrow{c} B \xrightarrow{c} B$. Since $L(B)$ evaluates to N, this trace terminates in a state with a nominal label.

Notice that the other trace is: $A \xrightarrow{c} B \xrightarrow{c} B \xrightarrow{c} B \xrightarrow{c} B \xrightarrow{c} B \xrightarrow{v} F \xrightarrow{c} H$. Since $L(H)$ evaluates to F, this trace terminates in a state with a faulty label. There is a nominal explanation and a faulty explanation. Thus, the third observation sequence leads to an ambiguous diagnosis. Since there exists at least one trace that terminates in a state with a nominal label, the diagnostic result for the third observation sequence is nominal.

□

3.1.2 Definition for Diagnosability of DES

Diagnosability is an important property of diagnosing a DES model. It is a condition that system designers often want to enforce. Diagnosability holds if a fault will always be diagnosed [Sampath et al., 1995]. This property can be checked by searching for faulty traces that cannot be diagnosed precisely. Any such counterexample proves that the system is not diagnosable; failure to find such a counterexample proves that the system is diagnosable, assuming the search was complete.

In the diagnosability problem, a *diagnosis model* (M) and a *system model* (M') are needed [Grastien and Torta, 2011]. A system model (M') is a DES that captures how a system operates. Given an observation sequence, a diagnoser uses a diagnosis model (M), which is a DES, to diagnose the system status. Notice that the diagnosis model may be the same as the system model. In order to verify the diagnosability of a system model, a system model needs to generate a faulty trace. Then, the diagnoser tries to find a nominal trace in the diagnosis model that generates the same observation as the faulty trace. Finally, both models are synchronised so that a trace on the resulting twin plant represents two traces from each model generating the same observation.

Recall that Section 3.1.1 defines a diagnosis model as an automaton and also defines diagnosis at the language level. The diagnosability of DES is defined as follows.

Definition 8 (Diagnosability of DES) *A diagnosis model M is* diagnosable *w.r.t. a system model M' if the following holds:*

$$\exists n \in \mathbf{N}.\ \forall s \in \mathcal{L}'_F.\ \forall t \in \Sigma^*.\ \big(st \in \mathcal{L}' \wedge |t| \geq n \Rightarrow \Delta(M, obs(st)) = F\big)\,.$$

Based on the diagnosability definition of Sampath et al. [1995], this definition concentrates on a diagnosis model. Given a faulty trace s, a diagnosis model M, and further n events represented by t, the diagnosis of the observation $obs(st)$ produced by the trace of st should be faulty, i.e. the fault will always be detected within a finite delay.

Assumption 5 (Diagnosable DES Diagnosis Model) *A DES diagnosis model studied in this work is diagnosable.*

Assumption 5 will be useful for Section 3.2 to study the precision of a diagnostic algorithm. Since this work studies state-based diagnosis, a faulty event is a special case of unobservable events. Also, this work considers diagnosable systems. Therefore, each fault can be diagnosed separately. This work makes the single-fault assumption for simplicity purpose only.

Assumption 6 (Single Fault) *This work considers one fault in a system at any time.*

Section 2.2.2 of Chapter 2 examines that using automata, diagnosability can be checked in polynomial time w.r.t. the number of states [Jiang et al., 2001]. A twin plant is built, which is the classical automaton synchronisation of two copies of the model on the observable events. M is diagnosable w.r.t M' iff the twin plant contains no infinite cycle of states $\langle q, q' \rangle$ where $L(q)$ evaluates to N and $L(q')$ evaluates to F. A similar approach was proposed by Yoo and Lafortune [2002]. They presented polynomial-time algorithms to decide diagnosability by constructing a non-deterministic automaton called a verifier, which do not rely on any diagnosers. Furthermore, Section 2.2.3 examines how a twin plant can be implemented by the symbolic approach.

3.2 Precision of a Diagnostic Algorithm

This section presents a formal definition of diagnostic algorithms. Since a diagnostic algorithm may be imprecise, this section studies the precision criterion for a diagnostic algorithm. This work also shows how this criterion can be tested using the simulation method. The section ends with a comparison with the existing ad-hoc methods used in the literature to verify the precision of three diagnostic algorithms.

3.2.1 Definitions for Diagnostic Algorithms and Precision

This work defines diagnostic algorithms that may run faster than the exact diagnosis Δ. However, these algorithms may return results that are less precise. It is necessary to be able to give guarantees about the output of an algorithm, e.g. precision.

Definition 9 (Diagnostic Algorithm) *A* diagnostic algorithm *is a function* $A : M \times \Sigma_o{}^* \rightarrow \{N, F\}$ *where* M *is a diagnosis model, and the following conditions of monotonicity and correctness hold:*

Monotonicity: $A(M, o) = F \Rightarrow \forall e \in \Sigma_o.\ A(M, (oe)) = F$;

Correctness: $A(M, o) = F \Rightarrow \Delta(M, o) = F$.

The first condition ensures that the diagnosis is monotonic [Lamperti and Zanella, 2007], i.e. that if a fault has been diagnosed, this conclusion will not be withdrawn. The second condition ensures that the diagnosis is correct, i.e. that the algorithm returns faulty only when the system is faulty while the converse may not hold.

Based on Assumption 5, this work defines the precision of a diagnostic algorithm.

Definition 10 (Precision of a Diagnostic Algorithm) *The diagnostic algorithm* A *is* precise *for a diagnosis model* M *w.r.t. a system model* M' *if the following holds:*

$$\exists n \in \mathbf{N}.\ \forall s \in \mathcal{L}'_F.\ \forall t \in \Sigma^*.\ \left(st \in \mathcal{L}' \wedge |t| \geq n \Rightarrow A(M, obs(st)) = F\right).$$

This definition is similar to Definition 8 for diagnosability of DES. Since Definition 9 for a diagnostic algorithm does not give any constraint on the precision of diagnosis, Definition 10 now defines that a diagnostic algorithm is precise if the algorithm is able to return faulty (F) after a fault effectively occurs in the system, possibly after a finite delay.

3.2.2 Simulation

Section 3.1.2 reviews that the diagnosability of a DES model can be verified using the twin plant method by Jiang et al. [2001]. This work proposes to verify the precision of a diagnostic algorithm w.r.t. a DES model by constructing a *simulation*. A diagnostic algorithm is *precise* if the algorithm will diagnose the fault after it occurs. The precision of a diagnostic algorithm w.r.t. a DES model can be verified using known methods, such as the twin plant method by Jiang et al. [2001], *as long as* a simulation is built, i.e. a finite state machine that *simulates* how a diagnostic algorithm works for a given DES model. Precision is preserved iff no critical witness exists in the

synchronisation of the system and the simulation. Notice that if it is proved that a simulation cannot be built for a particular diagnostic algorithm w.r.t. a DES model, then this method is not applicable. In general, the simulation method provides a *theoretical framework*, which helps researchers to verify the precision of their diagnostic approaches. It also allows researchers to consider more aggressive approaches to reduce the complexity of diagnosis, as they can now assess the precision of those approaches.

Definition 11 (Simulation) *Given a diagnosis model M and a diagnostic algorithm A, the* simulation *of M and A is an automaton, denoted as $si(M, A)$, such that*

$$\forall o \in {\Sigma_o}^*,\ A(M, o) = \Delta(si(M, A), o).$$

Using the simulation, Theorem 1 proves the precision of a diagnostic algorithm w.r.t. a system model M'.

Theorem 1 (Precision of a Diagnostic Algorithm) *A diagnostic algorithm A for a diagnosis model M is precise w.r.t. a system model M' iff $si(M, A)$ is diagnosable w.r.t. M' where $si(M, A)$ is the simulation of M and A.*

Proof of Theorem 1 The proof consists of two parts. Let $si(M, A)$ be the simulation of M and A. The first part is prove that a diagnostic algorithm A for a diagnosis model M is precise w.r.t. a system model M' if $si(M, A)$ is diagnosable w.r.t. M'. Suppose $si(M, A)$ is diagnosable w.r.t. M'. Then, by Definition 8,

$$\exists n \in \mathbf{N}\ .\ \forall s \in \mathcal{L}'_F.\ \forall t \in \Sigma^*.\ \big(st \in \mathcal{L}' \wedge |t| \geq n \Rightarrow \Delta(si(M, A), obs(st)) = F\big)\ .$$

By Definition 11, $A(M, obs(st)) = \Delta(si(M, A), obs(st))$. Therefore,

$$\exists n \in \mathbf{N}\ .\ \forall s \in \mathcal{L}'_F.\ \forall t \in \Sigma^*.\ \big(st \in \mathcal{L}' \wedge |t| \geq n \Rightarrow A(M, obs(st)) = F\big)\ .$$

This means that A is precise w.r.t. M' by Definition 10. The first part is proved.

The second part is to prove that $si(M, A)$ is diagnosable w.r.t. M' if a diagnostic algorithm A for a diagnosis model M is precise w.r.t. a system model M'. Suppose a diagnostic algorithm A for a diagnosis model M is precise w.r.t. a system model M'. Then, by Definition 10,

$$\exists n \in \mathbf{N}\ .\ \forall s \in \mathcal{L}'_F.\ \forall t \in \Sigma^*.\ \big(st \in \mathcal{L}' \wedge |t| \geq n \Rightarrow A(M, obs(st)) = F\big)\ .$$

By Definition 11, $A(M, obs(st)) = \Delta(si(M, A), obs(st))$. Therefore,

$$\exists n \in \mathbf{N} \,.\, \forall s \in \mathcal{L}'_F.\, \forall t \in \Sigma^*.\, (st \in \mathcal{L} \wedge |t| \geq n \Rightarrow \Delta(si(M, A), obs(st)) = F)\,.$$

This means that $si(M, A)$ is diagnosable w.r.t. M' by Definition 8. The second part is proved.

□

Theorem 1 provides a novel approach of verifying the precision of a diagnostic algorithm w.r.t. a system model. Given the simulation of a diagnostic algorithm, one can compute the twin plant of the synchronisation of the system model and the simulation to search for ambiguous cycles. Such cycles are witnesses of system behaviours that are not diagnosable. Otherwise, the diagnostic algorithm is precise w.r.t. the system model.

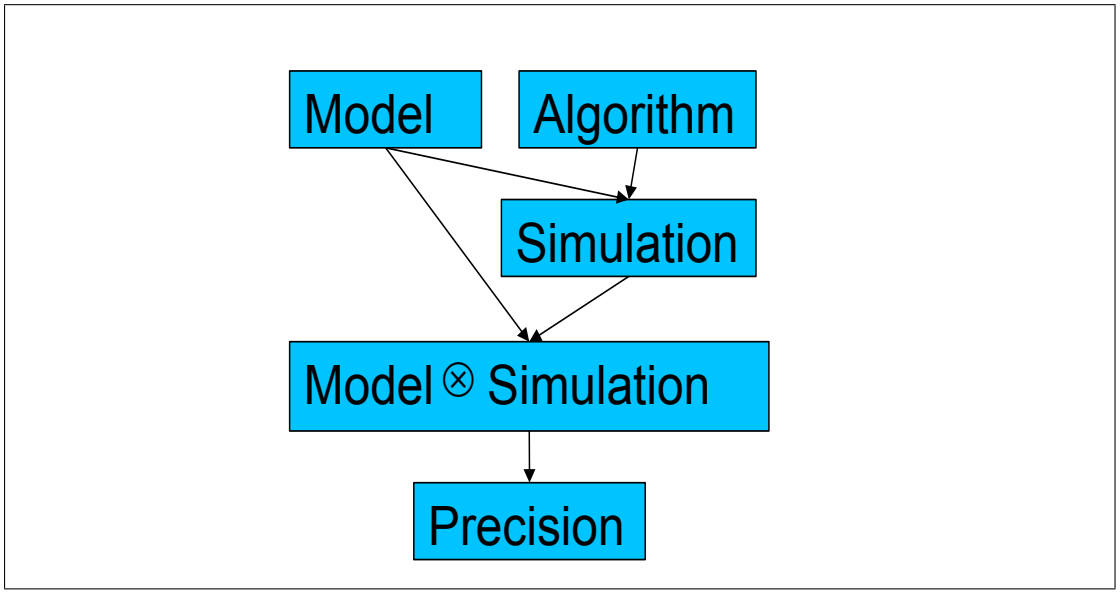

Figure 3.5: Visualisation for using a simulation to verify the precision of a diagnostic algorithm w.r.t. a DES system model where $\otimes$ represents automaton synchronization

Given a DES system model, and a diagnostic algorithm, Fig. 3.5 is the visualisation for using a simulation. Firstly, a simulation is constructed, which is a DES model that describes how the diagnostic algorithm works for the DES system model. Secondly, the DES system model and the simulation are synchronised, which is represented by $\otimes$. Thirdly, the twin plant method is applied to this synchronisation, which verifies the precision of the diagnostic algorithm w.r.t. the DES system model. If a nominal and a faulty behaviour can never be distinguished, then the diagnostic algorithm is not precise w.r.t. the DES system model. Otherwise, the diagnostic algorithm is precise w.r.t. the DES system model.

3.2.3 Related work

This section examines three cases of verifying the precision of diagnosis, which are distributed diagnosis, diagnosis using an *abstract model* of DES, as well as diagnosis using static and dynamic sensors. First, Kan John et al. [2010] argued that the complexity of distributed diagnosis of DES depends on the component connections. They removed unnecessary component connections to reduce the complexity, and addressed the problem of reduced precision by off-line analysis of which component connections can be safely removed. By Theorem 1, it is also feasible to verify the precision of distributed diagnosis by constructing a simulation for the distributed diagnostic process.

Second, Grastien and Torta [2011] proposed the theory of abstraction for DES. They studied how to build an *abstract model*, which is a model such that irrelevant details are removed. They also verified the diagnosability of an abstract model, which ensures the precision of the diagnosis using an abstract model. By Theorem 1, if diagnosis using an abstract model is simulated, then the precision of this diagnostic approach can also be verified.

Third, Cassez and Tripakis [2008] studied the sensor minimisation problems for both static and dynamic observers. For static observers, the goal is to minimise the amount of observable events. For dynamic observers, sensors can be switched on or off. As a result, the set of observable events changes over time. Furthermore, they considered masked observations such that some events are observable but not distinguishable. They also studied the diagnosability with static observers, dynamic observers, and masked observations. By Theorem 1, if a simulation is individually built for static observers, dynamic observers, and masked observations, then the simulation method is an alternative approach to verify the diagnosability. To sum up, it is feasible to verify the precision of the above three cases by constructing a simulation for each approach as defined in section 3.2.2.

3.3 Chronicle-Recognition Diagnosis

Chronicles are collections of events connected by temporal constraints on their occurrence time. In chronicle-recognition diagnosis, chronicles represent symptoms of failure [Dousson, 1996]. As opposed to the original work on chronicles by Dousson [1996], this work defines the time constraints in terms of the number of observations between two observed events, rather than the actual time between the event occurrences. This modified approach is named as chronicle-recognition diagnosis. Chronicles can be specified manually, or generated automatically from a diagnosis

model. A diagnoser is a system that recognises chronicles in the flow of observations. If a chronicle is recognised in the flow of observations, then the system is in the faulty mode. Otherwise, the system is in the nominal mode. Notice that this work does not study or assess chronicle-recognition diagnosis as a new diagnostic approach. Indeed, this work illustrates how the precision of chronicle-recognition diagnosis w.r.t. a DES model can be verified by constructing a simulation.

Diagnosability of chronicles is an important criterion. Pencolé and Subias [2009] proposed a method of verification, which computes the language associated with each chronicle, and checks whether these languages are exclusive. Diagnosability verification was carried out as exclusiveness tests on the reachability graphs of Time Petri Nets.

This section formally defines chronicle recognition, and demonstrates how to verify the precision of a chronicle w.r.t. a system model by building a simulation that represents how the chronicle-recognition diagnostic algorithm works on the system model. This section illustrates the *chronicle simulation* for a single chronicle. The extension to a set of chronicles can be achieved by computing the synchronisation of these simulations.

3.3.1 Definition of Chronicle Recognition

This work defines the fundamental concepts of *time interval*, *semantics of a time interval*, *time inconsistency*, three operations on time intervals, and *chronicle*. It also provides the definition for a chronicle being recognised in a sequence of observable events.

Definition 12 (Time Interval) *A* time interval *represents a set of numbers of observable event occurrences.* $\overline{\mathbf{Z}}$ *denotes* $\mathbf{Z} \cup \{+\infty, -\infty\}$. *A time interval is written as* $I = [beginning, end]$, *where* $beginning \in \overline{\mathbf{Z}}$, *and* $end \in \overline{\mathbf{Z}}$. $\mathbf{I}^+$ *is the set of strictly positive time intervals, and* $\mathbf{I}^-$ *is the set of strictly negative time intervals.* $\mathbf{I}$ *represents the set of time intervals.*

Definition 13 (Semantics of a Time Interval) *Given a time interval* $I = [beginning, end]$, *the* semantics *of* I, *denoted as* $Sem(I)$, *is defined as a list of integers ranging from beginning to end, i.e.* $Sem(I) = \{beginning, \ldots, end\}$.

This work also considers a special case of time interval, called *time inconsistency* ($I_\emptyset$), which is defined as follows.

Definition 14 (Time Inconsistency $I_\emptyset$) *Given a time interval* $I = [beginning, end]$, *if* $beginning > end$, *then* I *becomes* $I_\emptyset$ *meaning that no occurrence time for the associated event will satisfy the constraints on the time occurrence.*

Definition 15 (Operations on Time Intervals) *Three operations on time intervals are defined as follows.*

- *Time-interval intersection:*
  if $I = [b, e]$ *and* $I' = [b', e']$, *then* $I \cap I' = [max(b, b'), min(e, e')]$;
- *Time-elapsed operation: if* $I = [b, e]$, *then* $I - t = [b - t, e - t]$;
- *Time-disabled operation:* $I \setminus [b', e']$ *means that* $[b', e']$ *is taken out of the time interval* I.

The time-disabled operation may result in disjunctive time intervals. In this work, however, it will be applied in situations where this does not happen.

Dechter et al. [1991] proposed the Temporal Constraint Satisfaction Problem (TCSP) model, which includes a set of event time points, unary constraints, and binary constraints. A *unary constraint* restricts the domain of an event time point. A *binary constraint* restricts the distance between the time points of two events. In particular, the Simple Temporal Problem (STP) model is a TCSP such that each constraint has a single time interval. An instance of STP can be represented by a *directed edge-weighted graph*, or a *time constraint graph* [Dechter et al., 1991]. In a time constraint graph, each node represents an event, and each edge represents the distance between the time points of two events. In order to ensure the consistency of all constraints, *Floyd-Warshall*'s all-pairs-shortest-paths-algorithm is applied to a time constraint graph. The computational complexity is $O(n^3)$ where n is the number of nodes [Dechter et al., 1991].

Notice that as opposed to the original work on chronicles [Dousson, 1996], this work defines the time constraints in terms of the number of observations between two observed events, rather than the actual time between the event occurrences. This work formulates chronicle-recognition diagnosis using the STP model, and formally defines *chronicle* as follows.

Definition 16 (Chronicle) *A* chronicle *is a tuple* $\langle N, EL, B \rangle$ *where*

- N *is a finite set of* nodes *in a STP model;*
- $EL : N \rightarrow \Sigma$ *is an* event label function*;*
- $B : N \times N \rightarrow \mathbf{Z} \times \mathbf{Z}$ *is a* binary constraint function *for a pair of nodes.*

A chronicle is recognised in a sequence of observations iff there exists a recognition function f_{CR} that indexes each node of the chronicle with an event of the flow, and these events satisfy all constraints. Chronicle-recognition diagnosis checks

whether the chronicle is recognised in the observations. If it is recognised, then the chronicle-recognition diagnoser returns that the system is faulty; otherwise, it returns that the system is nominal. Chronicle recognition is formally defined as follows.

Definition 17 (Chronicle Recognition) *Given a chronicle* $Ch = \langle N, EL, B \rangle$*, and a sequence of observations* $o_1, o_2, o_3, \ldots, o_k$*, the chronicle is recognised in the sequence of observations iff* $\exists f_{CR} : N \rightarrow \{1, 2, 3, \ldots, k\}$ *such that* $\forall n \in N,\ \forall n' \in N,\ (EL(n) = o_{f_{CR}(n)}) \wedge (f_{CR}(n') - f_{CR}(n) \in B(\langle n, n' \rangle)) \wedge (n \neq n' \Rightarrow f_{CR}(n) \neq f_{CR}(n'))$.

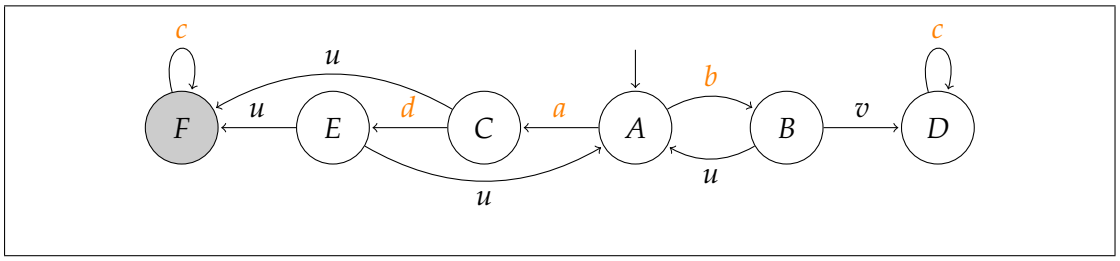

Figure 3.6: A simple DES model M_2 where a, b, c, d are observable events, and u, v are unobservable events

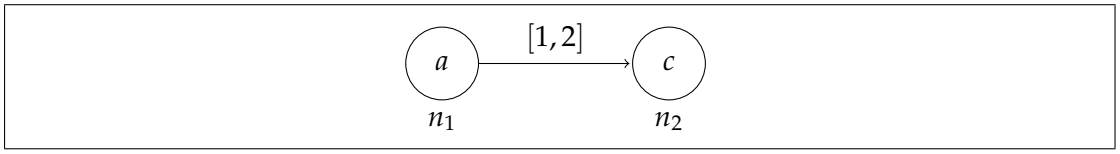

Figure 3.7: Chronicle 1 (Ch_1) for M_2 represented by a time constraint graph: if there is a c event within one or two time steps after an a event, then M_2 is faulty

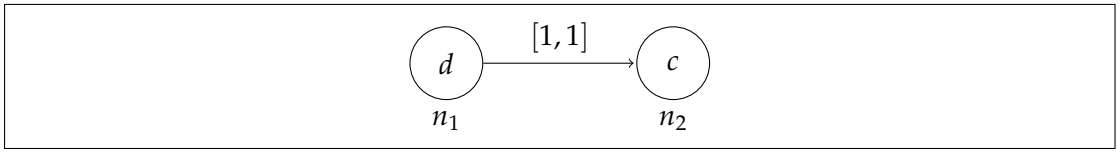

Figure 3.8: Chronicle 2 (Ch_2) for M_2 represented by a time constraint graph: if there is a c event within one time step after a d event, then M_2 is faulty

Fig. 3.6 shows a simple DES diagnosis model M_2. State F is faulty while other states are nominal. Observable events are a, b, c, d, and unobservable events are u, v. Fig. 3.7 and Fig. 3.8 show chronicle 1 (Ch_1) and chronicle 2 (Ch_2), each represented by a time constraint graph. Ch_1 and Ch_2 are two chronicles for M_2. Ch_1 specifies that if there is a c occurrence within two events after an event of a, then the chronicle is recognised and the system is faulty. Ch_2 specifies that if there is a c occurrence right after an event of d, then the chronicle is recognised and the system is faulty. Section 3.3.2 will examine the precision of Ch_1 and Ch_2 w.r.t. M_2.

Example 1: Chronicle Recognition of Ch_1 Given a DES diagnosis model M_2 and an observation sequence a, c, c, c, Ch_1 is recognised because there is a c occurrence within two events after an event of a. Thus, the diagnostic result is that the system is faulty.

□

Example 2: Chronicle Recognition of Ch_1 Given a DES diagnosis model M_2 and an observation sequence a, d, b, c, Ch_1 is not recognised because there is no c occurrence within two events after an event of a. Thus, the diagnostic result is that the system is nominal.

□

Example 3: Chronicle Recognition of Ch_2 Given a DES diagnosis model M_2 and an observation sequence a, c, c, c, Ch_2 is not recognised because there is no d occurrence. Thus, the diagnostic result is that the system is nominal. However, the system should be faulty according to the exact diagnosis of Δ. This example shows that Ch_2 is not precise to diagnose M_2. Section 3.3.2 will study the precision of a chronicle w.r.t. a DES system model.

□

Example 4: Chronicle Recognition of Ch_2 Given a DES diagnosis model M_2 and an observation sequence a, d, b, c, Ch_2 is not recognised because there is no c occurrence within exactly one event after an event of d. Thus, the diagnostic result is that the system is nominal.

□

3.3.2 Chronicle Automaton and Chronicle Simulation

This section shows how to verify the precision of a chronicle w.r.t. a DES system model. The approach is to compute a simulation for a chronicle, and the construction requires two steps. The first step is to build a non-deterministic finite state machine, called *chronicle automaton*. Then, the simulation will be the determinisation of such a chronicle automaton. Each state of a chronicle automaton stores a list of observable events of the chronicle that have been recognised so far, and when the next events are expected to occur. This work defines Partially Recognised Chronicle ($PRCh$), and Fully Recognised Chronicle ($FRCh$) as follows.

Definition 18 (Partially Recognised Chronicle ($PRCh$**))** *A Partially Recognised Chronicle (PRCh) is a tuple* $\langle Ch, U \rangle$ *where*

- *Ch is a chronicle* $\langle N, EL, B \rangle$*;*
- $U : N \rightarrow \{I_R, I_\emptyset\} \cup \mathbf{I}^+$ *is a* unary constraint function *for a node, which defines a time interval that specifies when a node is expected to be recognised.*

If $U(n) \in \mathbf{I}^-$, then the observation $EL(n)$ has been made. Precisely when this observation has been made is no longer relevant assuming the implication on occurrences of the other events have been stored in U. Consequently, $U(n)$ is then replaced by I_R, i.e. $(-\infty, 0]$.

Definition 19 (Fully Recognised Chronicle (*FRCh*)) *A Fully Recognised Chronicle (FRCh) is a partially recognised chronicle* $\langle Ch, U \rangle$ *such that* $\forall n \in N,\ U(n) = I_R$.

Definition 20 (Chronicle Automaton (*CA*)) *Given a PRCh G such that* $\forall n \in N,\ U(n) = [1, +\infty)$*, the* chronicle automaton *(CA) of G is an automaton, which is a tuple* $\langle Q, \Sigma, T, I, L \rangle$ *where*

- Q *is a set of PRCh;*
- Σ *is a set of* events*;*
- $T \subseteq Q \times \Sigma \times Q$ *is the set of* transitions *where each transition is built using Algorithm 5;*
- $I = \{G\}$*;*
- $\forall q \in Q,\ L(q) = F$ *iff q is a FRCh; otherwise,* $L(q) = N$.

In the initial state I, the unary constraint function labels every node with $[1, +\infty)$, which represents an unknown time in the future since the chronicle recognition process has not started yet. The function L labels the final states. The faulty states are *FRCh* such that $\forall n \in N,\ U(n) = I_R$.

The function presented in Algorithm 5 is to build a transition from *PRCh* to *PRCh′* given an observable event e. Starting with the initial state, a chronicle automaton is built by iteratively computing the successors of all the *PRCh* found. Algorithm 5 first acknowledges that a new observation has been made by reducing all intervals by one, which means that one time step has passed (Line 1). Next, if a node is recognised at this moment, this algorithm enforces the current time step ($[0, 0]$) using a time interval intersection operation (Line 2–3). Otherwise, the node is not recognised, and $[0, 0]$ is disabled using the time-disabled operation (Line 5). Then, the Floyd-Warshall's algorithm (*update_constraint*) is applied to ensure the consistency of all constraints [Dechter et al., 1991] (Line 6). If a node has been recognised,

Algorithm 5: build_one_transition

Input: $PRCh(\langle Ch, U\rangle)$ and e
Output: $PRCh'$

1 $U_1 := U - 1$
2 $U_2 : N \to \mathbf{I}$ such that $\forall n \in N$, **if** $EL(n) = e$ **then**
3 $\quad U_2(n) := U_1(n) \cap [0,0]$
4 **else**
5 $\quad U_2(n) := U_1(n) \setminus [0,0]$
6 $U_3 := update\_constraint(U_2)$
7 $U_4 : N \to \mathbf{I}$ such that $\forall n \in N$, **if** $U_3(n) \in \mathbf{I}^-$ *or* $U_3(n) = [0,0]$ **then**
8 $\quad U_4(n) := I_R$
9 **else**
10 $\quad U_4(n) := U_3(n)$
11 $PRCh' := \langle Ch, U_4\rangle$
12 return $PRCh'$

then this algorithm uses I_R to record this recognition, though the details of when the observations were made are ignored (Line 7–8). Otherwise, this algorithm keeps this node (Line 10). If there is any $I_\emptyset$ for a unary constraint in a $PRCh$, then this $PRCh$ will be ignored from now on because $I_\emptyset$ means time inconsistency, and this $PRCh$ is no longer valid. Finally, this algorithm creates a new $PRCh'$ based on $PRCh$ and the updated unary constraint (Line 11).

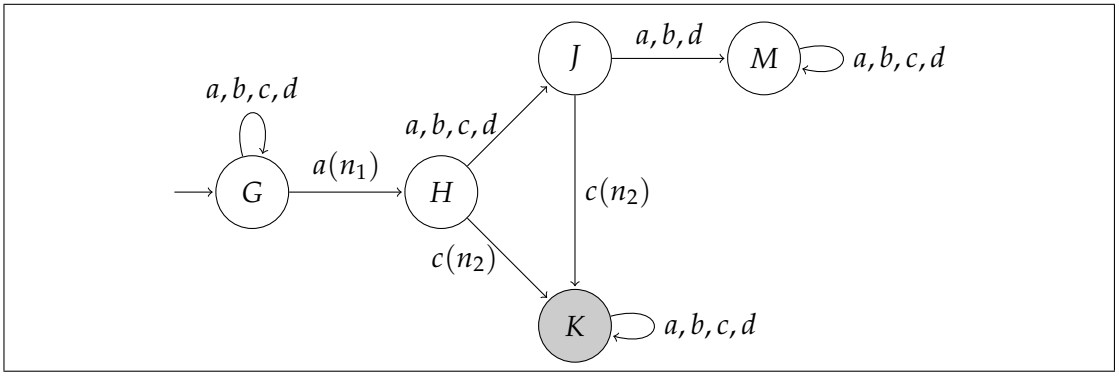

Figure 3.9: Chronicle automaton for Ch_1 in Fig. 3.7: only n_1 and n_2 are the nodes in the chronicle while the other events are not part of the chronicle recognition. In particular, state M contains time inconsistency, i.e. $I_\emptyset$.

A chronicle automaton is non-deterministic, i.e. when an event of a chronicle is observed, it is unknown whether it is part of a chronicle recognition or not. For example, Fig. 3.9 shows the chronicle automaton built for Ch_1. The set of events of the chronicle automaton is $\{a, b, c, d, n_1, n_2\}$ where n_1 (a) and n_2 (c) are the nodes in the chronicle while a, b, c, d are the events that correspond to the observations that are

not part of the chronicle recognition. Given an observation sequence 'a, b, b, b, a, c', observing the first 'a' does not mean observing the event of the node n_1. In fact, the event of n_1, i.e. a, is recognised at the fifth step, and the event of n_2, i.e. c, is recognised at the sixth.

Example 1: Chronicle Automaton for Ch_1 Fig. 3.9 shows the chronicle automaton built for Ch_1. Recall that the observable events are a, b, c, d.

- State G is the initial state such that $U(n_1) = [1, +\infty)$ and $U(n_2) = [1, +\infty)$, which means that the recognition has not started.
- In state H, $U(n_1) = I_R$ and $U(n_2) = [1, 2]$, which means that the node n_1 has been recognised and the node n_2 has a unary constraint such that if the event c corresponding to n_2 appears at the next one or two time steps, n_2 will be recognised.
- In state J, $U(n_1) = I_R$, and $U(n_2) = [1, 1]$, which means that n_1 has been recognised, and n_2 is expected in the next time step.
- In state M, $U(n_1) = I_R$ and $U(n_2) = I_\emptyset$, which means that c should have been observed, but c is not presented. This causes time inconsistency.
- State K is a $FRCh$ because $U(n_1) = I_R$ and $U(n_2) = I_R$. Therefore, K is the final and accepting state.

□

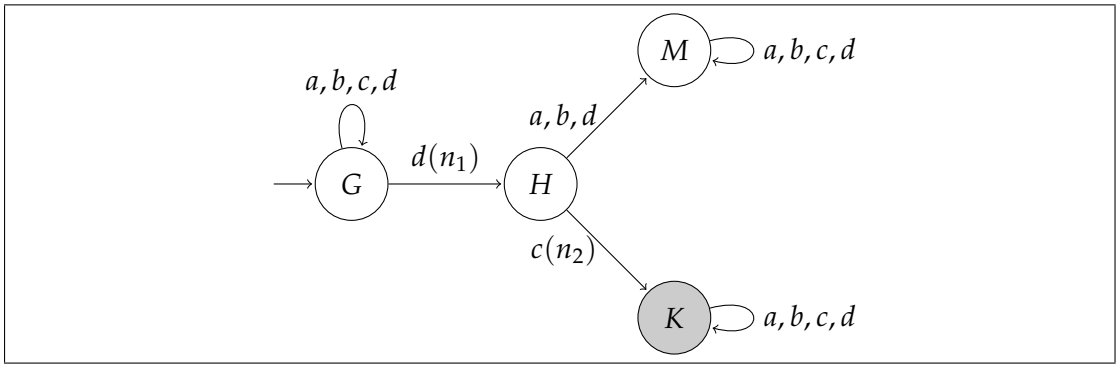

Figure 3.10: Chronicle automaton for Ch_2 in Fig. 3.8: only n_1 and n_2 are the nodes in the chronicle while the other events are not part of the chronicle recognition. In particular, state M contains time inconsistency, i.e. $I_\emptyset$.

Example 2: Chronicle Automaton for Ch_2 Fig. 3.10 shows the chronicle automaton of Ch_2. Recall that the observable events are a, b, c, d.

- State G is the initial state such that $U(n_1) = [1, +\infty)$ and $U(n_2) = [1, +\infty)$, which means that the recognition has not started.
- In state H, $U(n_1) = I_R$ and $U(n_2) = [1, 1]$, which means that the node n_1 has been recognised, and the node n_2 has a unary constraint such that if the event c corresponding to n_2 appears at the next time step, n_2 will be recognised.
- In state M, $U(n_1) = I_R$ and $U(n_2) = I_\emptyset$, which means that n_1 has been recognised; c should have been observed, but c is not presented. This causes time inconsistency.
- State K is a *FRCh* because $U(n_1) = I_R$ and $U(n_2) = I_R$. Therefore, K is the final and accepting state.

□

A chronicle automaton is not the simulation for a chronicle. Indeed, their semantics are different. In a chronicle automaton, the fault is diagnosed if there exists a path consistent with the observations that leads to a faulty state. In a simulation, all such paths should lead to a faulty state. The simulation for a chronicle is computed by determinisation [Epp, 2004] on a chronicle automaton.

Definition 21 (Construction of Deterministic Automaton by Subset Construction) *Given an non-deterministic automaton* $NFA = \langle Q_N, \Sigma_N, T_N, I_N, L_N \rangle$*, the deterministic automaton is constructed by* subset construction*, i.e.* $DFA = \langle Q_D, \Sigma_D, T_D, I_D, L_D \rangle$ *where*

- $Q_D = 2^{Q_N}$;
- $\Sigma_D = \Sigma_N$;
- $T_D \subseteq Q_D \times \Sigma_D \times Q_D$ *and*
  $T_D = \{\langle \{q_1, q_2, q_3, \ldots, q_x\}, e_z, \{q_y \mid i \in \{1, 2, 3, \ldots x\} \wedge \langle q_i, e_z, q_y \rangle \in T_N\}\rangle \mid e_z \in \Sigma_D \wedge x = |Q|\}$;
- $I_D = I_N$;
- $L_D(Q_N) = F$ *iff* $\exists q \in Q_N.\ L_N(q) = F$*; otherwise,* $L_D(Q_N) = N$.

Theorem 2 (Simulation for Chronicle-Recognition Diagnosis) *The determinisation of the chronicle automaton of a chronicle is the simulation for chronicle-recognition diagnosis where chronicle automaton defined in Definition 20 is the diagnosis model, and chronicle-recognition diagnosis defined in Definition 17 is the diagnostic algorithm.*

Proof of Theorem 2 Let $M = \langle Q_M, \Sigma_M, T_M, I_M, L_M \rangle$ be the system model; let Ch be a chronicle; let $CA = \langle Q_{CA}, \Sigma_{CA}, T_{CA}, I_{CA}, L_{CA} \rangle$ be the chronicle automaton; let OBS be an observation sequence. The proof consists of two parts.

The first part is to prove that if Ch is recognised in OBS, then the determinisation of CA for Ch says that Ch is fully recognised. Assuming that Ch is recognised in OBS, one can use a recognition function f_{CR} to index the observations. When following a single path on M labelled by this sequence, a state is reached by construction that is labelled faulty, i.e. there exists a single path in M, denoted as $path_1 : q_0 \xrightarrow{e_{11}} \ldots \xrightarrow{e_{x1}} q_x$ such that $q_0 \in I_M \wedge L_M(q_x) = F$, which corresponds to the path in CA, denoted as $path_2 : p_0 \xrightarrow{e_{12}} \ldots \xrightarrow{e_{x2}} p_x$ such that $p_0 \in I_{CA} \wedge L_{CA}(p_x) = F \wedge obs(e_{11} \ldots e_{x1}) = obs(e_{12} \ldots e_{x2})$. Hence, there exists a path corresponding to $path_2$ in the determinised CA that leads to the faulty state, which means that Ch is fully recognised in OBS.

The second part is to prove that if the determinisation of CA for Ch says that Ch is fully recognised in OBS, then there exists a recognition function f_{CR} to describe this chronicle recognition. Let $paths$ denote all paths in the chronicle simulation, i.e. $paths = \{q_0 \rightarrow \ldots \rightarrow q_x | q_o \in I_{CA} \wedge L_{CA}(q_x) = F\}$. For each path $(path_1 = q_0 \xrightarrow{e_{11}} \ldots \xrightarrow{e_{x1}} q_x) \in paths$, there is a corresponding path in M, denoted as $path_2 : p_0 \xrightarrow{e_{12}} \ldots \xrightarrow{e_{x2}} p_x$ such that $p_0 \in I_M \wedge L_M(p_x) = F \wedge obs(e_{11} \ldots e_{x1}) = obs(e_{12} \ldots e_{x2})$. In other words, when following a path $path_2$ on M labelled by this sequence, a state is reached by construction that is labelled faulty. Therefore, one can use a recognition function f_{CR} to index the observations. Both parts are proved. By Definition 11, the determinisation of the chronicle automaton of a chronicle is the simulation for chronicle-recognition diagnosis.

□

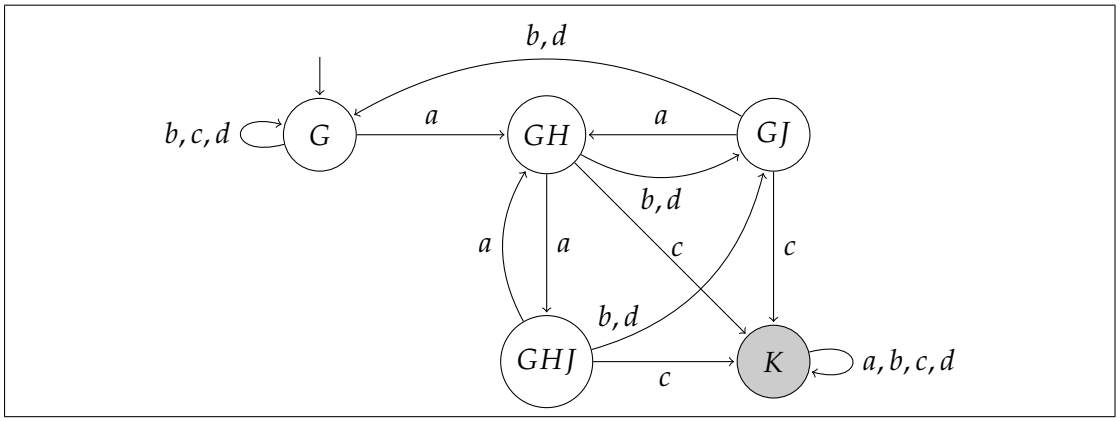

Figure 3.11: Chronicle simulation for Ch_1 in Fig. 3.7

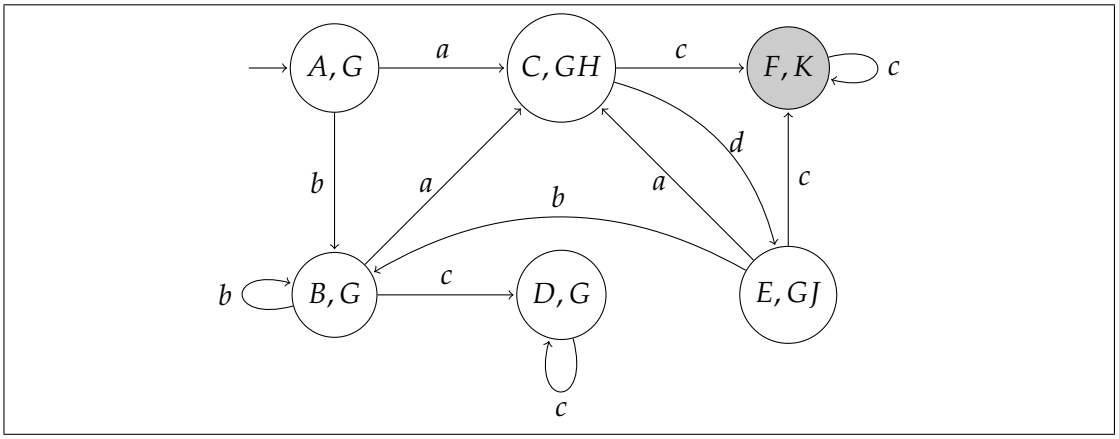

Figure 3.12: The synchronisation for the system model M_2 in Fig. 3.6 and the chronicle simulation for Ch_1 in Fig. 3.11

Example 1: Chronicle Simulation for Ch_1 Fig. 3.11 shows the chronicle simulation computed for Ch_1. K is a state to represent that once a chronicle is recognised in an observation sequence, it remains recognised. Fig. 3.12 shows the synchronisation for M_2 in Fig. 3.6 with the chronicle simulation for Ch_1 in Fig. 3.11.

Then, the twin plant method is applied to this synchronisation in order to verify the precision of chronicle Ch_1 w.r.t. M_2. There is no loop on any ambiguous path since there is no ambiguous path. Therefore, this chronicle-recognition diagnostic algorithm is precise w.r.t. M_2 by Theorem 1.

□

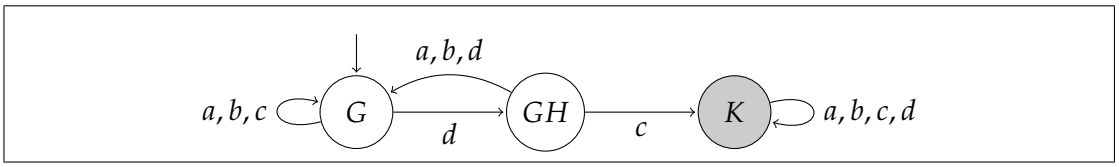

Figure 3.13: Chronicle simulation for Ch_2 in Fig. 3.8

Example 2: Chronicle Simulation for Ch_2 Fig. 3.13 shows the chronicle simulation computed for Ch_2. K is a state to represent that once a chronicle is recognised in an observation sequence, it remains recognised. Fig. 3.14 shows the synchronisation for M_2 in Fig. 3.6 with the chronicle simulation for Ch_2 in Fig. 3.13.

Then, the twin plant method is applied to this synchronisation in order to verify the precision of the chronicle Ch_2 w.r.t. M_2. There is a loop on the ambiguous path as highlighted. Therefore, this chronicle-recognition diagnostic algorithm is not precise w.r.t. M_2 by Theorem 1.

□

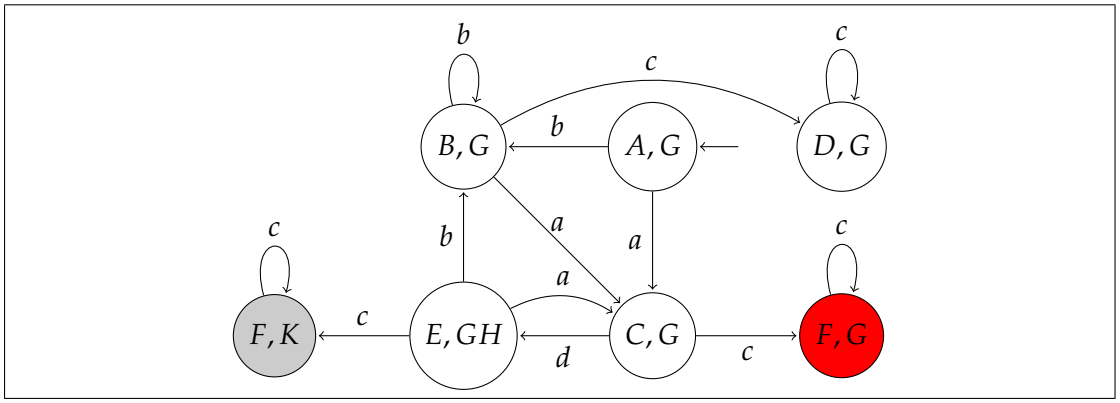

Figure 3.14: The synchronisation for the system model M_2 in Fig. 3.6 and the chronicle simulation for Ch_2 in Fig. 3.13

Finally, this chapter examines the complexity of generating chronicle automaton, and a simulation for chronicle-recognition diagnosis. Given a chronicle $Ch = \langle N, EL, B \rangle$, in the worst case, the number of states in a chronicle automaton is $O(|e|^r)$ where $|e|$ is the number of events; $[beginning, end]$ represents the widest binary constraint in a given chronicle, i.e.

$$max(\{(end - beginning) | [beginning, end] = B(n_1, n_2) \wedge n_1 \in N \wedge n_2 \in N\});$$

and $r = end - beginning + 2$. Nevertheless, after a transition is computed, any invalid $PRCh$ will be removed immediately due to the time inconsistency denoted as $I_\emptyset$. Therefore, this time consistency checking reduces the complexity from the worst case. Also, it should be noted that for a given chronicle, the computation for a chronicle automaton and a simulation is off-line and one-off.

3.4 Summary

This chapter studies the problem of computing the precision of a diagnostic algorithm w.r.t. a DES system model. Firstly, this chapter defines diagnosis and diagnosability of DES. Secondly, it provides the definitions for a diagnostic algorithm and its precision. Thirdly, this chapter proposes the simulation method to decide whether the precision is maintained if an imprecise diagnostic algorithm is used for DES diagnosis. This work defines a simulation for a diagnostic algorithm and a DES diagnosis model, which is a modification of the diagnosis model that simulates the process of a diagnostic algorithm generating a diagnostic result. Theorem 1 proves the correctness of the simulation method, and then uses the modified twin plant method on the synchronisation of a system and the simulation, in order to decide whether the

precision is maintained despite the fact of using an imprecise diagnostic algorithm.

This chapter illustrates the simulation method on chronicle-recognition diagnosis. Chronicle-recognition diagnosis by Dousson [1996] uses pattern recognition techniques. Given a DES diagnosis model and an observation sequence, if a chronicle is recognised in the observation sequence, then the chronicle-recognition diagnoser returns that the system is in the faulty mode. This algorithm however is potentially less precise than the exact model-based diagnosis, such as Δ. This chapter presents how to build a simulation for chronicle-recognition diagnosis. Theorem 2 proves the correctness of this approach.

In conclusion, the simulation method is beneficial for researchers to verify the precision of their diagnostic approaches. This theoretical framework will also allow researchers to consider more aggressive approaches to reduce the complexity of diagnosis, as they can now assess the precision of those approaches.

Chapter 4 will illustrate the simulation method on a new class of diagnostic algorithms, called Independent-Window Algorithms (IWAs). IWAs are potentially imprecise compared to the exact diagnosis Δ. Hence, Chapter 4 will examine their precision using the simulation method.

Chapter 4

Diagnosis of DES using Independent-Window Algorithms

Diagnosis of DES is performed by computing the paths on the complete model that generate the observations received on the system. Many previous works address this problem by the use of a diagnoser automaton [Sampath et al., 1995; Rozé and Cordier, 2002a], automata unfoldings [Baroni et al., 1999] that can also be distributed [Pencolé and Cordier, 2005; Su and Wonham, 2005], or encoded with BDD [Schumann et al., 2010], and finally by the use of a SAT solver [Grastien et al., 2007]. This work uses the term *belief state* to represent the set of global states which the system could be in after the given observations [Pencolé and Cordier, 2005; Rintanen, 2007]. On-line diagnosis iteratively computes the set of states that can be reached from the current belief state through the transitions that would produce exactly the next observation. If this is done explicitly [Baroni et al., 1999], the number of these states makes this approach inapplicable for many real-world problems. Comparatively, Schumann et al. [2010] proposed the symbolic approach where the states are represented in propositional logic, e.g. BDD. However, this representation is still subject to exponential blow-up. An alternative approach to on-line diagnosis is the off-line pre-computation of the potential belief states, as proposed by Sampath et al. [1995]. However, this is proven to be exponential w.r.t. the number of system states [Rintanen, 2007].

Any of the precited work, except the SAT approach, proposes diagnostic algorithms that attempt to compute at any time a belief state that is consistent with the observable flow from the time when the system starts operating to the current time. The main drawback of such a conservative strategy is the inability to follow the observable flow for a large system due to the exponential size of the generated belief states and therefore the temporal complexity to handle them. Although diagnosis using SAT computes one trace in the system for an observation sequence, the complexity of a SAT problem is exponential to the number of propositional variables, which is linear to the number of state variables [Grastien et al., 2007].

For the conservative strategy of exact diagnosis, the exponential blow-up arises from the fact that the diagnoser has to maintain all the information about the observations that it has received so far. For instance, the current belief state could be $\mathcal{B}'$ instead of $\mathcal{B}$ because of an earlier observation. Exact diagnosis has to compute and maintain every belief state. This chapter proposes a class of new diagnostic algorithms, called Independent-Window Algorithms (IWAs), for diagnosis of DES. IWAs slice the observations into time windows, and diagnose each of them independently. IWAs also share the property that they only consider the most recent observations. IWAs diagnose a certain number of observations for one time window, and move to another time window without keeping any information. This chapter presents four implementations of IWAs that differ only in the window selection, namely Al_p, Al_1, Al_2, and Al_3.

IWAs offer an array of benefits. First, IWAs improve the flexibility and feasibility of diagnosis by computing diagnosis independently on separate time windows. Second, IWAs avoid the overhead of maintaining a precise tracking of the system states, and thus reduce the computational complexity of diagnosis. Third, IWAs are able to handle intermittent loss of communication, whereby the state of the system becomes unknown. This is because at the beginning of one time window, IWAs reset to every nominal state, or every faulty state, depending on the result at the end of the previous time window. In other words, diagnosis using IWAs does not require a complete observation sequence that originates from the initial state of the given DES model.

On the other hand, IWAs may potentially lead to imprecise diagnosis. Since IWAs diagnose time windows independently, imprecision happens when both current and past observations are necessary to understand the system behaviour. In order to measure the quality of diagnosis, this work adapts the well-known criterion of diagnosability. Diagnosability is the property of an observable system, which states that using the model, a fault can be diagnosed after it occurs [Sampath et al., 1995]. This chapter applies the simulation method to verify the precision of an IWA w.r.t. a DES model, as proposed in Chapter 3. A simulation is a modified model that simulates how the diagnostic algorithm computes the diagnosis.

This chapter is organised as follows. Section 4.1 discusses the motivations for IWAs, and then presents the definitions of IWAs. Section 4.2 illustrates four implementations of IWAs. Section 4.3 demonstrates how to verify the precision of an IWA using the simulation method, and proves the correctness of this approach. Section 4.4 compares the precisions of the four IWA implementations. Section 4.5 discusses the application of alarm log handling. Section 4.6 summarises IWAs and concludes this chapter.

4.1 Independent-Window Algorithms

This section explains the intuitions and motivations for the IWAs. It also defines single-window diagnosis and window-based diagnosis.

4.1.1 Intuitions and Motivations

This section begins with explaining the intuition for IWAs. Given an observation sequence, IWAs perform independent diagnostic analyses on separate time windows. The simplest variant of IWAs is Al_1, which is visualised as follows.

$$\overbrace{o_1 \cdots\cdots o_k}^{\text{time window 1}}\overbrace{o_{k+1} \cdots\cdots o_{2k}}^{\text{time window 2}} \cdots\cdots \overbrace{o_{n-k+1} \cdots\cdots o_n}^{\text{time window } \frac{n}{k}}$$

Given a complete sequence of observations with n observable events, Al_1 divides all of the observations into $\frac{n}{k}$ time windows where k represents the number of observable events in each time window. This work assumes that n is a multiple of k, and Al_1 divides n observations exactly into $\frac{n}{k}$ time windows. IWAs differ from each other depending on their definitions of the time windows. Section 4.2 will study four implementations of IWAs, namely Al_p, Al_1, Al_2, and Al_3.

There are essentially three motivations for IWAs, namely, the improved flexibility, the reduction of the diagnostic complexity, and the use of IWAs to tackle the masking issue. First, IWAs perform independent diagnostic analyses on separate time windows. Because each diagnosis is performed separately, it is feasible to skip the ones that are believed irrelevant. Consider for instance a problem occurring on a web-service whose activity is logged. It is not feasible to diagnose from the initial states, i.e. from the beginning of the log, especially when the log dates back a long time ago. This is because using any existing exact diagnostic approach to diagnose an entire log requires maintaining all of the belief states, which is proved to be exponential w.r.t. the number of system states [Rintanen, 2007]. Alternatively, analysing the data some time before the incident is detected may be sufficient, which enhances the flexibility of diagnosis.

Second, the complexity of diagnosis is expected to be more manageable using independent windows. Diagnosis of DES requires maintaining the belief state of the system. Section 2.1 of Chapter 2 reviews the existing work on diagnosis. The belief state of a system can be computed off-line [Sampath et al., 1995], on-line [Baroni et al., 1999], in a distributed way [Pencolé and Cordier, 2005], or using symbolic tools [Schumann et al., 2010]. However, diagnosis for a large-scale system quickly becomes unmanageable. Let n be the number of states. The number of belief states

that can be reached is $O(2^n)$ [Rintanen, 2007].

With windows of fixed size, the complexity changes completely. For instance, given b observable events in the system model, and a size k of the window, the number of belief states that can be reached in the worst case is $\Sigma_{i\in\{0,1,2,\ldots,k\}} b^i = \frac{b^{k+1}-1}{b-1}$. The complexity is $O(b^k)$. Practically, this means that a diagnoser of a polynomial size can be built on k transitions of the system model.

Finally, when the communication layer used to transmit the observation is also subject to faults, certain observations are *masked*, which means that the complete sequence of observations is not available. IWAs are less subject to this issue. At the beginning of a time window, the diagnosis is reset to every nominal state, or every faulty state, depending on the result at the end of the previous time window.

4.1.2 Single-Window Diagnosis

Single-window diagnosis computes the diagnosis on windows. A *time window* is a small chunk of observations, which is formally defined as follows.

Definition 22 (Time Window) *Given a sequence of observations of a finite length $o = o_1, \ldots, o_n$, and two indexes $i < j$, the time window $o[i, j]$ is defined as the subsequence of observations $o_i, \ldots, o_j$ if $j < n$; otherwise, $o[i, j]$ is defined as $o_i, \ldots, o_n$.*

Given a time window $o_1, \ldots, o_k$, a system trace $\sigma \in \mathcal{L}$ is compatible with the window if the window appears in the observations of σ.

Definition 23 (Compatible System Trace with a Time Window) *Given a time window $o_1, \ldots, o_k$ where $k \geq 1$, a system trace $\sigma \in \mathcal{L}$ is compatible with the window if there exist two sequences of observations $o'_1, \ldots, o'_m$ and $o''_1, \ldots, o''_p$ where $m \geq 1$ and $p \geq 1$ such that $obs(\sigma) = o'_1, \ldots, o'_m, o_1, \ldots, o_k, o''_1, \ldots, o''_p$.*

Definition 24 (Single-Window Algorithm) *The* single-window algorithm*, denoted as $W_{i,j}$, is defined as follows:*

- *$W_{i,j}(M, o) = N$ if there exists a nominal trace $\sigma \in \mathcal{L}_N$ compatible with $o[i, j]$;*
- *$W_{i,j}(M, o) = F$ otherwise.*

In this work, a diagnosis problem is defined on a model and an observation. In general, the diagnoser should indicate whether the system is in the nominal mode, in the faulty mode, or it cannot decide, i.e. the system status is ambiguous. Assumption 4 of Chapter 3 states that the diagnosis policy of this work does not distinguish

between the nominal mode and ambiguous mode, i.e. the diagnoser assumes that the system is not faulty unless proved otherwise.

Definition 24 says that as long as there is a nominal explanation for a time window $o[i,j]$, the diagnostic result for this time window is nominal. If there is no nominal explanation for a time window $o[i,j]$, then the diagnostic result for this time window must be faulty.

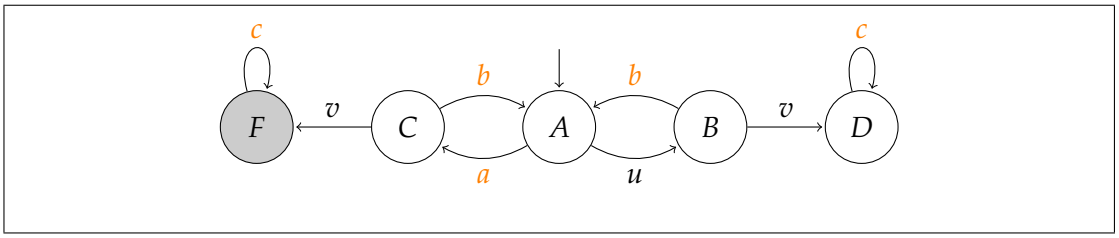

Figure 4.1: A simple DES model M_3 where a, b, c are observable events, and u, v are unobservable events

Fig. 4.1 shows a simple DES model M_3. State F is faulty while the other states are nominal. Observable events are a, b, c, and unobservable events are u, v. This work provides three examples. The first two examples have a nominal output while the third example has a faulty output.

Example 1: Single-Window Algorithm Given the system model M_3, an observation sequence $o = a,b,a,b,b,c,c,c$, and a time window $o[1,4] = a,b,a,b$, the Single-Window Algorithm $W_{1,4}(M_3, o)$ evaluates to N since there exists a sequence $\sigma = a,b,a,b$ and a corresponding trace $t : A \xrightarrow{a} C \xrightarrow{b} A \xrightarrow{a} C \xrightarrow{b} A$ such that $\sigma \in \mathcal{L}_N$. Therefore, there exists a nominal trace σ that is compatible with $o[1,4]$.

□

Example 2: Single-Window Algorithm Given the system model M_3, an observation sequence $o = a,b,a,b,b,c,c,c$, and a time window $o[5,6] = b,c$, the Single-Window Algorithm $W_{5,6}(M_3, o)$ evaluates to N since there exists a sequence $\sigma = a,b,a,b,b,c$ and a corresponding trace $t : A \xrightarrow{a} C \xrightarrow{b} A \xrightarrow{a} C \xrightarrow{b} A \xrightarrow{u} B \xrightarrow{b} A \xrightarrow{u} B \xrightarrow{v} D \xrightarrow{c} D$ such that $\sigma \in \mathcal{L}_N$. Therefore, there exists a nominal trace σ that is compatible with $o[5,6]$.

□

Example 3: Single-Window Algorithm Given the system model M_3, an observation sequence $o = a,b,a,c,c,c,c,c$, and a time window $o[1,5] = a,b,a,c,c$, the Single-Window Algorithm $W_{1,5}(M_3, o)$ evaluates to F since there only exists one sequence

$\sigma = a, b, a, c, c$ and the corresponding trace $t : A \xrightarrow{a} C \xrightarrow{b} A \xrightarrow{a} C \xrightarrow{v} F \xrightarrow{c} F \xrightarrow{c} F$ such that $\sigma \in \mathcal{L}_F$. Therefore, there is no nominal trace σ that is compatible with $o[1,5]$.

□

Lemma 1 $W_{i,j}(M, o) = F \Rightarrow \Delta(M, o) = F$.

Proof of Lemma 1 Suppose $W_{i,j}(M, o) = F$. Then, there is no nominal trace $\sigma \in \mathcal{L}_N$ that is compatible with $o[i, j]$. Therefore, there is no σ such that $\sigma \in \mathcal{L}_N \wedge obs(\sigma) = o$. By Definition 7, $\Delta(M, o) = F$.

□

Lemma 1 is similar to the correctness property of a diagnostic algorithm as in Definition 9. It provides an important inference that as soon as a time window is diagnosed as faulty, the system must be faulty. Lemma 1 will be used in Section 4.1.3 to prove the monotonicity and correctness of window-based diagnostic algorithms.

4.1.3 Windows-based Diagnosis

Windows-based diagnosis offers various ways to slice an observation sequence into time windows. The final output will be the conjunction of the diagnostic results of several time windows, i.e. the system is in the faulty mode if the diagnosis on at least one time window returns faulty; it is in the nominal mode if all time windows return nominal.

Definition 25 (Window-based Diagnostic Algorithm) *Given a diagnostic algorithm A, a system model M, observations $o \in {\Sigma_o}^*$, and time windows $T = \{[i_1, j_1], [i_2, j_2], [i_3, j_3], \dots\}$, the diagnosis of $A(M, o, T)$ is defined as follows.*

$$A(M, o, T) = \begin{cases} N & \text{if } \forall [i, j] \in T,\ W_{i,j}(M, o) = N \\ F & \text{otherwise.} \end{cases}$$

Notice that more observations imply that the number of time windows increases, which makes it more likely for a single-window diagnosis to diagnose the fault. This work provides two examples. The first example has a nominal output while the second example has a faulty output.

Example 1: Window-based Diagnostic Algorithm Given the system model M_3, an observation sequence $o = a, b, a, b, b, c, c, c$, and $T = \{[1, 4], [5, 6]\}$, the Window-based Diagnostic Algorithm $A(M_3, o)$ evaluates to N since both $W_{1,4}(M_3, o)$ and $W_{5,6}(M_3, o)$ evaluate to N.

□

Example 2: Window-based Diagnostic Algorithm Given the system model M_3, an observation sequence $o = a, b, a, c, c, c, c, c$, and $T = \{[1,5],[6,7]\}$, the Window-based Diagnostic Algorithm $A(M_3, o)$ evaluates to F since $W_{1,5}(M_3, o)$ evaluates to F.

□

Recall that Definition 9 specifies the two properties of monotonicity and correctness for a diagnostic algorithm. The following two theorems examine the monotonicity and correctness of window-based diagnostic algorithms.

Theorem 3 (Monotonicity of Window-based Diagnostic Algorithm) *A window-based diagnostic algorithm preserves the monotonicity condition.*

Proof of Theorem 3

$$\text{Suppose } A(M, o, T) = F. \tag{4.1}$$
$$\Rightarrow \text{The diagnosis on at least one time window returns faulty, i.e.} \tag{4.2}$$
$$\exists [i,j] \in T,\ W_{i,j}(M, o) = F \tag{4.3}$$
$$\forall e \in \Sigma_o, o[i,j] \text{ is a subsequence of } oe[i,j]. \tag{4.4}$$
$$\Rightarrow \exists [i,j] \in T,\ W_{i,j}(M, oe) = F \tag{4.5}$$
$$\Rightarrow A(M, oe, T) = F. \tag{4.6}$$

Therefore, a window-based diagnostic algorithm preserves the monotonicity property, i.e. $A(M, o, T) = F \Rightarrow \forall e \in \Sigma_o,\ A(M, (oe), T) = F$. □

Theorem 4 (Correctness of Window-based Diagnostic Algorithm) *A window-based diagnostic algorithm preserves the correctness condition.*

Proof of Theorem 4 Lemma 1 states that $W_{i,j}(M, o) = F \Rightarrow \Delta(M, o) = F$. If $A(M, o, T) = F$, then $\exists [i,j] \in$ Time-Windows, $W_{i,j}(M, o) = F$. By Lemma 1, $\Delta(M, o) = F$. Therefore, a window-based diagnostic algorithm preserves the correctness condition, i.e. $A(M, o, T) = F \Rightarrow \Delta(M, o) = F$.

□

4.2 Four Implementations of IWAs

This section illustrates four implementations of IWAs, namely Al_p, Al_1, Al_2, and Al_3, which differ only in the window selection T. This section also explains their expected benefits. These algorithms follow Definition 24 for a single-window algorithm, and Definition 25 for a window-based diagnostic algorithm.

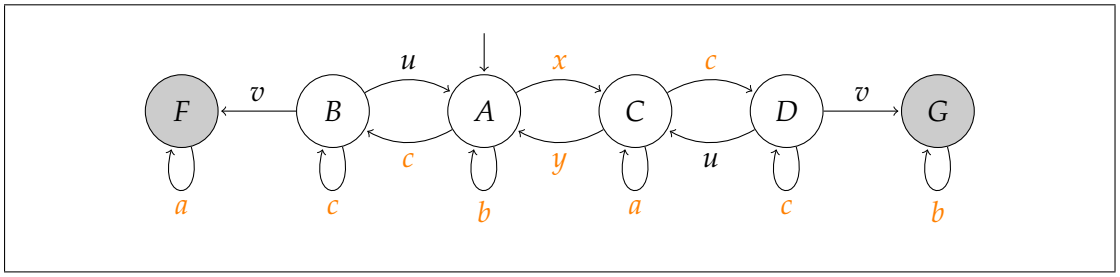

Figure 4.2: A simple DES model M_4 where a, b, c, x, y are observable events, and u, v are unobservable events

Observations	Model	Diagnoser	Diagnostic Result
a, b, b, b, b, c, c, c	M_3	Δ	N
a, b, b, b, a, c, c, c	M_3	Δ	F
b, a, b, a, c, c, c, c	M_3	Δ	F
x, c, c, b, b, b, b, b	M_4	Δ	F
x, c, c, c, b, b, b, b	M_4	Δ	F
b, x, c, c, c, b	M_4	Δ	F

Table 4.1: Diagnostic results for M_3 in Fig. 4.1 and M_4 in Fig. 4.2

Fig. 4.2 shows a simple DES model M_4 where a, b, c, x, y are observable events, and u, v are unobservable events. Tab. 4.1 shows five examples of diagnostic results for the system model M_3 in Fig. 4.1 and M_4 in Fig. 4.2. This section compares the diagnostic results of the four IWAs with exact diagnosis Δ.

We define the generalised IWA. *IWA* begins at the i_1-th observation, and the size of each time window is k. There is a finite delay d between two time windows, which is measured by the difference between the starting index of two consecutive time windows. *IWA* aims at slicing observations and running diagnosis using samples of time windows. *IWA* also enforces the time windows to be frequently reoccurring. Although this approach may not diagnose a fault as early as using exact diagnosis, *IWA* aims to diagnose a fault in one of the samples of time windows. *IWA* may skip some time windows, which reduces the computational complexity of diagnosis. The time windows of *IWA* are defined as follows.

$$T_{IWA}(i_1, k, d) = \{[i_1, i_1 + k - 1], [i_2, i_2 + k - 1], [i_3, i_3 + k - 1], \dots\} \text{ s.t. } d = i_x - i_{x-1},$$

$$\text{where } x \geq 2 \wedge x \in \mathbf{Z} \wedge d > 0$$

For example, when $i_1 = 1$ and $d > k$, the first two time windows of *IWA* are visualised as follows.

$$\overbrace{o_1 \dots\dots o_k}^{\text{time window 1}} \dots\dots \overbrace{o_{d+1} \dots\dots o_{d+k}}^{\text{time window 2}} \dots\dots$$

4.2.1 Preliminary Independent-Window Algorithm Al_p

Al_p is a preliminary and conservative IWA. Al_p proposes to use sliding time windows for diagnosis. A sliding time window moves along an observation flow by one observation at a time. Al_p begins at the i-th observation, and the size of each time window is k. Let the length of a time window be k. The finite delay d between two time windows is 1. The time windows of Al_p are defined as follows.

$$T_{Al_p}(i,k,1) = \{[i, i+k-1], [i+1, i+k], [i+2, i+k+1] \dots\}$$

For example, when $i = 1$, the first two time windows are visualised as follows.

$$\overbrace{o_1 \underbrace{o_2 \dots\dots o_k}_{\text{time window 2}} o_{k+1}}^{\text{time window 1}} \dots\dots$$

Input	Time Window	Diagnosis	Output
x,c,c,b,b,b,b,b	(x,c,c,b)	F	F (precise)
	(c,c,b,b)	Al_p may stop	
	(c,b,b,b)	Al_p may stop	
	(b,b,b,b)	Al_p may stop	
	(b,b,b,b)	Al_p may stop	
x,c,c,c,b,b,b,b	(x,c,c,c)	N	N (imprecise)
	(c,c,c,b)	N	
	(c,c,b,b)	N	
	(c,b,b,b)	N	
	(b,b,b,b)	N	

Table 4.2: Examples of Al_p running on M_4 in Fig. 4.2 where $i = 1$ and $k = 4$

Example of Al_p Tab. 4.2 shows a set of results of running Al_p on M_4 where i is 1 and k is 4. In this case,

$$T_{Al_p}(1,4,1) = \{[1,4], [2,5], [3,6], [4,7], [5,8]\}.$$

The first output is precise while the second output is not precise. Given the first observation sequence x,c,c,b,b,b,b,b, the time windows are (x,c,c,b), (c,c,b,b), (c,b,b,b), (b,b,b,b), and (b,b,b,b). For the first time window (x,c,c,b), there is only one trace that is compatible with this time window, i.e. $A \xrightarrow{x} C \xrightarrow{c} D \xrightarrow{c} D \xrightarrow{v} G \xrightarrow{b} G$. In particular, $L(G)$ evaluates to F. This trace is faulty, and therefore there is no nominal explanation. Thus, the diagnostic result for the first time window is F, i.e.

$W_{1,4}(M, o)$ evaluates to F. Therefore, the first observation sequence is diagnosed as F.

Given the second observation sequence x, c, c, c, b, b, b, b, the time windows are: (x, c, c, c), (c, c, c, b), (c, c, b, b), (c, b, b, b), and (b, b, b, b).

- For the first time window (x, c, c, c), there exists a nominal trace that is compatible with this time window, i.e. $A \xrightarrow{x} C \xrightarrow{c} D \xrightarrow{c} D \xrightarrow{c} D$. In particular, $L(D)$ evaluates to N. Thus, the diagnostic result of the first time window is N, i.e. $W_{1,4}$ evaluates to N.

- For the second time window (c, c, c, b), there exists a nominal trace that is compatible with this time window, i.e. $A \xrightarrow{c} B \xrightarrow{c} B \xrightarrow{c} B \xrightarrow{u} A \xrightarrow{b} A$. In particular, $L(A)$ evaluates to N. Thus, the diagnostic result of the second time window is N, i.e. $W_{2,5}$ evaluates to N.

- For the third time window (c, c, b, b), there exists a nominal trace that is compatible with this time window, i.e. $A \xrightarrow{c} B \xrightarrow{c} B \xrightarrow{u} A \xrightarrow{b} A \xrightarrow{b} A$. In particular, $L(A)$ evaluates to N. Thus, the diagnostic result of the third time window is N, i.e. $W_{3,6}$ evaluates to N.

- For the fourth time window (c, b, b, b), there exists a nominal trace that is compatible with this time window, i.e. $A \xrightarrow{c} B \xrightarrow{u} A \xrightarrow{b} A \xrightarrow{b} A \xrightarrow{b} A$. In particular, $L(A)$ evaluates to N. Thus, the diagnostic result of the fourth time window is N, i.e. $W_{4,7}$ evaluates to N.

- For the fifth time window (b, b, b, b), there exists a nominal trace that is compatible with this time window, i.e. $A \xrightarrow{c} B \xrightarrow{u} A \xrightarrow{b} A \xrightarrow{b} A \xrightarrow{b} A \xrightarrow{b} A$. In particular, $L(A)$ evaluates to N. Thus, the diagnostic result of the fifth time window is N, i.e. $W_{5,8}$ evaluates to N.

Therefore, the second observation sequence is diagnosed as N, i.e. $\forall [i, j] \in T_{Al_p}$ such that $W_{i,j}(M, o)$ evaluates to N. However, this observation should lead to F using exact diagnosis Δ. Therefore, the diagnostic result of the second time window is imprecise. □

Al_p is a very conservative IWA. Its time windows moves along an observation sequence by one observation at a time. This preliminary IWA attempts to diagnose a fault as early as using exact diagnosis. However, Al_p has two major disadvantages.

- Al_p is inefficient to run in terms of time. Suppose the number of observations is $|obs|$, and the time window size is k. Since Al_p moves by one observation at a time, the number of time windows to diagnose in the worst case is $(|obs| -$

$k+1$). This leads to a large amount of time windows to diagnose, and the computational time will be long.

- Although Al_p has very conservative selections of time windows, the precision of using Al_p varies between DES models. The above example of Al_p demonstrates that it is imprecise w.r.t. M_4. In general, Al_p cannot handle the situations where some particular events must be included to one time window in order to produce a precise diagnostic result. The precision of Al_p w.r.t. a DES model will be examined in Section 4.3.

In summary, Al_p is a preliminary and very conservative IWA. Al_p inspires the other IWAs, namely Al_1, Al_2, and Al_3, which will be explained in the following sub-sections.

4.2.2 Independent-Window Algorithm Al_1

This work proposes Al_1, which is the simplest variant of IWAs. Al_1 begins at the i-th observation, and slices a sequence of observations every k observations. The finite delay d between two time windows is k. It is useful for those systems that is sufficient to diagnose a fault using a bounded time window. The time windows of Al_1 are defined as follows.

$$T_{Al_1}(i,k,k) = \{[i, i+k-1], [i+k, i+2k-1], [i+2k, i+3k-1], \dots\}$$

The first three time windows of Al_1 are visualised as follows.

$$\overbrace{o_i \dots\dots o_{i+k-1}}^{\text{time window 1}}\overbrace{o_{i+k} \dots\dots o_{i+2k-1}}^{\text{time window 2}}\overbrace{o_{i+2k} \dots\dots o_{i+3k-1}}^{\text{time window 3}} \dots\dots$$

Observation	Time Window	Diagnosis	Output
a,b,b,b,b,c,c,c	(a,b,b,b)	N	N (precise)
	(b,c,c,c)	N	
a,b,b,b,a,c,c,c	(a,b,b,b)	N	F (precise)
	(a,c,c,c)	F	
b,a,b,a,c,c,c,c	(b,a,b,a)	N	N (imprecise)
	(c,c,c,c)	N	

Table 4.3: Examples of Al_1 running on M_3 in Fig. 4.1 where $i=1$ and $k=4$

Example of Al_1 Tab. 4.3 shows a set of results of running Al_1 on M_3 in Fig. 4.1 with the starting observation index $i=1$ and the time window size $k=4$. In this case,

$T_{Al_1}(1,4,4) = \{[1,4],[5,8]\}$. Given the first observation sequence $o = a,b,b,b,b,c,c,c$, the time windows are: (a,b,b,b), and (b,c,c,c).

- For the first time window (a,b,b,b), there exists a nominal trace that is compatible with this time window, i.e. $A \xrightarrow{a} C \xrightarrow{b} A \xrightarrow{u} B \xrightarrow{b} A \xrightarrow{u} B \xrightarrow{b} A$. In particular, $L(A)$ evaluates to N. Therefore, the first time window leads to N, i.e. $W_{1,4}(M,o)$ evaluates to N.

- For the second time window (b,c,c,c), there exists a nominal trace that is compatible with this time window, i.e. $A \xrightarrow{u} B \xrightarrow{b} A \xrightarrow{u} B \xrightarrow{v} D \xrightarrow{c} D \xrightarrow{c} D \xrightarrow{c} D$. In particular, $L(D)$ evaluates to N. Thus, the diagnostic result of the second time window is N, i.e. $W_{5,8}(M,o)$ evaluates to N.

Therefore, the first observation sequence is diagnosed as N, i.e. $\forall[i,j] \in T_{Al_1}$ such that $W_{i,j}(M,o)$ evaluates to N.

Given the second observation sequence a,b,b,b,a,c,c,c, the time windows are: (a,b,b,b), and (a,c,c,c).

- For the first time window (a,b,b,b), there exists a nominal trace that is compatible with this time window, i.e. $A \xrightarrow{a} C \xrightarrow{b} A \xrightarrow{u} B \xrightarrow{b} A \xrightarrow{u} B \xrightarrow{b} A$. In particular, $L(A)$ evaluates to N. Therefore, the first time window leads to N, i.e. $W_{1,4}(M,o)$ evaluates to N.

- For the second time window (a,c,c,c), there is only one trace that is compatible with this time window, i.e. $A \xrightarrow{a} C \xrightarrow{v} F \xrightarrow{c} F \xrightarrow{c} F \xrightarrow{c} F$. In particular, $L(F)$ evaluates to F. This trace is faulty, and therefore there is no nominal explanation. Thus, the diagnostic result of the second time window is F, i.e. $W_{5,8}(M,o)$ evaluates to F.

Therefore, the second observation sequence is diagnosed as F.

Given the third observation sequence b,a,b,a,c,c,c,c, the time windows are: (b,a,b,a) and (c,c,c,c).

- For the first time window (b,a,b,a), there exists a nominal trace that is compatible with this time window, i.e. $A \xrightarrow{u} B \xrightarrow{b} A \xrightarrow{a} C \xrightarrow{b} A \xrightarrow{a} C$. In particular, $L(C)$ evaluates to N. Therefore, the first time window leads to N, i.e. $W_{1,4}(M,o)$ evaluates to N.

- For the second time window (c,c,c,c), there exists a nominal trace that is compatible with this time window, i.e. $A \xrightarrow{u} B \xrightarrow{v} D \xrightarrow{c} D \xrightarrow{c} D \xrightarrow{c} D \xrightarrow{c} D$. In particular, $L(D)$ evaluates to N. Therefore, the diagnostic result of the second time window is N, i.e. $W_{5,8}(M,o)$ evaluates to N.

Therefore, the third observation sequence is diagnosed as N, i.e. $\forall[i,j] \in T_{Al_1}(1,4,4)$ such that $W_{i,j}(M,o)$ evaluates to N. However, Tab. 4.1 shows this observation should lead to F. This example demonstrates that Al_1 has its drawbacks of imprecise diagnosis, i.e. it cannot handle the situations where some particular events must be included to one time window in order to produce a precise diagnostic result. The precision of Al_1 w.r.t. a DES model will be examined in Section 4.3.1.

□

4.2.3 Independent-Window Algorithm Al_2

Al_2 aims to make improvement to Al_1. Al_2 is based on Al_1, and begins at the i-th observation. The size of each time window is k for simplicity. This work assumes that k is a multiple of 2. The finite delay d between two time windows is $\frac{k}{2}$. As seen in Section 4.2.2, Al_1 is unable to deal with the situation which requires some consecutive observations to diagnose a fault. In addition to the time windows of Al_1, Al_2 includes all slicing points of the time windows. Al_2 makes sure the windows overlap so that consecutive observations always appear in at least one time window. The time windows of Al_2 are defined as follows.

$$T_{Al_2}(i,k,\frac{k}{2}) = \{[i,i+k-1],[i+k,i+2k-1],[i+2k,i+3k-1],\dots\}\cup$$

$$\{[i+\frac{k}{2},i+\frac{3k}{2}-1],[i+\frac{3k}{2},i-1+\frac{5k}{2}],[i+\frac{5k}{2},i-1+\frac{7k}{2}],\dots\}$$

For example, when $i = 1$, the first three time windows of Al_2 are visualised as follows.

$$\overbrace{o_1 \dots o_{\frac{k}{2}} \underbrace{o_{\frac{k}{2}+1} \dots o_k}_{\text{time window 2}}}^{\text{time window 1}} \overbrace{o_{k+1} \dots o_{\frac{3}{2}k}}^{\text{time window 3}} o_{\frac{3}{2}k+1} \dots o_{2k} \dots$$

Example 1 of Al_2 Tab. 4.4 shows a set of results of running Al_2 on M_3 with $i = 1$ and $k = 4$. The input observation sequences are the same as those in Tab. 4.3. In this case, $T_{Al_2}(1,4,2) = \{[1,4],[3,6],[5,8]\}$. All diagnostic results in this table are precise. Al_1 cannot precisely diagnose the third observation. In contrast, Al_2 returns F, which is a precise result. This is because it is necessary to include a, c in one time window to diagnose the fault in this DES model. Al_1 cannot guarantee that a and c always appear in one time window while Al_2 is able to provide such guarantee.

□

Input	Time Window	Diagnosis	Output
a,b,b,b,b,c,c,c	(a,b,b,b)	N	N (precise)
	(b,b,b,c)	N	
	(b,c,c,c)	N	
a,b,b,b,a,c,c,c	(a,b,b,b)	N	F (precise)
	(b,b,a,c)	F	
	(a,c,c,c)	F	
b,a,b,a,c,c,c,c	(b,a,b,a)	N	F (precise)
	(b,a,c,c)	F	
	(c,c,c,c)	N	

Table 4.4: Examples of Al_2 running on M_3 in Fig. 4.1 where $i = 1$ and $k = 4$

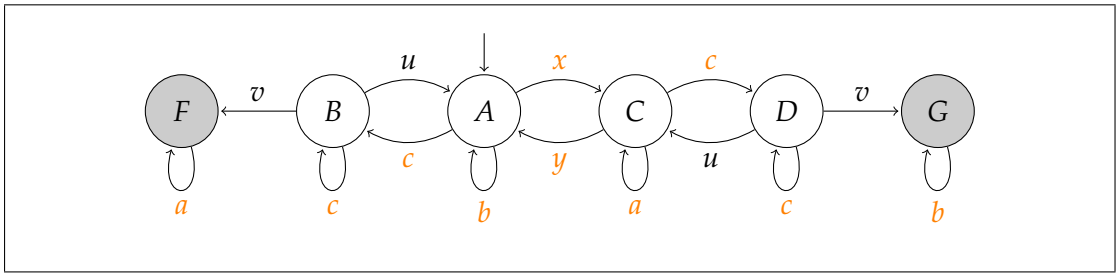

Figure 4.3: A simple DES model M_4 where a,b,c,x,y are observable events, and u,v are unobservable events

Input	Time Window	Diagnosis	Output
b,x,c,c,c,b	(b,x)	N	N (imprecise)
	(x,c)	N	
	(c,c)	N	
	(c,c)	N	
	(c,b)	N	

Table 4.5: Examples of Al_2 running on M_4 in Fig. 4.3 where $i = 1$ and $k = 2$

Example 2 of Al_2 Fig. 4.3 shows a DES model M_4 where a,b,c,x,y are observable events, and u,v are unobservable events. Tab. 4.5 shows the result of running Al_2 on M_4 with $i = 1$ and $k = 2$. In this case, $T_{Al_2}(1,2,1) = \{[1,2],[2,3],[3,4],[4,5],[5,6]\}$. The diagnostic output is not precise. Given the observation sequence b,x,c,c,c,b, the time windows are (b,x), (x,c), (c,c), (c,c), and (c,b).

- For the first time window (b,x), there exists a nominal trace that is compatible with this time window, i.e. $A \xrightarrow{b} A \xrightarrow{x} C$. In particular, $L(C)$ evaluates to N. Thus, the first time window leads to N, i.e. $W_{1,2}(M,o)$ evaluates to N.

- For the second time window (x,c), there exists a nominal trace that is compat-

ible with this time window, i.e. $A \xrightarrow{x} C \xrightarrow{c} D$. In particular, $L(D)$ evaluates to N. Thus, the second time window leads to N, i.e. $W_{2,3}(M, o)$ evaluates to N.

- For the third and the fourth time window (c, c), there exists a nominal trace that is compatible with this time window, i.e. $A \xrightarrow{c} B \xrightarrow{c} B$. In particular, $L(B)$ evaluates to N. Thus, both of the third and the fourth time window lead to N, i.e. both $W_{3,4}(M, o)$ and $W_{4,5}(M, o)$ evaluates to N.
- For the fifth time window (c, b), there exists a nominal trace that is compatible with this time window, i.e. $A \xrightarrow{c} B \xrightarrow{u} A \xrightarrow{b} A$. In particular, $L(A)$ evaluates to N. Thus, the fifth time window leads to N.

Therefore, this observation sequence is diagnosed as N, which is a precise diagnostic result, i.e. $\forall[i, j] \in T_{Al_2}(1, 2, 1)$ such that $W_{i,j}(M, o)$ evaluates to N. However, this observation should lead to F using exact diagnosis Δ, as seen in Tab. 4.1. This example demonstrates that Al_2 as an IWA may lead to imprecise diagnostic results. The precision of Al_2 w.r.t. a DES model will be examined in Section 4.3.2.

□

4.2.4 Independent-Window Algorithm Al_3

We investigate the variable value of the d parameter in IWA. This case of IWA is named as Al_3.

Input	Time Window	Diagnosis	Output
a, b, b, b, b, c, c, c	(a, b)	N	N (precise)
	(b, b)	N	

Table 4.6: Examples of Al_3 running on M_3 in Fig. 4.1 where $i_1 = 1, k = 2$, and $d = 3$

Example of Al_3 Tab. 4.6 shows the result of running Al_3 on M_3 with $i_1 = 1, k = 2, d = 3$. In this case, $T_{Al_3}(1, 2, 3) = \{[1, 2], [4, 5]\}$. The diagnostic output is precise. Given the observation sequence a, b, b, b, b, c, c, c, the time windows are $(a, b), (b, b)$.

- For the first time window (a, b), there exists a nominal trace that is compatible with this time window, i.e. $A \xrightarrow{a} C \xrightarrow{b} A$. In particular, $L(A)$ evaluates to N. Thus, the first time window leads to N, i.e. $W_{1,2}(M, o)$ evaluates to N.
- For the second time window (b, b), there exists a nominal trace that is compatible with this time window, i.e. $A \xrightarrow{u} B \xrightarrow{b} A \xrightarrow{u} B \xrightarrow{b} A$. In particular, $L(A)$ evaluates to N. Thus, the second time window leads to N, i.e. $W_{4,5}(M, o)$ evaluates to N.

Therefore, this observation sequence is diagnosed as N, i.e. $\forall[i,j] \in T_{Al_3}(1,2,3)$ such that $W_{i,j}(M,o)$ evaluates to N. The precision of Al_3 w.r.t. a DES model will be examined in Section 4.3.4.

□

4.3 Precision Verification for IWAs

Section 4.2 presents four implementations of IWAs. As seen in those examples, IWAs may lead to imprecise diagnostic results. This section examines the precision of each of the four IWAs w.r.t. a DES model by constructing a simulation as proposed in Chapter 3. Recall that the simulation is a modified model that simulates how a diagnostic algorithm works on a DES model.

Although a simulation has more states than the original system model, a simulation is only used for precision verification, and not used for diagnosis. Furthermore, the states and transitions of the simulation automata do not need to be explicitly represented, but can be represented symbolically, e.g. using BDD [Schumann et al., 2010].

4.3.1 Precision of Al_1 w.r.t. a DES Model

Given k, the Al_1-simulation for a DES model is formally defined as follows.

Definition 26 (Al_1-Simulation) *Given k, the Al_1-simulation for model $M = \langle Q, \Sigma, T, I, L\rangle$ is the automaton $M_1' = \langle Q_1', \Sigma_1', T_1', I_1', L_1'\rangle$ where*

- $Q_1' = Q \times \{0,1,2,\ldots,k\}$;
- $\Sigma_1' = \Sigma \cup \{\varepsilon\}$;
- $T_1' = T_u \cup T_o \cup T_\varepsilon$ *is defined by:*
  - $T_u = \{\langle\langle q,i\rangle, u, \langle q',i\rangle\rangle \mid i \in \{0,1,2,\ldots,k\} \wedge \langle q,u,q'\rangle \in T \wedge u \in \Sigma \setminus \Sigma_o\}$;
  - $T_o = \{\langle\langle q,i\rangle, o, \langle q',i+1\rangle\rangle \mid i \in \{0,1,2,\ldots,k-1\} \wedge \langle q,o,q'\rangle \in T \wedge o \in \Sigma_o\}$;
  - $T_\varepsilon = \{\langle\langle q,k\rangle, \varepsilon, \langle q',0\rangle\rangle \mid L(q) = L(q')\}$;
- $I_1' = I \times \{0\}$;
- $L_1'(\langle q,i\rangle) = L(q)$ *for all $q \in Q$ and $i \in \{0,1,2,\ldots,k\}$.*

The simulation for Al_1 and a DES model M, denoted as $si(M, Al_1)$, is an automaton that contains $(k+1)$ copies of the states of M. Each state $\langle q,i\rangle$ of $si(M, Al_1)$

records not only the system state q but also the number i of observations in the current window. When the number of observations in one time window reaches k, the state of the simulation is "reset", i.e. an empty transition, denoted as ε, takes the simulation from state $\langle q, k\rangle$ to state $\langle q', 0\rangle$ such that q' and q only have in common that their diagnostic information is the same, i.e. $L(q) = L(q')$.

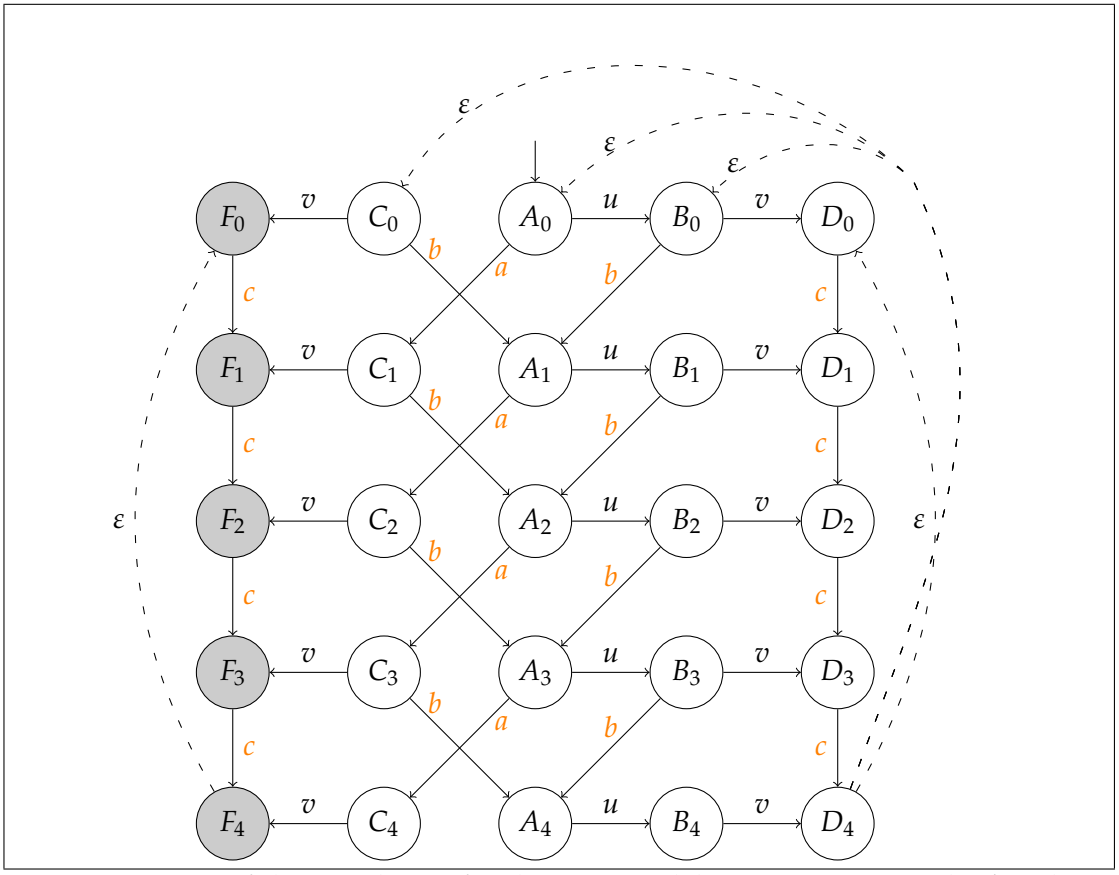

Figure 4.4: Part of the simulation for the DES model M_3 in Fig. 4.1 and Al_1, where $k = 4$. Dotted lines also need to link A_4 to A_0, B_0, C_0 and D_0. Same applies to B_4 and C_4.

Example 1: Simulation for M_3 and Al_1 Fig. 4.4 illustrates part of the simulation for M_3 in Fig. 4.1 and Al_1 where k is 4. F_0, F_1, F_2, F_3 and F_4 are faulty states while the rest states are nominal. Definition 26 says for every observable transition, a link is created from the beginning state to the target state on the next row, which represents that the event of this transition is observed in this time window of length k; for every unobservable transition, a link is created from the beginning state to the target state on the same row, which represents that no event is observed; for each faulty state at the end of the time window, a link ε is created from F_4 to faulty states only at the beginning of the time window, i.e. F_0; for each nominal state at the end of the

time window, a link ε is created to every nominal state at the beginning of the time window, i.e. C_0, A_0, B_0, D_0.

To verify the precision of Al_1 w.r.t. M_3, the simulation for Al_1 needs to be synchronised with M_3. A twin plant is then built for this synchronisation. Any ambiguous path is checked on the twin plant. There exists an ambiguous path, and it contains a loop, i.e. $AA_0 \xrightarrow{a} CC_1 \xrightarrow{b} AA_2 \xrightarrow{a} CC_3 \xrightarrow{b} AA_4 \xrightarrow{c} DF_1 \xrightarrow{c} DF_2 \xrightarrow{c} DF_3 \xrightarrow{c} DF_4 \xrightarrow{c} DF_1 \ldots$. Therefore, Al_1 is not precise w.r.t. M_3 when $k = 4$ by Theorem 1 of Chapter 3.

□

Theorem 5 proves the correctness of the simulation for M and Al_1.

Theorem 5 (Simulation for a DES model M and Al_1) *The Al_1-simulation of a diagnosis model M as defined in Definition 26 is the simulation for M and Al_1.*

Proof of Theorem 5 Given a model M and any observation sequence $o \in {\Sigma_o}^*$, a path on the Al_1-simulation of M with $|o|$ observations, denoted as p_1, is the concatenation of $\lceil |o|/k \rceil$ paths of length k, denoted as p_2, such that the diagnostic information at the end of p_1 is the same as the one at the beginning of p_2. Therefore, all paths in the simulation that are consistent with the observations will end in a faulty state iff all paths of one window end in a faulty state, i.e. $\forall o \in {\Sigma_o}^*,\ Al_1(M, o, T_{Al_1}) = \Delta(k\text{-simulation}(M, Al_1), o)$. This satisfies Definition 11 for the simulation of a diagnostic algorithm and a DES model.

□

This work discusses the complexity of a simulation for a DES model M and Al_1. The size of the simulation is k times that of M, where k is the time window size. Thus, verifying the precision of Al_1 w.r.t. M using the twin plant method remains polynomial. Remember that a simulation is only used to verify the precision of a diagnostic algorithm w.r.t. a DES model, and not for diagnosis.

4.3.2 Precision of Al_2 w.r.t. a DES Model

With the help of Definition 26, the simulation for Al_2 is defined as follows.

Definition 27 (Al_2-Simulation) *Given the time window size k, assuming k is a multiple of 2, the Al_2-simulation for model $M = \langle Q, \Sigma, T, I, L \rangle$ is the automaton $M' = M'_1 \otimes M'_2$ where M'_1 is the Al_1-simulation as defined in Definition 26 and $M'_2 = \langle Q'_2, \Sigma'_2, T'_2, I'_2, L'_2 \rangle$ where*

- $Q'_2 = Q \times \{0, 1, 2, \ldots, k\}$;

- $\Sigma_2' = \Sigma \cup \{\varepsilon\}$;
- $T_2' = T_{u_2} \cup T_{o_2} \cup T_{\varepsilon_2}$ *is defined by:*
  - $T_{u_2} = \{\langle\langle q, i\rangle, u, \langle q', i\rangle\rangle \mid i \in [\frac{k}{2}, \frac{k}{2}+1, \frac{k}{2}+2, \ldots, k, 0, 1, 2, \ldots, \frac{k}{2}-1] \wedge \langle q, u, q'\rangle \in T \wedge u \in \Sigma \setminus \Sigma_o\}$;
  - $T_{o_2} = \{\langle\langle q, i\rangle, o, \langle q', i+1\rangle\rangle \mid i \in [\frac{k}{2}, \frac{k}{2}+1, \frac{k}{2}+2, \ldots, k, 0, 1, 2, \ldots, \frac{k}{2}-2] \wedge \langle q, o, q'\rangle \in T \wedge o \in \Sigma_o\}$;
  - $T_{\varepsilon_2} = \{\langle\langle q, \frac{k}{2}-1\rangle, \varepsilon, \langle q', \frac{k}{2}\rangle\rangle \mid L(q) = L(q')\}$;
- $I_2' = I \times \{\frac{k}{2}\}$;
- $L_2'(\langle q, i\rangle) = L(q)$.

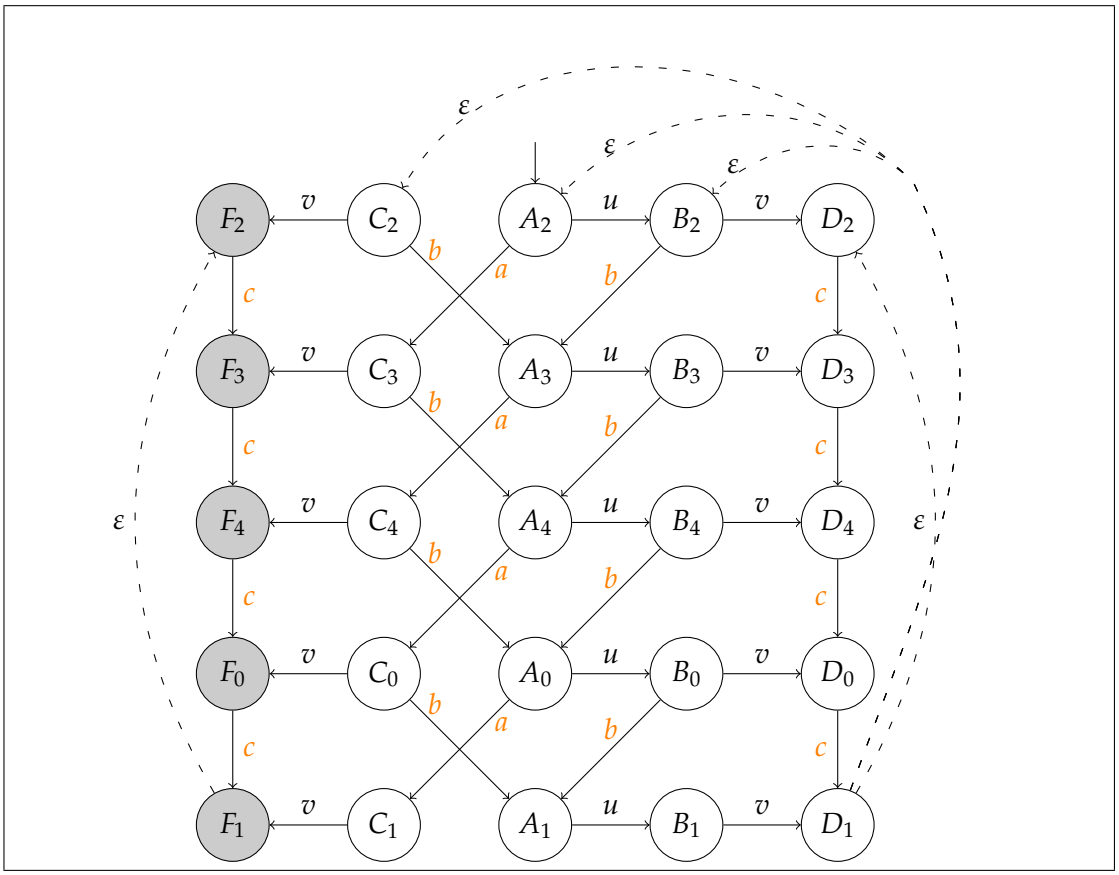

Figure 4.5: Part of the simulation for the DES model M_3 in Fig. 4.1 and Al_2, where $k = 4$. Dotted lines also need to link A_1 to A_2, B_2, C_2 and D_2. Same applies to B_1 and C_1.

Example 2: Simulation for M_3 and Al_2 Fig. 4.4 shows the Al_1-simulation, and Fig. 4.5 shows M'_2. To construct the simulation for Al_2, Fig. 4.4 and Fig. 4.5 are synchronised in order to compute the Al_2-simulation. To verify the precision of Al_2 w.r.t. M_3, the simulation for Al_2 needs to be synchronised with M_3 in Fig. 4.1. The twin plant is then constructed for this synchronisation. Any ambiguous path is checked on the twin plant. There is no loop on any ambiguous path since there is no ambiguous path. Therefore, Al_2 is precise w.r.t. M_3 by Theorem 1 when k is 4.

□

Theorem 6 proves the correctness of the simulation for M and Al_2.

Theorem 6 (Simulation for a DES model M and Al_2) *The Al_2-simulation of a diagnosis model M as defined in Definition 27 is the simulation for M and Al_2.*

Proof of Theorem 6 Given a DES model M and any observation sequence $o \in {\Sigma_o}^*$, a path on the Al_2-simulation of M with $|o|$ observations denoted as p_1 is the concatenation of $\lceil |o|/k \rceil$ paths of length k, where p_2 denotes the paths in the Al_1-simulation and p_3 denotes the paths in M'_2. Therefore, all paths in the simulation that are consistent with the observations will end in a faulty state iff all paths of one time window end in a faulty state, i.e. $\forall o \in {\Sigma_o}^*,\ Al_2(M, o, T_{Al_2}) = \Delta(Al_2\text{-simulation}(M, Al_2), o)$. This satisfies Definition 11 for the simulation of a diagnostic algorithm and a DES model.

□

This work discusses the complexity of a simulation for a DES model M and Al_2. Let k be the time window size. Verifying the precision of Al_2 w.r.t. M using the twin plant method remains polynomial. Remember that a simulation is only used to verify the precision of a diagnostic algorithm w.r.t. a DES model, and not for diagnosis.

4.3.3 Precision of Al_p w.r.t. a DES Model

With the help of Definition 26, the simulation for Al_p is defined as follows. The simulation for Al_p is defined first because the Al_3-simulation will use part of the simulation for Al_p.

Definition 28 (Al_p-Simulation) *Given the time window size k, the Al_p-simulation for model $M = \langle Q, \Sigma, T, I, L \rangle$ is the automaton $M' = M'_1 \otimes M'_2 \otimes M'_3 \otimes \cdots \otimes M'_k \otimes M'_{k+1}$ where M'_1 is the Al_1-simulation as defined in Definition 26 and for $j \in [2, k+1]$, $M'_j = \langle Q'_j, \Sigma'_j, T'_j, I'_j, L'_j \rangle$ where*

- $Q'_j = Q \times \{0, 1, 2, \ldots, k\}$;

- $\Sigma'_j = \Sigma \cup \{\varepsilon\}$;
- $T'_j = T_{u_j} \cup T_{o_j} \cup T_{\varepsilon_j}$ *is defined by:*
  - $T_{u_j} = \{\langle\langle q, i\rangle, u, \langle q', i\rangle\rangle \mid i \in [j, j+1, j+2, \ldots, k, 0, 1, 2, \ldots, j-1] \wedge \langle q, u, q'\rangle \in T \wedge u \in \Sigma \setminus \Sigma_o\}$;
  - $T_{o_j} = \{\langle\langle q, i\rangle, o, \langle q', i+1\rangle\rangle \mid i \in [j, j+1, j+2, \ldots, k, 0, 1, 2, \ldots, j-2] \wedge \langle q, o, q'\rangle \in T \wedge o \in \Sigma_o\}$;
  - $T_{\varepsilon_j} = \{\langle\langle q, j-1\rangle, \varepsilon, \langle q', j\rangle\rangle \mid L(q) = L(q')\}$;
- $I'_j = I \times \{j-1\}$;
- $L'_j(\langle q, i\rangle) = L(q)$.

Example 3: Simulation for a DES model M_3 and Al_p This work examines the simulation for Al_p, which will be useful in the next example for Al_3. The simulation for Al_p is very similar to the simulation for Al_2 as in Fig. 4.5. The difference is that five such models are needed. The first model has index 0 for each state in the first row; the second model has index 1 for each state in the first row; the third model has index 2 for each state in the first row; the fourth model has index 3 for each state in the first row; the fifth model has index 4 for each state in the first row. All five models are synchronised to form the simulation for Al_p. Al_p is precise w.r.t. M_3 when k is 4. □

Theorem 7 proves the correctness of the simulation for M and Al_p.

Theorem 7 (Simulation for a DES model M and Al_p) *The Al_p-simulation of a diagnosis model M as defined in Definition 28 is the simulation for M and Al_p.*

Proof of Theorem 7 Given a DES model M and any observation sequence $o \in {\Sigma_o}^*$, a path on the Al_p-simulation of M with $|o|$ observations, denoted as p_1, is the concatenation of $\lceil |o|/k \rceil$ paths of length k, where p_2 denotes the paths in the Al_1-simulation, and for $j \in [2, k+1]$, p_j denotes the paths in M'_j. Therefore, all paths in the simulation that are consistent with the observations will end in a faulty state iff all paths of one window end in a faulty state, i.e. $\forall o \in {\Sigma_o}^*$, $Al_p(M, o, T_{Al_p}) = \Delta(Al_p\text{-simulation}(M, Al_p), o)$. This satisfies Definition 11 for the simulation of a diagnostic algorithm and a DES model.

□

This work discusses the complexity of a simulation for a DES model M and Al_p. Let k be the time window size. Verifying the precision of Al_p w.r.t. M using the twin plant method remains polynomial. Remember that a simulation is only used to verify the precision of a diagnostic algorithm w.r.t. a DES model, and not for diagnosis.

4.3.4 Precision of Al_3 w.r.t. a DES Model

With the help of Definition 26 and 27, the simulation for Al_3 is defined as follows.

Definition 29 (Al_3-Simulation) *Given the time window size k, the Al_3-simulation is defined based on the following three cases.*

Case 1: $d = 1$. In this situation, the Al_3-simulation is the same as the Al_p-simulation, which is defined in Definition 28.

Case 2: $1 < d < k$. The Al_3-simulation for model $M = \langle Q, \Sigma, T, I, L\rangle$ is the automaton $M' = M'_1 \otimes M'_2$ where M'_1 is the Al_1-simulation as defined in Definition 26 and $M'_2 = \langle Q'_2, \Sigma'_2, T'_2, I'_2, L'_2\rangle$ where

- $Q'_2 = Q \times \{d, d+1, d+2, \ldots, k, 0, 1, 2, \ldots, d-1\}$;
- $\Sigma'_2 = \Sigma \cup \{\varepsilon\}$;
- $T'_2 = T_{u_2} \cup T_{o_2} \cup T_{\varepsilon_2}$ *is defined by:*
  - $T_{u_2} = \{\langle\langle q, i\rangle, u, \langle q', i\rangle\rangle \mid i \in \{d, d+1, d+2, \ldots, k, 0, 1, 2, \ldots, d-1\} \wedge \langle q, u, q'\rangle \in T \wedge u \in \Sigma \setminus \Sigma_o\}$;
  - $T_{o_2} = \{\langle\langle q, i\rangle, o, \langle q', i+1\rangle\rangle \mid i \in \{d, d+1, d+2, \ldots, k, 0, 1, 2, \ldots, d-2\} \wedge \langle q, o, q'\rangle \in T \wedge o \in \Sigma_o\}$;
  - $T_{\varepsilon_2} = \{\langle\langle q, d-1\rangle, \varepsilon, \langle q', d\rangle\rangle \mid L(q) = L(q')\}$;
- $I'_2 = I \times \{d\}$;
- $L'_2(\langle q, i\rangle) = L(q)$.

Case 3: $d \geq k$. The Al_3-simulation is the same as the simulation for Al_1 as defined in Definition 26.

Example 4: Simulation for a DES model M_3 and Al_3 Al_3 is categorised into three cases depending on the value of d.

Case 1: $d = 1$. In this situation, Al_3 is the same as Al_p. Al_3 is precise w.r.t. M_3 for $k = 4$ and $d = 1$.

Case 2: $1 < d < k$. The simulation for Al_3 is similar to the simulation for Al_2 as in Fig. 4.5. The difference is that the index for each state in the first row needs to reflect d, i.e. the number of observations that are skipped. For example, if d is 3, then the first row should have 3 index for each state. Al_3 is precise w.r.t. M_3 for $k = 4$ and $1 < d < k$.

Case 3: $d \geq k$. The simulation for Al_3 is the same as the simulation for Al_1 as in Fig. 4.4. Al_3 is not precise w.r.t. M_3 for $k = 4$ and $d \geq k$.

□

Theorem 8 proves the correctness of the simulation for M and Al_3.

Theorem 8 (Simulation for a DES model M and Al_3) *The Al_3-simulation of a diagnosis model M as defined in Definition 29 is the simulation for M and Al_3.*

Proof of Theorem 8 Given a DES model M and any observation sequence $o \in {\Sigma_o}^*$, case 1 is proved using Theorem 7. Also, case 3 is proved using Theorem 5.

For case 2, a path on the Al_3-simulation of M with $|o|$ observations, denoted as p_1, is the concatenation of $\lceil |o|/k \rceil$ paths of length k, where p_2 denotes the paths in the Al_1-simulation, and p_3 denotes the paths in M_2'. Therefore, all paths in the simulation that are consistent with the observations will end in a faulty state iff all paths of one time window end in a faulty state, i.e. $\forall o \in {\Sigma_o}^*,\ Al_3(M, o, T_{Al_3}) = \Delta(Al_3\text{-simulation}(M, Al_3), o)$. This satisfies Definition 11 for the simulation of a diagnostic algorithm and a DES model.

□

This work discusses the complexity of a simulation for a DES model M and Al_3. Let k be the time window size. Verifying the precision of Al_3 w.r.t. M using the twin plant method remains polynomial. Remember that a simulation is only used to verify the precision of a diagnostic algorithm w.r.t. a DES model, and not for diagnosis.

4.4 Comparison of Four Implementations of IWAs

This section compares the four implementations of IWAs. First, this work defines *coverage* of one time window on another one.

Definition 30 (Coverage of Two Time Windows) *Given two time windows $[i, j]$ and $[i', j']$, if $i' \leq i$ and $j' \geq j$, then $[i', j']$* covers *$[i, j]$, denoted as $[i, j] \sqsubseteq [i', j']$.*

Based on Definition 30, this work defines *coverage* of a set of time windows on another set.

Definition 31 (Coverage of Two Sets of Time Windows) *Given two set of time windows T_1 and T_2, if there is always at least one time window in T_2 that covers every time window in T_1, then T_2* covers *T_1, denoted as $T_1 \sqsubseteq T_2$, i.e.*

$$\textit{if } \forall [i, j] \in T_1,\ \exists [i', j'] \in T_2 \textit{ such that } [i', j'] \textit{ covers } [i, j],\ \textit{then } T_1 \sqsubseteq T_2.$$

This work studies the properties of coverage in the context of single-window algorithms and window-based diagnostic algorithms, by proposing Lemma 2 and 3.

Lemma 2 *Given time windows $[i,j]$ and $[i',j']$, if $[i,j] \sqsubseteq [i',j']$, and $W_{i,j}(M,o)$ evaluates to F, then $W_{i',j'}(M,o)$ evaluates to F.*

Proof of Lemma 2 Given two time windows $[i,j]$ and $[i',j']$, suppose $[i,j] \sqsubseteq [i',j']$, and $W_{i,j}(M,o)$ evaluates to F. Then, $W_{i',j'}(M,o)$ can be represented by a window-based diagnostic algorithm, i.e. $W_{i',j'}(M,o) \equiv A(M,o,T)$ where

$$T = \begin{cases} \{[i',i-1],[i,j],[j+1,j']\} & \text{if } i < i' \text{ and } j' > j \\ \{[i,j],[j+1,j']\} & \text{if } i = i' \text{ and } j' > j \\ \{[i',i-1],[i,j]\} & \text{if } i < i' \text{ and } j' = j. \end{cases}$$

$$\exists [i,j] \in T,\ W_{i,j}(M,o) = F \tag{4.7}$$
$$\Rightarrow A(M,o,T) = F \tag{4.8}$$
$$\Rightarrow W_{i',j'}(M,o) = F. \tag{4.9}$$

□

Lemma 3 *Given two sets of time windows T_1 and T_2, if $T_1 \sqsubseteq T_2$ and $A(M,o,T_1)$ evaluates to F, then $A(M,o,T_2)$ evaluates to F.*

Proof of Lemma 3 Given two sets of time windows T_1 and T_2, suppose $T_1 \sqsubseteq T_2$, and $A(M,o,T_1)$ evaluates to F. Then,

$$\exists [i,j] \in T_1,\ W_{i,j}(M,o) = F \text{ and } T_1 \sqsubseteq T_2 \tag{4.10}$$
$$\Rightarrow \exists [i',j'] \in T_2,\ W_{i',j'}(M,o) = F \tag{4.11}$$
$$\Rightarrow A(M,o,T_2) = F. \tag{4.12}$$

□

Based on Lemma 2 and 3, Theorem 9 proposes the *precision* relation between two time windows.

Theorem 9 (Precision Relation) *Given a DES model M, two window-based diagnostic algorithms A with T and A' with T', if $T \sqsubseteq T'$, and (A with T) is precise w.r.t. M, then (A' with T') is precise w.r.t. M. The* precision relation *is denoted as (A with T) $\Rightarrow$ (A' with T').*

Proof of Theorem 9 Given a DES model M, two window-based diagnostic algorithms A with T and A' with T', since $T \sqsubseteq T'$ and $A(M,o,T)$ evaluates to F, Lemma 3 says that $A'(M,o,T')$ evaluates to F.

Definition 10 of Chapter 3 says that a diagnostic algorithm A is *precise* for a diagnosis model M w.r.t. a system model M' if the following holds:

$$\exists n \in \mathbf{N},\ \forall s \in \mathcal{L}'_F,\ \forall t \in \Sigma^*,\ \left(st \in \mathcal{L}' \wedge |t| \geq n \Rightarrow A(M, obs(st)) = F\right).$$

Since $T \sqsubseteq T' \wedge A(M, o, T) = F \Rightarrow A'(M, o, T') = F$ holds, (A' with T') is precise w.r.t. the DES model, i.e. (A with T) $\Rightarrow$ (A' with T').

□

Based on Lemma 2, Lemma 3, and Theorem 9, given a DES model M, Property 2 examines the precision relations among Al_p, Al_1, and Al_2 w.r.t. M.

Property 2 (Precision Comparisons)

$$(Al_1 \textit{ with } T_{Al_1}(i,k,k) \textit{ is precise}) \Rightarrow (Al_2 \textit{ with } T_{Al_2}(i,k,\frac{k}{2}) \textit{ is precise}) \tag{4.13}$$

$$(Al_2 \textit{ with } T_{Al_2}(i,k,\frac{k}{2}) \textit{ is precise}) \Rightarrow (Al_p \textit{ with } T_{Al_p}(i,k,1) \textit{ is precise}) \tag{4.14}$$

$$(Al_1 \textit{ with } T_{Al_1}(i,k,k) \textit{ is precise}) \Rightarrow (Al_p \textit{ with } T_{Al_p}(i,k,1) \textit{ is precise}) \tag{4.15}$$

$$(Al_1 \textit{ with } T_{Al_1}(i,k,k) \textit{ is precise}) \Rightarrow (Al_1 \textit{ with } T_{Al_1}(i,x\times k, x\times k) \textit{ is precise}) \tag{4.16}$$

$$\textit{where } x \geq 2 \wedge x \in \mathbf{N} \tag{4.17}$$

$$(Al_2 \textit{ with } T_{Al_2}(i,k,\frac{k}{2}) \textit{ is precise}) \Rightarrow (Al_2 \textit{ with } T_{Al_2}(i,x\times k, \frac{x\times k}{2}) \textit{ is precise}) \tag{4.18}$$

$$\textit{where } x \geq 2 \wedge x \in \mathbf{N} \tag{4.19}$$

$$(Al_p \textit{ with } T_{Al_p}(i,k,1) \textit{ is precise}) \Rightarrow (Al_p \textit{ with } T_{Al_p}(i,k+x,1) \textit{ is precise}) \tag{4.20}$$

$$\textit{where } x \geq 1 \wedge x \in \mathbf{N} \tag{4.21}$$

$$(Al_p \textit{ with } T_{Al_p}(i,k,1) \textit{ is precise}) \Rightarrow (Al_2 \textit{ with } T_{Al_2}(i,x\times k, \frac{x\times k}{2}) \textit{ is precise}) \tag{4.22}$$

$$\textit{where } x \geq 2 \wedge x \in \mathbf{N} \tag{4.23}$$

The converse of each above line does not always hold.

Proof of Property 2 On each line, given a DES model M, Al with $T_{Al}(i,k)$ on the left hand side, and Al' with $T_{Al'}(i,k)$ on the right hand side, by the definition of Al and that of Al', $T_{Al}(i,k) \sqsubseteq T_{Al'}(i,k)$. By Theorem 9, Al with $T_{Al}(i,k) \Rightarrow Al'$ with $Al'(i,k)$ w.r.t. M.

□

In addition, Property 3 examines the incomparable precision relations.

Property 3 (Incomparable Precision Relations)

$$(Al_p \text{ with } T_{Al_p}(i,k,1) \text{ is not precise}) \nRightarrow (Al_p \text{ with } T_{Al_p}(i,k+x,1) \text{ is precise}) \quad (4.24)$$

$$\text{where } x \geq 1 \wedge x \in \mathbf{N} \quad (4.25)$$

$$(Al_1 \text{ with } T_{Al_1}(i,k,k) \text{ is not precise}) \nRightarrow (Al_1 \text{ with } T_{Al_1}(i,k+x,k+x) \text{ is precise}) \quad (4.26)$$

$$\text{where } x \geq 1 \wedge x \in \mathbf{N} \quad (4.27)$$

$$(Al_2 \text{ with } T_{Al_2}(i,k,\frac{k}{2}) \text{ is not precise}) \nRightarrow (Al_2 \text{ with } T_{Al_2}(i,k+x,\frac{k+x}{2}) \text{ is precise}) \quad (4.28)$$

$$\text{where } x \geq 1 \wedge x \in \mathbf{N} \quad (4.29)$$

Also,

$$(Al_p \text{ with } T_{Al_p}(i,k+x,1) \text{ is not precise}) \nRightarrow (Al_p \text{ with } T_{Al_p}(i,k,1) \text{ is precise}) \quad (4.30)$$

$$\text{where } x \geq 1 \wedge x \in \mathbf{N} \quad (4.31)$$

$$(Al_1 \text{ with } T_{Al_1}(i,k+x,k+x) \text{ is not precise}) \nRightarrow (Al_1 \text{ with } T_{Al_1}(i,k,k) \text{ is precise}) \quad (4.32)$$

$$\text{where } x \geq 1 \wedge x \in \mathbf{N} \quad (4.33)$$

$$(Al_2 \text{ with } T_{Al_2}(i,k+x,\frac{k+x}{2}) \text{ is not precise}) \nRightarrow (Al_2 \text{ with } T_{Al_2}(i,k,\frac{k}{2}) \text{ is precise}) \quad (4.34)$$

$$\text{where } x \geq 1 \wedge x \in \mathbf{N} \quad (4.35)$$

Property 3 states that for IWAs Al_p, Al_1, and Al_2, increasing the number of observations in one time window does not guarantee to enhance the precision. Also, reducing the number of observations in one time window does not guarantee to maintain the precision.

Property 3 can be proved by counterexamples.

Input	Time Window Size	Time Window	Diagnosis	Output
x,c,c,c,b,b,b,b	3	(x,c,c)	N	N (imprecise)
		(c,c,c)	N	
		(c,c,b)	N	
		(c,b,b)	N	
		(b,b,b)	N	
		(b,b,b)	N	
x,c,c,c,b,b,b,b	4	(x,c,c,c)	N	N (imprecise)
		(c,c,c,b)	N	
		(c,c,b,b)	N	
		(c,b,b,b)	N	
		(b,b,b,b)	N	

Table 4.7: Examples of Al_p running on M_4 in Fig. 4.2 where $i = 1$ and $k = 3, 4$

For 4.18, Tab. 4.7 shows the diagnostic results of Al_p running on M_4 in Fig. 4.2 where $i = 1$ and $k = 3, 4$. This counterexample shows that the following double negation does not hold.

$$(Al_p \text{ with } T_{Al_p}(i, 4, 1) \text{ is not precise}) \Rightarrow (Al_p \text{ with } T_{Al_p}(i, 3, 1) \text{ is precise})$$

Therefore, $(Al_p$ with $T_{Al_p}(i, 3, 1)$ is not precise$) \not\Rightarrow (Al_p$ with $T_{Al_p}(i, 4, 1)$ is precise$)$ is valid.

For 4.24, the following double negation does not hold.

$$(Al_p \text{ with } T_{Al_p}(i, 3, 1) \text{ is not precise}) \Rightarrow (Al_p \text{ with } T_{Al_p}(i, 4, 1) \text{ is precise})$$

Therefore, $(Al_p$ with $T_{Al_p}(i, 4, 1)$ is not precise$) \not\Rightarrow (Al_p$ with $T_{Al_p}(i, 3, 1)$ is precise$)$ is valid.

Input	Time Window Size	Time Window	Diagnosis	Output
x, c, c, c, b, b, b, b	3	(x, c, c)	N	N (imprecise)
		(c, b, b)	N	
x, c, c, c, b, b, b, b	4	(x, c, c, c)	N	N (imprecise)
		(b, b, b, b)	N	

Table 4.8: Examples of Al_1 running on M_4 in Fig. 4.2 where $i = 1$ and $k = 3, 4$

For 4.20, Tab. 4.8 shows the diagnostic results of Al_1 running on M_4 in Fig. 4.2 where $i = 1$ and $k = 3, 4$. This counterexample shows that the following double negation does not hold.

$$(Al_1 \text{ with } T_{Al_1}(i, 4, 4) \text{ is not precise}) \Rightarrow (Al_1 \text{ with } T_{Al_1}(i, 3, 3) \text{ is precise})$$

Therefore, $(Al_1$ with $T_{Al_1}(i, 3, 3)$ is not precise$) \not\Rightarrow (Al_1$ with $T_{Al_1}(i, 4, 4)$ is precise$)$ is valid.

For 4.26, the following double negation does not hold.

$$(Al_1 \text{ with } T_{Al_1}(i, 3, 3) \text{ is not precise}) \Rightarrow (Al_1 \text{ with } T_{Al_1}(i, 4, 4) \text{ is precise})$$

Therefore, $(Al_1$ with $T_{Al_1}(i, 4, 4)$ is not precise$) \not\Rightarrow (Al_1$ with $T_{Al_1}(i, 3, 3)$ is precise$)$ is valid.

For 4.22, Tab. 4.9 shows the diagnostic results of Al_2 running on M_4 in Fig. 4.2 where $i = 1$ and $k = 2, 4$. This counterexample shows that the following double

Input	Time Window Size	Time Window	Diagnosis	Output
b, x, c, c, c, b, b, b	2	(b, x)	N	N (imprecise)
		(x, c)	N	
		(c, c)	N	
		(c, c)	N	
		(c, b)	N	
		(c, c)	N	
		(c, c)	N	
	4	(b, x, c, c)	N	N (imprecise)
		(c, c, c, b)	N	
		(c, b, b, b)	N	

Table 4.9: Examples of Al_2 running on M_4 in Fig. 4.2 where $i = 1$ and $k = 2, 4$

negation does not hold.

$$(Al_2 \text{ with } T_{Al_2}(i, 4, 2) \text{ is not precise}) \Rightarrow (Al_2 \text{ with } T_{Al_2}(i, 2, 1) \text{ is precise})$$

Therefore, $(Al_2$ with $T_{Al_2}(i, 2, 1)$ is not precise$) \not\Rightarrow (Al_2$ with $T_{Al_2}(i, 4, 2)$ is precise$)$ is valid.

For 4.28, the following double negation does not hold.

$$(Al_2 \text{ with } T_{Al_2}(i, 2, 1) \text{ is not precise}) \Rightarrow (Al_2 \text{ with } T_{Al_2}(i, 4, 2) \text{ is precise})$$

Therefore, $(Al_2$ with $T_{Al_2}(i, 4, 2)$ is not precise$) \not\Rightarrow (Al_2$ with $T_{Al_2}(i, 2, 1)$ is precise$)$ is valid.

Finally, this section examines the maximum number of observations in one time window of Al_p, Al_1, and Al_2 that guarantees the precision. Given a diagnosable DES model M, and the number of states is $|Q|$, Yoo and Lafortune [2002] proved that the maximum number of transitions required to diagnose a fault is $|Q|^2$. Therefore, if the number of observations in one time window of Al_p, Al_1, and Al_2 is at least $|Q|^2$, then the precision w.r.t. M holds.

4.5 Application of Alarm Log Handling

One major application is intelligent alarm log handling. Alarm log handling is a problem that operators of a large-scale system are facing every day [Bauer et al., 2011], e.g. electricity distribution and telecommunication control. Examples of alarms are faults, warnings, and irregular performance. The amount of alarms in a faulty scenario tends to be overwhelming for human operators to diagnose and

handle in a real-time and critical environment. It is a complicated task to diagnose a large and complex system that is running for a long period of time. Sampath et al. [1995] is the original work and it is accurate. However, exact diagnosis requires the entire sequence of observations starting from the beginning of system operations, which is not practical for a real-world application.

This work has proposed window-based diagnosis. IWAs presented in this chapter have three advantages compared to using exact diagnosis, i.e. improved flexibility, reduced complexity, and the capability of dealing with the masking issue. First, IWAs improve the diagnostic flexibility because it only runs on individual time windows. An IWA allows to skip irrelevant time windows. Also, each time windows can be diagnosed separately. This flexibility allows a diagnoser to jump to one particular time window and perform diagnosis. Therefore, an IWA can diagnose a system status based on the observations from a recent period of time. In comparison, exact diagnosis requires a log of system observations starting from the initial state to derive the belief state at the end of the observations. A log of the system observations may be gathered over a long period of time. As a result, the size of a log with all observations since the beginning of the system operation is likely to be large, which has a significant impact on the computational complexity of exact diagnosis.

Second, it is unnecessary to analyse from the beginning of the system operation to diagnose a system. Using an IWA, it will be sufficient to analyse the recent history of observations to diagnose the status of a system. Furthermore, using time windows is beneficial to organise a sequence of observations by manageable pieces, which facilitates the identification, diagnosis, and tracking of each time window. Therefore, time windows allow an intelligent alarm log handling system to reduce the diagnostic complexity, capture the essential information within a time window to perform diagnosis, and reuse previously computed results.

Third, IWAs are advantageous to deal with the masking issue. Rozé and Cordier [2002b] has defined the masking issue . When a component sends an alarm signal, it must be transmitted by multiple components before reaching the supervision centre. If one of the components in the transmission path is not in the transmitting mode, then the supervision centre will not receive this alarm. Consequently, a fault in some sensor is likely to result in incomplete gathering of all events from the sensors. Nevertheless, IWAs are less subject to the masking issue of a sequence of observations gathered by sensors. This is because the system states are reset between the neighbouring time windows, and IWAs are capable to diagnose one or more independent time windows.

4.6 Summary

This chapter reviews the existing approaches for DES diagnosis. The size of a belief state makes the existing approaches not practical for a real-world application because it has been proved that the size of a belief state grows exponentially w.r.t. the number of system states [Rintanen, 2007]. This chapter proposes a new class of diagnostic algorithms, called IWAs, to diagnose a DES model. It also examines their precision w.r.t. a DES model. IWAs aim to slice a sequence of observations into time windows, and diagnose them independently. This new approach does not require maintaining a precise tracking of the system states, reduces computational complexity of diagnosis, and has the advantage to handle intermittent loss of observations which may occur in the transmission of observations. This chapter provides the definitions for IWAs. Definition 9 of Chapter 3 defines the monotonicity and correctness properties for a diagnostic algorithm. This chapter proves that IWAs preserve both properties. It then demonstrates four implementations of IWAs, i.e. Al_p, Al_1, Al_2, and Al_3. Furthermore, this chapter studies the precision relations among the four implementations of IWAs.

On the other hand, IWAs may lead to imprecise diagnostic results. Based on the simulation method proposed in Chapter 3 to verify the precision of a diagnostic algorithm w.r.t. a DES model, this chapter demonstrates how to construct a simulation, which is a modified model to simulate how an IWA works on a DES model. The output simulation is then synchronised with the system model. Finally, the twin plant method is used to verify the precision of an IWA w.r.t. the model.

In conclusion, this chapter proposes a diagnostic strategy, which differs from the conservative strategy for exact diagnosis that diagnostic algorithms that are only applied on the very last events of the observable flow and *forget* about the past. Although this approach may cause precision loss when both current and past observations are necessary to understand the system behaviour, the precision can be measured by applying the simulation method described in Chapter 3.

Chapter 5 will propose a compromise between exact diagnosis and IWAs. It will propose two new window-based diagnostic algorithms, called Time-Window Algorithms (TWAs). TWAs extend IWAs such that TWAs carry over some information about the current state of the system from one time window to the next. TWAs look for the minimum piece of information to *remember* from the past so that a window-based algorithm will certainly ensure the same precision as using an exact diagnostic algorithm.

Chapter 5

Diagnosis of DES using Time-Window Algorithms

Given a flow of observable events generated by the underlying system, the on-line DES diagnosis problem consists in determining whether the DES is operating normally or not based on its behavioural model. Section 2.1 of Chapter 2 reviews the existing work of on-line diagnosis. Furthermore, Section 3.1.1 of Chapter 3 defines the exact diagnosis. The existing approaches in the literature use the conservative strategy of exact diagnosis, with the exception of diagnosis by SAT. The main drawback of exact diagnosis is the inability to *follow* the observable flow for a large system due to the exponential size of the generated belief states, and therefore the temporal complexity to handle them. Although diagnosis using SAT computes one trace in the system for an observation sequence, the complexity of a SAT problem is exponential to the number of propositional variables, which is linear to the number of state variables [Grastien et al., 2007].

Chapter 4 proposes the *window-based approach*, which is different to the conservative strategy, e.g. diagnostic algorithms that are only applied on the very last events of the observable flow and *forget* about the past. Obviously, this approach may cause precision loss when both current and past observations are necessary to understand the system behaviour. This precision loss can be measured using the simulation method described in Chapter 3.

This chapter proposes a compromise between the two extreme strategies. This work looks for the minimum piece of information to *remember* from the past, called *abstracted belief state*, so that a window-based algorithm will certainly ensure the same precision as any conservative algorithms. The first contribution of this chapter is to formally present two new window-based algorithms, called Time-Window Algorithms (TWAs). TWAs are extensions to IWAs presented in Chapter 4. TWAs carry over some information about the current state of the system from one time window to the next. The second contribution of this chapter is to describe how the precision

of these new TWAs is verified w.r.t. a DES model using the simulation method as presented in Chapter 3. Finally, this chapter proposes a formal procedure to minimise the amount of information that a window-based algorithm needs to carry over to ensure no precision loss.

This chapter is organised as follows. Section 5.1 recalls the necessary background about the diagnosis problem, and reviews the existing window-based algorithms IWAs. Section 5.2 proposes new diagnostic algorithms, called Time-Window Algorithm (TWAs), and presents the first case of TWA, called Al_5. Section 5.3 presents the second case of TWA, called Al_6. Section 5.4 demonstrates how to verify the precision of the two TWAs. Section 5.5 discusses the symbolic implementation and the procedure that minimises the amount of information required for a TWA to be precise. Section 5.6 concludes this chapter, and provides an outline for the future work.

5.1 Background of Independent-Window Algorithms (IWAs)

TWAs are extensions to IWAs presented in Chapter 4. This section reviews the existing IWAs. A diagnosis problem is defined on a model and an observation sequence. In general, a diagnoser should indicate whether the system is in the nominal mode, in the faulty mode, or it cannot decide, i.e. the system status is ambiguous. Assumption 4 of Chapter 3 states that the diagnostic policy of this work does not distinguish between the nominal mode and the ambiguous mode, i.e. a diagnoser assumes that the system is not faulty unless proved otherwise.

Chapter 4 proposes IWAs, which are diagnostic algorithms that slice the flow of observations $\Theta = o_1, o_2, o_3, \ldots, o_n$ into small and independent time windows θ_t to be diagnosed independently. A time window θ is a sub-sequence $\Theta[i, j] = o_i, o_{i+1}, o_{i+2}, \ldots, o_j$ of the actual observations Θ. The diagnosis of a time window θ consists in determining whether there exists a nominal trace that generates this sub-sequence of observations. IWAs diagnose each time window separately, and return that the system is faulty as soon as the fault is diagnosed in one time window.

Chapter 4 also presents four implementations of IWAs, namely Al_p, Al_1, Al_2, and Al_3. Each IWA diagnoses k observations for one time window, and moves to another time window without keeping any information. These four implementations of IWAs differ only in the time window selection. Al_p diagnoses on sliding time windows, i.e. the first time window is $\Theta[1, k]$, the second is $\Theta[2, k+1]$, etc. A sliding time window moves by one observed event at a time. Al_1 slices a sequence of observations into consecutive time windows of identical length. Al_2 ensures that the time windows overlap so that short observations of $o_i, o_{i+1}, o_{i+2}, \ldots, o_{i+k-1}$ always appear in at least one time window. Al_3 constructs "sample" time windows such that

not every observation appears in a time window.

There are advantages and disadvantages of IWAs. On the one hand, IWAs aim at improving the flexibility of diagnosis because time windows are diagnosed separately. Also, the complexity of diagnosis is reduced because the size of the time windows is bounded. On the other hand, the precision of IWAs may be reduced as the links between the time windows are lost. For instance, this happens when a fault can be diagnosed only by observing two specific events; however, these two events cannot be guaranteed to appear in the same time window.

5.2 Time-Window Algorithm (TWA) Al_5

This chapter proposes two Time-Window Algorithms, namely Al_5 and Al_6. TWAs are extensions to the IWAs proposed in Chapter 4. The principle of these TWAs is to remember from one time window to the next some of the knowledge about the current belief state. Al_5 remembers the key information from the previous consecutive time window whereas Al_6 remembers the history from the previous overlapping time window.

This section introduces Al_5 by an example, and then provides a formal definition. It also presents the examples of using Al_5. This section ends with a discussion on the pros and cons of using Al_5.

5.2.1 Motivation for Al_5

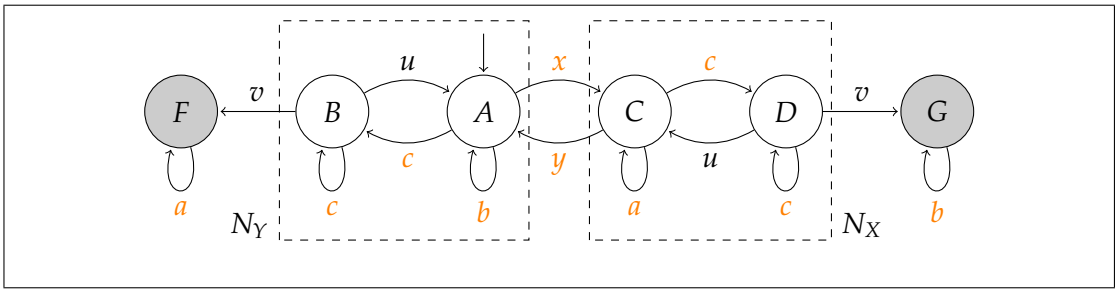

Figure 5.1: A simple DES model M_4 where a, b, c, x, y are observable events, and u, v are unobservable events

The information propagated by TWAs throughout the time windows relies on the notion of *abstract state* of a system. TWAs are useful for a real-world application once the abstract states are defined in a DES model. To illustrate abstract states, Fig. 5.1 shows a simple DES model M_4. N_X and N_Y are such abstract states. The abstract state of the system is N_Y if the current state is within $\{A, B\}$. The abstract state is N_X if the current state is within $\{C, D\}$. Note that these abstract states have been chosen

so that there are discriminable. Right after the observation of x (resp. y), the system is definitely in N_X (resp. N_Y)). Moreover, in this example, if the system is in N_X, 'b' is a symptom of a fault but not 'a'. If the system is in N_Y, 'a' is a symptom of a fault but not 'b'. Faults can thus be precisely diagnosed simply by keeping track of the current abstract state and checking for each 'a' and each 'b' whether they are not symptomatic from this current abstract state.

On the one hand, none of the four IWAs is precise for this example. This is because IWAs only look at bounded groups of consecutive observations while the distance between the last abstract state change and the first observation showing the failure may be arbitrarily large. For example, given an observation sequence $x, c, c, c, c, c, c, c, c, c, b$, and the time window size k is 4, the observations between the last abstract state change ('x') and the first observation showing the failure ('b') are all 'c' events. As a result, 'x' and 'b' are not guaranteed to be in the same time window.

On the other hand, it is unnecessary to remember precisely the current state of the system. The only relevant piece of information about the current state of the system is the *abstract state* of the system. This is precisely what the two proposed TWAs do. Firstly, a rough partition of the state space is assumed. How this partition is chosen is the topic of Section 5.5. The information that is passed from one time window to the next is the subset of abstract states that the current system state is belong to.

5.2.2 Definition of Al_5

Recall that Section 3.1.1 of Chapter 3 denotes $Q_0, Q_1, Q_2, \ldots, Q_p \subseteq Q$ as a collection of $(p+1)$ subsets of states. Al_5 relies on a partition $\Pi = \{Q_1, Q_2, Q_3, \ldots, Q_z, Q_F\}$ of the state-space Q where $z \in \mathbf{Z}, z \geq 1$, and Q_F denotes the faulty states. Let $\mathbb{L} = \{N_1, N_2, N_3, \ldots, N_z, F\}$ be a set of labels such that any N_i represents the nominal states Q_i, and F represents the faulty states Q_F. *Abstract state* is formally defined as follows.

Definition 32 (Abstract State) *An* abstract state *is an element Q_i of the state-space partition Π, and is represented by its corresponding label of $\mathbb{L}$.*

Similar to IWA Al_1, Al_5 slices a sequence of observations every k observations. The difference is that Al_5 also remembers some information from the previous time window. This is done by computing an *abstract belief state*, which is formally defined as follows.

Definition 33 (Abstract Belief State (ABS)) *An* abstract belief state *(ABS) is any non-empty subset of* $\mathbb{L}$.

Notice that Al_1 is a special case of Al_5. First, Al_1 only has one *label* of nominal state while Al_5 has several *labels* of nominal states, i.e. $\mathbb{L}_1 = \{N, F\}; \mathbb{L}_5 = \{N_1, N_2, N_3, \ldots, N_z, F\}$. Second, Al_1 diagnoses k observations as one time window, and moves to the next consecutive time window without keeping any information. For Al_5, diagnosis on one time window considers the abstract knowledge of the system states from the previous time window. The only exception is that the diagnosis for first time window begins with the initial state.

Recall that Definition 6 defines the *ct* predicate, and Definition 1 defines L^{-1} in Chapter 3. This work formally defines Al_5 as follows.

Definition 34 (Al_5) *Given a model M, the initial states I, a state abstraction $\mathbb{L}$, a time window size k, a sequence of observations $\Theta = \theta_1, \theta_2, \theta_3, \ldots, \theta_t$, the number of time windows $t = \frac{|\Theta|}{k}$, and $T_{Al_5} = \{[1,k], [k+1, 2k], [2k+1, 3k], \ldots, [|\Theta| - k + 1, |\Theta|]\}$,*

$$Al_5(M, \Theta, T_{Al_5}) = \begin{cases} N & \text{if } \exists \ell_1, \ell_2, \ldots, \ell_t \in \mathbb{L} \setminus \{F\} \text{ such that} \\ & ct([I, L^{-1}(\ell_1)], [\theta_1]) \wedge \\ & ct([L^{-1}(\ell_1), L^{-1}(\ell_2)], [\theta_2]) \wedge \cdots \wedge \\ & ct([L^{-1}(\ell_{t-1}), L^{-1}(\ell_t)], [\theta_t]) \\ F & \text{otherwise.} \end{cases}$$

Definition 34 says that the observation sequence Θ is sliced into t time windows such that each time window contains a sub-sequence of observations θ_i. These traces must have the same label at the beginning and the end of the time windows. Each trace provides an explanation for one time window. Al_5 returns nominal if for each time window, there exists a trace that is consistent with the sub-sequence of observations, and ends in a nominal state. Otherwise, it returns faulty.

Fig. 5.2 is a visual representation for the beginning part of Al_5. I represents the abstract state at the beginning of the first time window, to which the initial states belong, and $L^{-1}(\ell_1)$ represents the abstract state at the end of the first time window. $\theta_1 = obs(\sigma_1)$ is the observations of the trace σ_1 in the first time window, and $\theta_2 = obs(\sigma_2)$ is the observations of the trace σ_2 in the second time window. $[I, L^{-1}(\ell_1)]$ and $[\theta_1]$ represents the diagnosis on the first time window. $[L^{-1}(\ell_1), L^{-1}(\ell_2)]$ and $[\theta_2]$ represents the diagnosis on the second time window. In general, $q_1 \neq q_1'$; however, $L(q_1) = L(q_1')$.

This work studies the correctness and monotonicity properties of Al_5 as a diagnostic algorithm. Recall that Chapter 3 defines the two properties for a diagnostic algorithm in Definition 9.

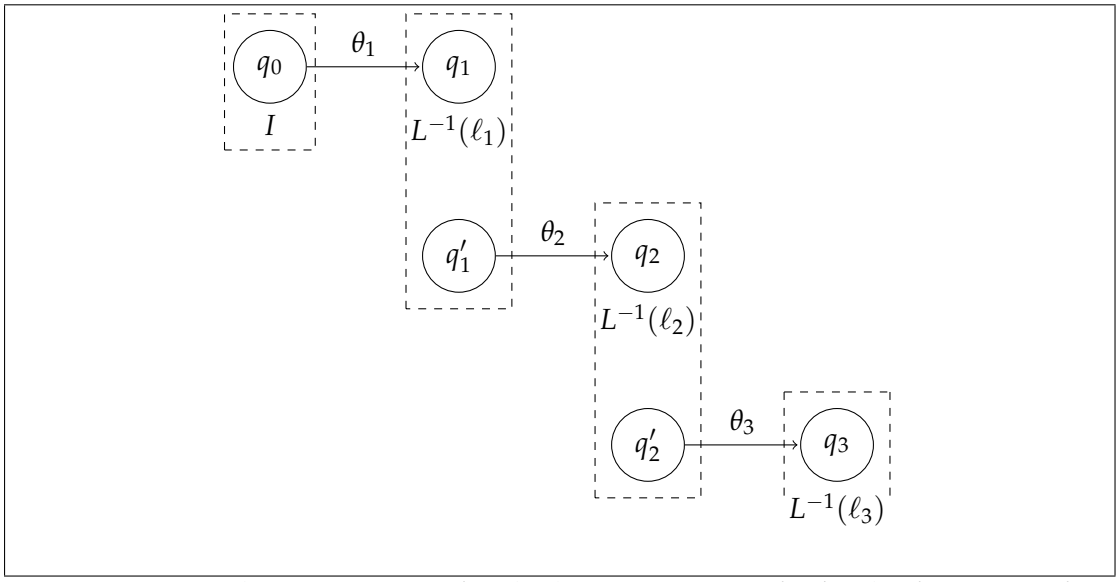

Figure 5.2: Visual representation for the beginning part of Al_5: the first row refers to the first time window; the second row and the third row refer to the following consecutive time window. $L^{-1}(\ell_3)$ will link to the fourth time window.

Property 4 *Al_5 is correct.*

Proof of Property 4 Given a model M, a sequence of observations Θ, and the definitions for a *consistent trace* and the *ct predicate*, Al_5 is correct $\Leftrightarrow$ $(\Delta(M,\Theta) = N \Rightarrow Al_5(M,\Theta,T_{Al_5}) = N)$.

$$\Delta(M,\Theta) = N \Rightarrow ct([I, L^{-1}(N)], [\Theta]) \text{ by Definition 7} \tag{5.1}$$

$$\Rightarrow \exists \text{ a trace } t : q_0 \xrightarrow{\sigma_1^{t_1}} q_1 \xrightarrow{\sigma_2^{t_1}} \dots \xrightarrow{\sigma_p^{t_m}} q_p \text{ such that } q_0 \in I \tag{5.2}$$

$$\Rightarrow \exists \text{ a trace } t_1 : q_0 \xrightarrow{\sigma_1^{t_1}} \dots \xrightarrow{\sigma_p^{t_1}} q_{k_1} \wedge \theta_1 = obs(\sigma_1^{t_1} \dots \sigma_p^{t_1}) \wedge \tag{5.3}$$

$$\exists \text{ a trace } t_2 : q_{k'_1} \xrightarrow{\sigma_1^{t_2}} \dots \xrightarrow{\sigma_p^{t_2}} q_{k_2} \wedge \theta_2 = obs(\sigma_1^{t_2} \dots \sigma_p^{t_2}) \wedge \tag{5.4}$$

$$\dots \wedge \tag{5.5}$$

$$\exists \text{ a trace } t_m : q_{k'_{t-1}} \xrightarrow{\sigma_1^{t_m}} \dots \xrightarrow{\sigma_p^{t_m}} q_{k_m} \wedge \theta_t = obs(\sigma_1^{t_m} \dots \sigma_p^{t_m}) \wedge \tag{5.6}$$

$$L(q_{k_1}) = L(q_{k'_1}) = \ell_1 \wedge \ell_1 \in \mathbb{L} \setminus \{F\} \wedge \dots \tag{5.7}$$

$$\Rightarrow ct([I, L^{-1}(\ell_1)], [\theta_1]) \wedge ct([L^{-1}(\ell_1), L^{-1}(\ell_2)], [\theta_2]) \wedge \dots \tag{5.8}$$

$$\wedge\, ct([L^{-1}(\ell_{m-2}), L^{-1}(\ell_{m-1})], [\theta_t]) \tag{5.9}$$

$$\Rightarrow Al_5(M,\Theta,T_{Al_5}) = N. \tag{5.10}$$

□

Property 5 *Al_5 is monotonic.*

Proof of Property 5 Al_5 is monotonic
$\Leftrightarrow (Al_5(M, \Theta, T_{Al_5}) = F \Rightarrow Al_5(M, \Theta\ \theta', T'_{Al_5}) = F)$, where
$\theta' = e_1e_2e_3 \ldots e_k, e_i \in \Sigma_o, i \in [1,k], i \in \mathbf{Z}, T'_{Al_5} = T_{Al_5} \cup \{[|\Theta|+1, |\Theta|+k]\}$
$\Leftrightarrow (\forall e \in \Sigma_o,\ Al_5(M, \Theta\ \theta', T'_{Al_5}) = N \Rightarrow Al_5(M, \Theta, T_{Al_5}) = N)$.

$$\forall e \in \Sigma_o,\ Al_5(M, \theta_1\theta_2\theta_3 \ldots \theta_t\ \theta', T'_{Al_5}) = N \tag{5.11}$$

$$\Rightarrow \exists \ell_1, \ell_2, \ell_3, \ldots, \ell_t, \ell_{t+1} \in \mathbb{L} \setminus \{F\} \text{ such that} \tag{5.12}$$

$$\begin{aligned} ct([I, L^{-1}(\ell_1)], &\ [\theta_1]) \wedge \\ ct([L^{-1}(\ell_1), L^{-1}(\ell_2)], &\ [\theta_2]) \wedge \cdots \wedge \\ ct([L^{-1}(\ell_{t-1}), L^{-1}(\ell_t)], &\ [\theta_t]) \wedge \\ ct([L^{-1}(\ell_t), L^{-1}(\ell_{t+1})], &\ [\theta']) \text{ by Definition 34} \end{aligned} \tag{5.13}$$

$$\Rightarrow \exists \ell_1, \ell_2, \ell_3, \ldots, \ell_t, \ell_{t+1} \in \mathbb{L} \setminus \{F\} \text{ such that} \tag{5.14}$$

$$\begin{aligned} ct([I, L^{-1}(\ell_1)], &\ [\theta_1]) \wedge \\ ct([L^{-1}(\ell_1), L^{-1}(\ell_2)], &\ [\theta_2]) \wedge \cdots \wedge \\ ct([L^{-1}(\ell_{t-1}), L^{-1}(\ell_t)], &\ [\theta_t]) \end{aligned} \tag{5.15}$$

$$\Rightarrow Al_5(M, \theta_1\theta_2\theta_3 \ldots \theta_t, T_{Al_5}) = N \tag{5.16}$$

$$\Rightarrow Al_5(M, \Theta, T_{Al_5}) = N \tag{5.17}$$

□

5.2.3 Example of Al_5

Algorithm	Slice	ABS	Diagnosis
Δ	Θ	NA	F (precise)
Al_1	bx	$\{N\}$	N (imprecise)
	cc	$\{N\}$	
	cb	$\{N\}$	
Al_5	bx	$\{N_X\}$	F (precise)
	cc	$\{N_X\}$	
	cb	$\emptyset$	

Table 5.1: Diagnostic results of Δ, Al_1 and Al_5 for M_1, and observations $\Theta = bxcccb$: ABS refers to the abstract belief state at the end of each time window.

Tab. 5.1 shows the diagnostic results of observations $\Theta = bxcccb$ when k is 2 for M_4 in Fig. 5.1. The abstract state labels of Al_5 are $\mathbb{L} = \{N_X, N_Y, F\}$. For any time window θ_i, Al_1 concludes N since there always exists a nominal trace σ_i starting from a nominal state of the system such that $obs(\sigma_i) = \theta_i$. In contrast, Al_5 is able to diagnose this sequence precisely.

- For the first time window bx, there exists a trace that begins with the initial state, and is compatible with this time window, i.e. $A \xrightarrow{b} A \xrightarrow{x} C$. In particular, $L(C)$ evaluates to N. The ABS at the end of this time window is N_X. Thus, the first time window leads to N_X, and the next time window should continue from any state with the abstract state of N_X.

- For the second time window cc, there exists a trace that begins with a state in N_X, i.e. the state C, and is compatible with this time window, i.e. $C \xrightarrow{c} D \xrightarrow{c} D$. In particular, $L(D)$ evaluates to N. The ABS at the end this time window is N_X. Thus, the second time window leads to N_X, and the next time window should continue from any state with the abstract state of N_X.

- For the third time window cb, there exists a trace that begins with a state in N_X, i.e. the state D, and is compatible with this time window, i.e. $D \xrightarrow{c} D \xrightarrow{v} G \xrightarrow{b} G$. In particular, $L(G)$ evaluates to F. The ABS at the end of this time window is $\emptyset$ as there is no nominal explanation, by Definition 34. Thus, the third time window leads to F.

Therefore, the diagnostic output of Al_5 is F.

5.2.4 Discussions of Al_5

Al_5 is more precise than Al_1, but this comes at a price. Al_5 does not take the advantage of parallel computation, and must analyse time windows in order. In contrast, Al_1 is able to diagnose multiple time windows independently and simultaneously.

Furthermore, Al_5 is more vulnerable to the *masking* issue than Al_1. Observations are *masked* when the communication layer used to transmit the observations is momentarily faulty. Even if the complete sequence of observations is not available, Al_1 resets the diagnosis at the beginning of every time window, and therefore Al_1 is immune to the masking issue. In comparison, when certain observations are masked and a fault actually occurs, Al_5 relies on the assumption that the abstract state can be estimated with sufficient precision, but the masking wipes out this estimation.

5.3 Time-Window Algorithm (TWA) Al_6

This section introduces Al_6 by an example, and then provides a formal definition. It also presents examples of using Al_6. This section ends with a discussion on the pros and cons of using Al_6.

5.3.1 Motivation for Al_6

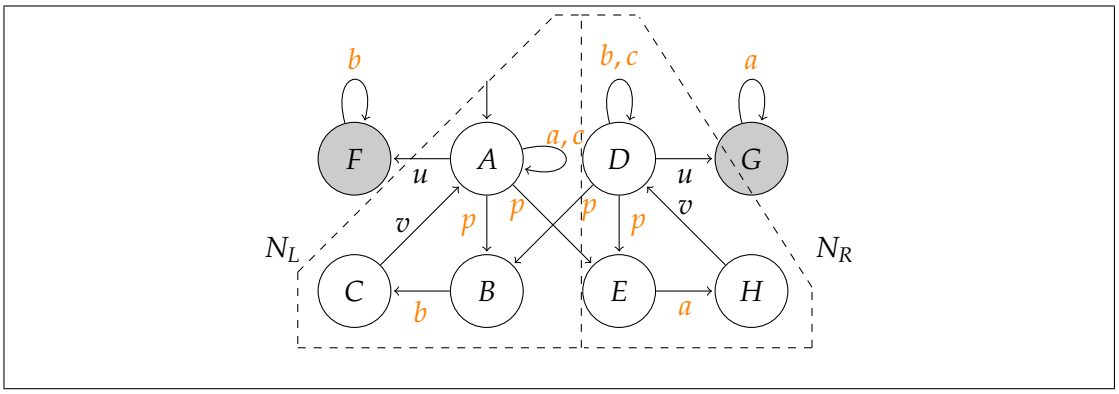

Figure 5.3: DES model M_5 with a set of abstract states $\{N_L, N_R\}$ where Al_5 is not precise. a, b, c, p are observable, and u, v are unobservable.

This section presents a more sophisticated example that motivates Al_6. Fig. 5.3 shows a DES model M_5. a, b, c, p are observable, and u, v are unobservable. Suppose that the selected partition splits the nominal states into two abstract states, i.e. $N_L = \{A, B, C\}$ and $N_R = \{D, E, H\}$. In this setting, a consecutive sequence of b's (resp. a's) means the system is faulty if the system is known to be in N_L (resp. N_R).

Diagnoser	Slice	ABS	Diagnosis
Δ	Θ	NA	F (precise)
Al_5	ap	$\{N_L, N_R\}$	N (imprecise)
	aa	$\{N_L\}$	

Table 5.2: Diagnostic results of Δ and Al_5 for M_5 in Fig. 5.3, and observations $\Theta = apaa$: ABS refers to the abstract belief state at the end of each time window.

Tab. 5.2 shows the diagnostic results of Al_5 when k is 2. If a sequence of observations $\Theta = apaa$ is sliced to ap and aa, the abstract belief state immediately after ap should be N_R. However, Al_5 is unable to precisely determine the abstract belief state after ap, which makes it unable to diagnose the fault. This is because this model requires two observable events to recognise the transition to a different abstract state, i.e. pa means transition to N_R and pb means transition to N_L. Regardless how large

value of k, if one time window ends after p, i.e. the next time window will start after p, then the abstract belief state cannot be precisely determined. Therefore, Al_5 is not precise w.r.t. M_5.

5.3.2 Definition of Al_6

Compared to Al_5, Al_6 slices a sequence of observations every k observations, and includes additional overlapping time windows. Assume k is a multiple of 2 for simplicity, and let h be $\frac{k}{2}$. Al_6 refines the carry-over information of any time window i (denoted as $@i$), and passes it to the next overlapping time window $i+1$ (denoted as $@i+1$) so that inconsistent abstract states will be eliminated.

Definition 35 (Al_6) *Given a model M, the initial states I, a state abstraction $\mathbb{L}$, a time windows size k with $h = \frac{k}{2}$, a sequence of observations Θ, the number of time windows $t = \frac{|\Theta|}{h} - 1$, and $T_{Al_6} = w\_6(1), w\_6(2), w\_6(3), \ldots, w\_6(t)$, $Al_6(M, \Theta, T_{Al_6})$ returns N if*

$$
\begin{aligned}
&\exists \ell_0@i, \ell_1@i, \ell_2@i \in \mathbb{L} \setminus \{F\}\ \forall i \in \{1, 2, 3, \ldots, t\} \text{ such that} \\
&\quad \ell_1@1 = \ell_0@2 \wedge \cdots \wedge \ell_1@(t-1) = \ell_0@t \\
&\wedge\ ct([I, L^{-1}(\ell_1@1), L^{-1}(\ell_2@1)], [w\_6(1)]) \\
&\wedge\ ct([L^{-1}(\ell_0@2), L^{-1}(\ell_1@2), L^{-1}(\ell_2@2)], [w\_6(2)]) \\
&\wedge\ \ldots \\
&\wedge\ ct([L^{-1}(\ell_0@t), L^{-1}(\ell_1@t), L^{-1}(\ell_2@t)], [w\_6(t))])
\end{aligned}
$$

where $w\_6(i) = (\Theta[H+1, H+h], \Theta[H+(h+1), H+2h])$, and $H = (i-1) \times h$; $Al_6(M, \Theta, T_{Al_6})$ returns F otherwise.

The function w_6 returns the i-th time window, which consists of two equal-length sub-sequences of observations θ_1 and θ_2. Al_6 returns nominal if for each time window, there exists a trace that is consistent with the sub-sequence of observations, and ends in a nominal state. Otherwise, it returns faulty.

Fig. 5.4 is a visual representation for the beginning part of Al_6. I represents the abstract state at the beginning of the first time window, to which the initial states belong. The diagnosis of the window j searches a trace that starts from a state of $L^{-1}(\ell_0@j)$, goes through a state of $L^{-1}(\ell_1@j)$ while generating the first half of the observations, and ends in a state of $L^{-1}(\ell_2@j)$ while generating the second half. The rationale behind Al_6 is that the estimation of $\ell_2@j$ is not very precise because it is supported only by previous observations, whilst that of $\ell_1@j$ is supported by both previous and latter observations. Therefore, the information that is carried over to the next window is $\ell_1@j = \ell_0@(j+1)$.

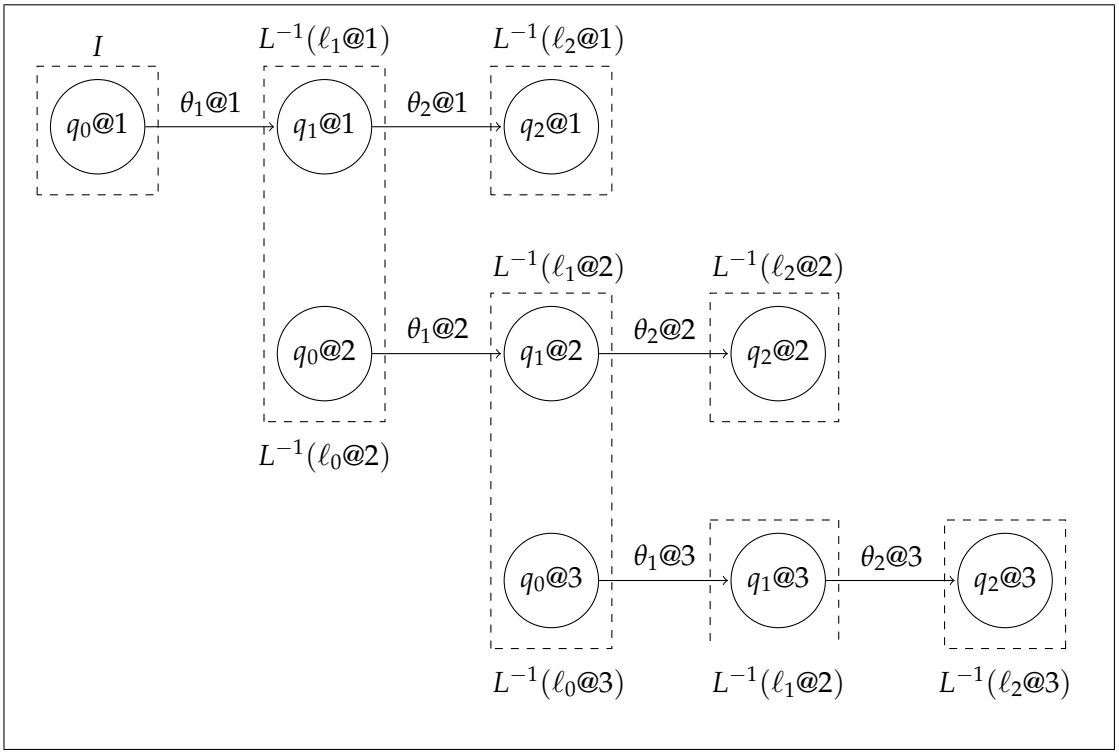

Figure 5.4: Visual representation for the beginning part of Al_6: the first row refers to the first time window; the second row and the third row refer to the following overlapping time window. $L^{-1}(\ell_1@3)$ will link to the fourth time window.

This work studies the correctness and monotonicity properties of Al_6 as a diagnostic algorithm. Recall that Chapter 3 defines the two properties for a diagnostic algorithm in Definition 9.

Property 6 *Al_6 is correct.*

Proof of Property 6 Given a model M, a sequence of observations Θ, and the definitions for a *consistent trace* and the *ct predicate*, Al_6 is correct $\Leftrightarrow (\Delta(M,\Theta) = N \Rightarrow Al_6(M,\Theta,T_{Al_6}) = N)$.

$$\text{Let } \mathbb{N} = \mathbb{L} \setminus \{F\}. \tag{5.18}$$

$$\text{Given } \Delta(M, \Theta) = N \tag{5.19}$$

$$\Rightarrow ct([I, L^{-1}(N)], [\Theta]) \text{ by Definition 7} \tag{5.20}$$

$$\Rightarrow \exists \text{ a trace } t : q_0 \xrightarrow{\sigma_1^{t_1}} q_1 \xrightarrow{\sigma_2^{t_1}} \ldots \xrightarrow{\sigma_p^{t_t}} q_p \text{ such that } q_0 \in I \tag{5.21}$$

$$\Rightarrow \quad \exists \text{ a trace } t_1 : q_0 \xrightarrow{\sigma_1^{t_1}} \ldots \xrightarrow{\sigma_g^{t_1}} q_{h_1} \wedge \theta_1 = obs(\sigma_1^{t_1} \ldots \sigma_g^{t_1}) \tag{5.22}$$

$$\wedge \exists \text{ a trace } t_2 : q_{h_1} \xrightarrow{\sigma_1^{t_2}} \ldots \xrightarrow{\sigma_g^{t_2}} q_{k_1} \wedge \theta_2 = obs(\sigma_1^{t_2} \ldots \sigma_g^{t_2}) \tag{5.23}$$

$$\wedge \exists \text{ a trace } t_3 : q_{h'_1} \xrightarrow{\sigma_1^{t_3}} \ldots \xrightarrow{\sigma_g^{t_3}} q_{k'_1} \wedge \theta_3 = obs(\sigma_1^{t_3} \ldots \sigma_g^{t_3}) \tag{5.24}$$

$$\wedge \exists \text{ a trace } t_4 : q_{k'_1} \xrightarrow{\sigma_1^{t_4}} \ldots \xrightarrow{\sigma_g^{t_4}} q_{h_2} \wedge \theta_4 = obs(\sigma_1^{t_4} \ldots \sigma_g^{t_4}) \tag{5.25}$$

$$\wedge \ldots \tag{5.26}$$

$$\wedge \exists \text{ a trace } t_{t-1} : q_{h'_{t-1}} \xrightarrow{\sigma_1^{t_{t-1}}} \ldots \xrightarrow{\sigma_g^{t_{t-1}}} q_{k'_{t-1}} \wedge \theta_{t-1} = obs(\sigma_1^{t_{t-1}} \ldots \sigma_g^{t_{t-1}}) \tag{5.27}$$

$$\wedge \exists \text{ a trace } t_t : q_{k'_{t-1}} \xrightarrow{\sigma_1^{t}} \ldots \xrightarrow{\sigma_g^{t}} q_{k_t} \wedge \theta_t = obs(\sigma_1^{t_t} \ldots \sigma_g^{t_t}) \tag{5.28}$$

$$\wedge L(q_0) = \ell_0@1 \wedge \ell_0@1 \in \mathbb{N} \tag{5.29}$$

$$\wedge L(q_{k_1}) = \ell_1@2 \wedge \ell_1@2 \in \mathbb{N} \tag{5.30}$$

$$\wedge L(q_{h_1}) = L(q_{h'_1}) = \ell_1@1 \wedge \ell_1@1 \in \mathbb{N} \tag{5.31}$$

$$\wedge \ldots \tag{5.32}$$

$$\Rightarrow \quad ct([I, L^{-1}(\ell_1@1), L^{-1}(\ell_2@1)], [w\_6(1)]) \tag{5.33}$$

$$\wedge ct([L^{-1}(\ell_0@2), L^{-1}(\ell_1@2), L^{-1}(\ell_2@2)], [w\_6(2)]) \tag{5.34}$$

$$\wedge \ldots \tag{5.35}$$

$$\wedge ct([L^{-1}(\ell_0@t), L^{-1}(\ell_1@t), L^{-1}(\ell_2@t)], [w\_6(t)]) \tag{5.36}$$

$$\wedge w\_6(1) = \theta_1\theta_2 \tag{5.37}$$

$$\wedge w\_6(2) = \theta_2\theta_3 \tag{5.38}$$

$$\wedge \ldots \tag{5.39}$$

$$\wedge w\_6(t) = \theta_{t-1}\theta_t \tag{5.40}$$

$$\Rightarrow Al_6(M, \Theta, T_{Al_6}) = N. \tag{5.41}$$

□

Property 7 *Al_6 is monotonic.*

Proof of Property 7 Al_6 is monotonic
$\Leftrightarrow (Al_6(M,\Theta,T_{Al_6}) = F \Rightarrow Al_6(M,\Theta\ \theta',T'_{Al_6}) = F)$, where
$\theta' = e_1e_2e_3\ldots e_{\frac{k}{2}}, e_i \in \Sigma_o, i \in [1,\frac{k}{2}], i \in \mathbf{Z}, T'_{Al_6} = T_{Al_6} \cup \{w_6(t+1)\}$
$\Leftrightarrow (Al_6(M,\Theta\ \theta',T'_{Al_6}) = N \Rightarrow Al_6(M,\Theta,T_{Al_6}) = N)$.

$$Al_6(M,\theta_1\theta_2\theta_3\ldots\theta_t\ \theta',T'_{Al_6}) = N \tag{5.42}$$

$$\Rightarrow \exists \ell_0@i,\ell_1@i,\ell_2@i \in \mathbb{N},\ \forall i \in \{1,2,3,\ldots,t,t+1\} \text{ such that} \tag{5.43}$$

$$\begin{aligned} & ct([I,L^{-1}(\ell_1@1),L^{-1}(\ell_2@1)], \quad [w\_6(1)]) \\ \wedge\ & ct([L^{-1}(\ell_0@2),L^{-1}(\ell_1@2),L^{-1}(\ell_2@2)], \quad [w\_6(2)]) \\ \wedge\ & \ldots \\ \wedge\ & ct([L^{-1}(\ell_0@t),L^{-1}(\ell_1@t),L^{-1}(\ell_2@t)], \quad [w\_6(t)]) \\ \wedge\ & ct([L^{-1}(\ell_0@(t+1)),L^{-1}(\ell_1@(t+1)),L^{-1}(\ell_2@(t+1))], \quad [w\_6(t+1)]) \end{aligned} \tag{5.44}$$

$$\wedge\ \ell_1@1 = \ell_0@2 \wedge \cdots \wedge \ell_1@t = \ell_0@(t+1) \text{ by Definition 35} \tag{5.45}$$

$$\Rightarrow \ell_0@i,\ell_1@i,\ell_2@i \in \mathbb{N},\ \forall i \in \{1,2,3,\ldots,t,t+1\} \text{ such that} \tag{5.46}$$

$$\begin{aligned} & ct([I,L^{-1}(\ell_1@1),L^{-1}(\ell_2@1)], \quad [w\_6(1)]) \\ \wedge\ & ct([L^{-1}(\ell_0@2),L^{-1}(\ell_1@2),L^{-1}(\ell_2@2)], \quad [w\_6(2)]) \\ \wedge\ & \ldots \\ \wedge\ & ct([L^{-1}(\ell_0@t),L^{-1}(\ell_1@t),L^{-1}(\ell_2@t)], \quad [w\_6(t))]) \end{aligned} \tag{5.47}$$

$$\wedge\ \ell_1@1 = \ell_0@2 \wedge \cdots \wedge \ell_1@(t-1) = \ell_0@t \tag{5.48}$$

$$\Rightarrow Al_6(M,\theta_1\theta_2\theta_3\ldots\theta_t,T_{Al_6}) = N \tag{5.49}$$

$$\Rightarrow Al_6(M,\Theta,T_{Al_6}) = N \tag{5.50}$$

□

5.3.3 Example of Al_6

Diagnoser	Slice	ABS-End	ABS-Middle	CT	Output
Δ	Θ	NA	NA	NA	F
Al_6	ap	$\{N_L,N_R\}$	$\{N_L\}$	Yes	F
	pa	$\{N_R\}$	$\{N_R\}$	Yes	
	aa	NA	$\emptyset$	No	
	Algorithm may stop here.				

Table 5.3: Diagnostic results of Δ and Al_6 for M_5 in Fig. 5.3 and the same observation as Tab. 5.2, i.e. $\Theta = apaa$: ABS means the abstract belief state in the middle of a time window; CT means whether a consistent trace holds.

Tab. 5.3 shows the diagnostic results when k is 2. Al_6 returns the precise diagnosis using the given abstract states as stated in Section 5.3.1. Al_6 is more precise than Al_5, without requiring a more detailed break-down of abstract states. This is because Al_6 refines the diagnosis on the slices. For instance, at the end of the first slice (a, p), the system may be in the state B or E. After the second slice (p, a), Al_6 eliminates the state B because the observation a is impossible from this state. The diagnostic result is more precise because it has been refined to be in the state E only.

- For the first time window ap, there are two traces that begin with the initial state, and are compatible with this time window, i.e. $A \xrightarrow{a} A \xrightarrow{p} B$, and $A \xrightarrow{a} A \xrightarrow{p} E$. The ABS at the end of this time window is $\{N_L, N_R\}$ since B is in N_L and E is in N_R. However, the ABS in the middle of this time window is $\{N_L\}$ since A is in N_L . Thus, a consistent trace holds, and the next time window should continue from a state with the abstract state of N_L.

- For the second time window pa, there exists a trace that begins with a state in N_L, and is compatible with this time window, i.e. $A \xrightarrow{p} E \xrightarrow{a} H$. The ABS at the end of this time window is $\{N_R\}$ since H is in N_R. Also, the ABS in the middle of this time window is $\{N_R\}$ since E is in N_R. Thus, a consistent trace holds, and the next time window should continue from a state with the abstract state of N_R.

- For the third time window aa, there exists a trace that begins with a state in N_R, and is compatible with this time window, i.e. $E \xrightarrow{a} H \xrightarrow{v} D \xrightarrow{u} G \xrightarrow{a} G$. The ABS at the end of this time window is $\emptyset$, and there is no consistent trace, since G is faulty.

Therefore, the diagnostic result is F.

5.4 Precision Verification for TWAs

Even though the diagnosability of a system holds, the two TWAs presented in Section 5.2 and 5.3 is not guaranteed to detect the faults. For example, their precision w.r.t. a model depends on the selections of abstract states. It is therefore important to be able to assess whether the imprecision introduced by the algorithms makes some faulty behaviours undetectable. Chapter 3 defines that a diagnostic algorithm A is precise w.r.t. a system model M if it always detects and identifies faults. It also shows that the precision verification is implemented using a *simulation*, which is a finite state machine that models how a diagnostic algorithm works on a DES model. Formally, the simulation is a model $si(M, A)$ such that diagnosing the observations

with model M and algorithm A yields the same results as using Δ diagnosis and model $si(M, A)$, i.e.

$$\forall\Theta,\ \Delta(si(M, A), \Theta) = A(M, \Theta).$$

This section demonstrates the constructions of the simulations for Al_5 and Al_6. Then, verifying whether a diagnostic algorithm A is precise consists in analysing $si(M, A)$ using the twin plant method as discussed in Section 2.2.2 of Chapter 2.

5.4.1 Precision of Al_5 w.r.t. a DES Model

The simulation for a DES model M and Al_5 is defined as follows.

Definition 36 (Al_5-Simulation) *The Al_5-simulation for a model $M = \langle Q, \Sigma, T, I, L\rangle$ is the automaton $M_5 = \langle Q_5, \Sigma_5, T_5, I_5, L_5\rangle$ where*

- $Q_5 = Q \times \{0, 1, 2, \ldots, k\}$;
- $\Sigma_5 = \Sigma \cup \{\varepsilon\}$;
- $T_5 = T_u \cup T_o \cup T_\varepsilon$ *is defined by:*
  - $T_u = \{\langle\langle q, i\rangle, u, \langle q', i\rangle\rangle \mid i \in \{0, 1, 2, \ldots, k\} \wedge \langle q, u, q'\rangle \in T \wedge u \in \Sigma \setminus \Sigma_o\}$,
  - $T_o = \{\langle\langle q, i\rangle, o, \langle q', i+1\rangle\rangle \mid i \in \{0, 1, 2, \ldots, k-1\} \wedge \langle q, o, q'\rangle \in T \wedge o \in \Sigma_o\}$, *and*
  - $T_\varepsilon = \{\langle\langle q, k\rangle, \varepsilon, \langle q', 0\rangle\rangle \mid L_5(q) = L_5(q')\}$.
- $I_5 = I \times \{0\}$; *and,*
- $L_5 : Q_5 \to \mathbb{L}$, $L_5(\langle q, i\rangle) = L(q)$.

Example of Simulation for M_4 and Al_5 Fig. 5.5 shows the simulation for the DES model M_4 in Fig. 5.1 and Al_5 with the window size k being 2. Each state of the simulation is associated with a counter, which simulates the number of observations made in the current time window. For instance, the state A_1 represents the situation where the current system state is A, and 1 observation was made so far in the current time window. When the counter reaches 2, the end of the time window is simulated by an unobservable ε-transition that obliterates the current state and only remembers the abstract state, i.e. *reset*. For instance, there is an ε-transition from A_2 to B_0 because A and B share the same label N_Y. On the other hand, there is no transition from A_2 to C_0 since A and C have different labels.

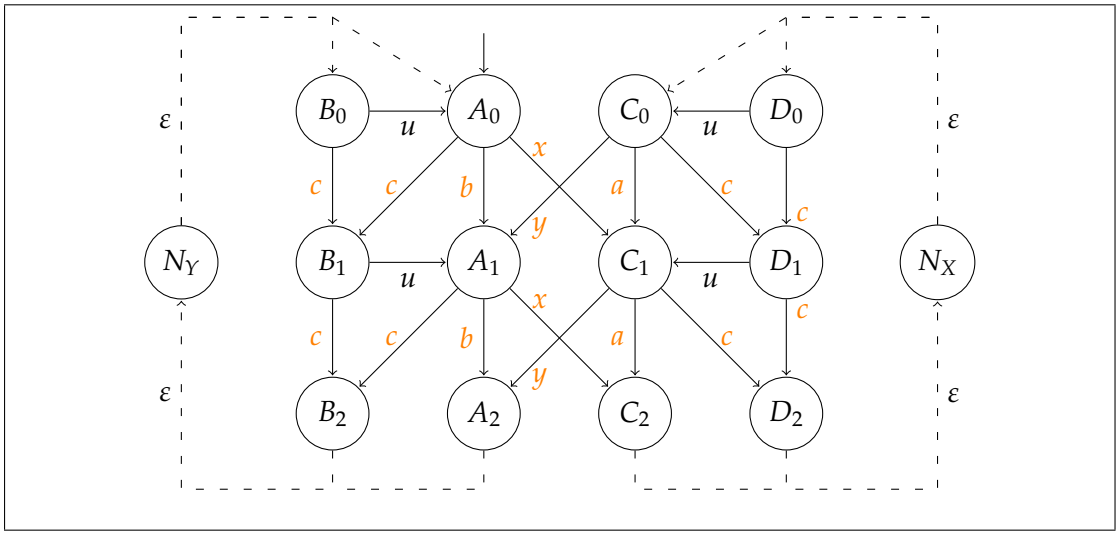

Figure 5.5: Simulation for M_4 in Fig. 5.1 and Al_5: N_Y and N_X are only used to highlight the abstract states. The faulty states are omitted because Assumption 4 of Chapter 3 states that the diagnostic policy of this work does not distinguish between the nominal mode and ambiguous mode, i.e. a diagnoser only looks for nominal explanations. Therefore, the faulty states are not included in the simulation.

To verify the precision of Al_5 w.r.t. M_4, the simulation for Al_5 needs to be synchronised with M_4. The twin plant is then built for this synchronisation. Any ambiguous path is checked on the twin plant. There is no ambiguous path. Therefore, Al_5 is precise w.r.t. M_4 when k is 2 by Theorem 1 of Chapter 3.

□

This work examines the difference between Al_5-simulation and Al_1-simulation. Section 4.3.1 of Chapter 4 defines Al_1-simulation. If a time window ends in a nominal state, Al_1-simulation resets to every nominal state. If a time window ends in a faulty state, Al_1-simulation resets to every faulty state. In comparison, Al_5-simulation only resets to those states that belong to the same abstract state.

Theorem 10 proves the correctness of the simulation for a DES model M and Al_5.

Theorem 10 (Simulation for a DES Model M and Al_5) *The Al_5-simulation as defined in Definition 36 returns a simulation for a system model M and Al_5 using the window size k.*

Proof of Theorem 10 The first part is to prove $\Delta(M_5, \Theta) = N \Rightarrow Al_5(M, \Theta, T_{Al_5}) = N$ as seen in the proof of Property 4.

The second part is to prove $Al_5(M, \Theta, T_{Al_5}) = N \Rightarrow \Delta(M_5, \Theta) = N$.

$$\text{Suppose } Al_5(M_5, \Theta, T_{Al_5}) = N, t = \frac{|\Theta|}{k}, \Theta = \theta_1\theta_2 \dots \theta_t \tag{5.51}$$

$$\Rightarrow \exists \ell_1, \ell_2, \ell_3, \dots, \ell_t \in \mathbb{N} \text{ such that} \tag{5.52}$$

$$\begin{aligned} ct([I, L^{-1}(\ell_1)], \quad [\theta_1]) \wedge \\ ct([L^{-1}(\ell_1), L^{-1}(\ell_2)], \quad [\theta_2]) \wedge \dots \wedge \\ ct([L^{-1}(\ell_{t-1}), L^{-1}(\ell_t)], \quad [\theta_t]) \end{aligned} \tag{5.53}$$

$$\text{holds by Definition 34} \tag{5.54}$$

$$\Rightarrow \ \exists \text{ a trace } t_1 : q_0 \xrightarrow{\sigma_1^{t_1}} \dots \xrightarrow{\sigma_k^{t_1}} q_{n_1} \text{ is consistent with } \theta_1 = obs(\sigma_1^{t_1} \dots \sigma_k^{t_1}) \tag{5.55}$$

$$\wedge \exists \text{ a trace } t_2 : q'_{n_1} \xrightarrow{\sigma_1^{t_2}} \dots \xrightarrow{\sigma_k^{t_2}} q_{n_2} \text{ is consistent with } \theta_2 = obs(\sigma_1^{t_2} \dots \sigma_k^{t_2}) \tag{5.56}$$

$$\wedge \dots \tag{5.57}$$

$$\Rightarrow \exists \text{ a trace } t : \langle q_0, 0\rangle \xrightarrow{\sigma_1^{t_1}} \dots \xrightarrow{\sigma_k^{t_1}} \langle q_{n_1}, k\rangle \xrightarrow{\varepsilon} \langle q'_{n_1}, 0\rangle \xrightarrow{\sigma_1^{t_2}} \dots \tag{5.58}$$

$$\Rightarrow ct([I, L^{-1}(N)], [\Theta]) \text{ holds} \tag{5.59}$$

$$\Rightarrow \Delta(M, \Theta) = N. \tag{5.60}$$

□

This work discusses the complexity of the Al_5-simulation. The size of the Al_5-simulation is k times that of the DES model, where k is the time window size. Thus, verifying the precision of Al_5 w.r.t. a DES model using the twin plant method remains polynomial. Remember that a simulation is only used to verify the precision of a diagnostic algorithm w.r.t. a DES model, and not for diagnosis.

5.4.2 Precision of Al_6 w.r.t. a DES Model

Before presenting the formal definition, this section explains how to build the Al_6-simulation. This work uses two copies of Q_5 defined in Section 5.4.1. The two copies are called Q_1 and Q_2. Q_1 represents one time window, and Q_2 represents the next overlapping time window. This work assumes that k is a multiple of 2 for simplicity. Let the step index of Q_1 be i, the step index of Q_2 be j, and the middle of a time window be $h = \frac{k}{2}$. This work specifies that the two time windows overlap as $i = j + h \pmod{k}$.

The initial state is that Q_1 beginning at the step 0, and Q_2 beginning at the middle of a time window h. Thus, Q_6 is the synchronization of two single-window algorithms such that two time windows overlap. The abstract belief state is refined from the end of one time window up to the middle of this time window. Then, T_ε

carries over the information between two overlapping time windows. The simulation for a DES model M and Al_6 is defined as follows.

Definition 37 (Al_6-Simulation) *The Al_6-simulation for $M = \langle Q, \Sigma, T, I, L\rangle$ is the automaton $M_6 = \langle Q_6, \Sigma_6, T_6, I_6, L_6\rangle$ where*

- $Q_6 = Q_1 \times Q_2$ *where* $Q_1 = Q \times \{0, k\}$, $Q_2 = Q \times \{0, k\}$, $\langle q_1, i\rangle$ *is an instance of* Q_1, *and* $\langle q_2, j\rangle$ *is an instance of* Q_2, *such that* $i = j + h \pmod k$;
- $\Sigma_6 = \Sigma \cup \{\varepsilon\}$;
- $T_6 \subseteq Q_6 \times \Sigma_6 \times Q_6$ *where* $T_6 = T_{u1} \cup T_{u2} \cup T_o \cup T_{\varepsilon 1} \cup T_{\varepsilon 2}$ *is defined by:*
  - $T_{u1} = \{\langle\langle q_1, i\rangle, \langle q_2, j\rangle\rangle, u, \langle\langle q_1', i'\rangle, \langle q_2', j'\rangle\rangle\rangle \mid i \in \{0, 1, 2, \ldots, k\} \wedge j \in \{0, 1, 2, \ldots, k\} \wedge \langle q_1, u, q_1'\rangle \in T \wedge u \in \Sigma \setminus \Sigma_o \wedge (q_2 = q_2') \wedge (i = i') \wedge (j = j')\}$,
  - $T_{u2} = \{\langle\langle q_1, i\rangle, \langle q_2, j\rangle\rangle, u, \langle\langle q_1', i'\rangle, \langle q_2', j'\rangle\rangle\rangle \mid i \in \{0, 1, 2, \ldots, k\} \wedge j \in \{0, 1, 2, \ldots, k\} \wedge \langle q_2, u, q_2'\rangle \in T \wedge u \in \Sigma \setminus \Sigma_o \wedge (q_1 = q_1') \wedge (i = i') \wedge (j = j')\}$,
  - $T_o = \{\langle\langle q_1, i\rangle, \langle q_2, j\rangle\rangle, o, \langle\langle q_1', i'\rangle, \langle q_2', j'\rangle\rangle\rangle \mid i \in \{0, 1, 2, \ldots, k-1\} \wedge j \in \{0, 1, 2, \ldots, k-1\} \wedge \langle q_1, o, q_1'\rangle \in T \wedge \langle q_2, o, q_2'\rangle \in T \wedge o \in \Sigma_o \wedge (i' = i + 1) \wedge (j' = j + 1)\}$,
  - $T_{\varepsilon 1} = \{\langle\langle q_1, h\rangle, \varepsilon, \langle q_2', 0\rangle\rangle \mid L(q_1) = L(q_2')\}$,
  - $T_{\varepsilon 2} = \{\langle\langle q_2, h\rangle, \varepsilon, \langle q_1', 0\rangle\rangle \mid L(q_2) = L(q_1')\}$;
- $I_6 = (I \times \{0\}) \times (I \times \{h\})$; *and,*
- $L_6 : Q_6 \to \mathbb{L}$, $L_6(\langle q_1, i\rangle, \langle q_2, j\rangle) = F$ *if* $L(q_1) = F$ *or* $L(q_2) = F$; *otherwise,* $L_6(\langle q_1, i\rangle, \langle q_2, j\rangle) = N$.

Theorem 11 proves the correctness of the simulation for a DES model M and Al_6.

Theorem 11 (Simulation for a DES model M and Al_6) *The Al_6-simulation defined in Definition 37 returns a simulation for a system model M and Al_6 using the window size k.*

Proof of Theorem 11 For simplicity, assume k is a multiple of 2, and the number of time windows $t = \frac{|\Theta|}{h} - 1$. The first part is to prove $\Delta(M_6, \Theta) = N \Rightarrow Al_6(M, \Theta, T_{Al_6}) = N$.

$$\text{Suppose } \Delta(M_6, \Theta) = N \tag{5.61}$$

$$\Rightarrow Al_6(M, \Theta, T_{Al_6}) = N \text{ by Property 6, i.e. the correctness of } Al_6. \tag{5.62}$$

The second part is to prove $Al_6(M, \Theta, T_{Al_6}) = N \Rightarrow \Delta(M_6, \Theta) = N$.

$$\text{Suppose } Al_6(M_6, \Theta, T_{Al_6}) = N, t = \frac{|\Theta|}{k}, \Theta = \theta_1\theta_2\theta_3 \dots \theta_t \tag{5.63}$$

$$\Rightarrow \ell_0@i, \ell_1@i, \ell_2@i \in \mathbb{N}\ \forall i \in \{1, 2, 3, \dots, t\} \text{ such that} \tag{5.64}$$

$$\begin{aligned} & ct([I, L^{-1}(\ell_1@1), L^{-1}(\ell_2@1)], \quad [w\_6(1)]) \\ \wedge\ & ct([L^{-1}(\ell_0@2), L^{-1}(\ell_1@2), L^{-1}(\ell_2@2)], \quad [w\_6(2)]) \\ & \qquad \wedge\ \dots \\ \wedge\ & ct([L^{-1}(\ell_0@t), L^{-1}(\ell_1@t), L^{-1}(\ell_2@t)], \quad [w\_6(t))]) \end{aligned} \tag{5.65}$$

$$\wedge\ \ell_1@1 = \ell_0@2 \wedge \dots \wedge \ell_1@(t-1) = \ell_0@t \tag{5.66}$$

$$\text{holds by Definition 35} \tag{5.67}$$

$$\Rightarrow\ \exists \text{ a trace } t_1 : q_0 \xrightarrow{\sigma_1^{t_1}} \dots \xrightarrow{\sigma_k^{t_1}} q_{n_1} \text{ is consistent with } \theta_1 = obs(\sigma_1^{t_1} \dots \sigma_k^{t_1}) \tag{5.68}$$

$$\wedge \exists \text{ a trace } t_2 : q'_{n_1} \xrightarrow{\sigma_1^{t_2}} \dots \xrightarrow{\sigma_k^{t_2}} q_{n_2} \text{ is consistent with } \theta_2 = obs(\sigma_1^{t_2} \dots \sigma_k^{t_2}) \tag{5.69}$$

$$\wedge \dots \tag{5.70}$$

$$\Rightarrow \exists \text{ a trace } t : \langle q_0, 0\rangle \xrightarrow{\sigma_1^{t_1}} \dots \xrightarrow{\sigma_k^{t_1}} \langle q_{n_1}, k\rangle \xrightarrow{\varepsilon} \langle q'_{n_1}, 0\rangle \xrightarrow{\sigma_1^{t_2}} \dots \tag{5.71}$$

$$\Rightarrow ct([I, L^{-1}(N)], [\Theta]) \text{ holds} \tag{5.72}$$

$$\Rightarrow \Delta(M, \Theta) = N. \tag{5.73}$$

□

This work discusses the complexity of the Al_6-simulation. The size of the Al_6-simulation is $(2k-1)$ times that of the DES model, where k is the time window size. Thus, verifying the precision of Al_6 w.r.t. a DES model using the twin plant method remains polynomial. Remember that a simulation is only used to verify the precision of a diagnostic algorithm w.r.t. a DES model, and not for diagnosis.

5.5 Implementation and Optimisation of Time-Window Algorithms

This section examines the symbolic implementation of the proposed TWAs. It also shows how to minimise the knowledge that needs to be carried over between the time windows of a TWA.

5.5.1 Symbolic Implementation of a Time-Window Algorithm

In this work, a system is modelled symbolically. A state is defined as an assignment of Boolean state variables V. This section defines *memory set* as follows.

Definition 38 (Memory Set) *Given the Boolean state variables V such that $Q = 2^V$, a* memory set *is a particular subset $V_{\mathbb{L}}$ of V that is used for encoding the information that is carried over the time windows.*

State	Memory Set Representation
A	$\neg f \wedge \neg p \wedge \neg q$
B	$\neg f \wedge \neg p \wedge q$
C	$\neg f \wedge p \wedge \neg q$
D	$\neg f \wedge p \wedge q$
F	$f \wedge \neg p \wedge \neg q$
G	$f \wedge p \wedge \neg q$

Table 5.4: A memory set for M_4 in Fig. 5.1

For instance, Tab. 5.4 shows the memory set representation for the states of M_4 in Fig. 5.1. The 6 states are represented by three variables $V = \{f, p, q\}$. The memory set is $V_{\mathbb{L}} = \{f, p\}$ such that f evaluates to true when the state is faulty, i.e. the system is in state F or G; p evaluates to true if the system is in state G or in a state within N_X. The last variable $q \in V \setminus V_{\mathbb{L}}$ in this example evaluates to true if the last event is c. For instance, the abstract state $N_Y \in \mathbb{L}$ is encoded as $\Phi(L^{-1}(N_Y)) = \neg f \wedge \neg p$.

This work concentrates on the symbolic approach for the TWAs using BDD [Schumann et al., 2010]. Other implementation options are the Sampath et al. diagnoser [Sampath et al., 1995], or SAT [Grastien et al., 2007]. Diagnosis is performed by tracking the belief state, i.e. the set of states that the system may be in at a given time. BDD allows to manipulate the sets of states efficiently. The implementation of Al_5 is mostly similar to the standard global model approach proposed by Schumann et al. [2010], except that the belief state is updated between two time windows. The belief state β' at the beginning of the time window i is an abstract state that shares the same label as one state of the belief state β at the end of the time window $(i-1)$, i.e. $\beta' = L^{-1}(L(\beta))$ where L is extended to sets of states. For instance, in Fig. 5.1, if $\beta = \{A\}$, then $L(\beta) = \{N_X\}$, and $\beta' = \{A, B\}$.

The remaining question is how to implement the operations L and L^{-1} using BDD. The symbolic representation allows for a succinct implementation of L. Given a formula $\Phi(C)$ that models a set $C \subseteq \mathbb{L}$ of labels, the formula that models the set of states $L^{-1}(C)$ is the same, i.e. $\Phi(L^{-1}(C)) = \Phi(C)$. Then, the L function can be implemented by a logical existential operator, i.e.

$$\Phi(L(Q')) = \exists(V \setminus V_{\mathbb{L}}).\, \Phi(Q').$$

For example, in Fig. 5.1, $V = \{f, p, q\}$. For $\{A, B\}$, $V_{\mathbb{L}} = \{f, p\}$. Then, $\Phi(L(\{A, B\})) = \exists q.\ \Phi(\{A, B\}) = \neg f \wedge \neg p$.

5.5.2 Optimisation of a Time-Window Algorithm

This section examines how to reduce the set of labels. The aim is to find a minimal memory set $V_{\mathbb{L}}$. This problem is very interesting because it allows to identify the minimal amount of information that is needed to summarise the past observations while maintaining the precision. This work proposes the following *refinement* property of $\mathbb{L}$.

Definition 39 (Refinement of $\mathbb{L}$) *Given two set of labels $\mathbb{L}$ and $\mathbb{L}'$, $\mathbb{L}'$ is a* refinement *of $\mathbb{L}$ if $|\mathbb{L}'| > |\mathbb{L}|$ and the following condition hold.*

$$\forall q, q' \in Q\ ,\ L'(q) = L'(q') \Rightarrow L(q) = L(q').$$

In other words, for any two system states, if they have the same label in the greater set of labels $\mathbb{L}'$, then they still have the same label in the smaller set of labels $\mathbb{L}$; $\mathbb{L}'$ provides more detailed categorisation of the states. This work also proposes the *abstract* function as follows.

Definition 40 (*Abstract* Function) *Given a detailed label L' of a state q, the function* abstract*: $\mathbb{L}' \rightarrow \mathbb{L}$ returns a more abstract label L of q.*

$$abstract(L'(q)) = L(q).$$

Based on Definition 39 and 40, Theorem 12 examines the precision of a TWA using two different sets of state labels.

Theorem 12 (Precision Relation using Refinement) *Given two set of labels $\mathbb{L}$, $\mathbb{L}'$, and a TWA, i.e. Al_5 or Al_6, if $\mathbb{L}'$ is a refinement of $\mathbb{L}$, and the TWA using $\mathbb{L}$ is precise, then the TWA using $\mathbb{L}'$ is also precise.*

Proof of Theorem 12 Let $TWA = Al_5$. Suppose TWA using $\mathbb{L}$ is precise.
The aim is to prove

$$TWA^{\mathbb{L}}(M, \Theta, T) = F \Rightarrow TWA^{\mathbb{L}'}(M, \Theta, T) = F.$$

Equivalently, the aim is to prove

$$TWA^{\mathbb{L}'}(M, \Theta, T) = N \Rightarrow TWA^{\mathbb{L}}(M, \Theta, T) = N.$$

$$\text{Suppose } TWA^{\mathbb{L}'}(M,\Theta,T) = N \text{ and} \tag{5.74}$$

$$\Theta = \theta_1\theta_2\theta_3\ldots\theta_t \tag{5.75}$$

$$\Rightarrow \exists \ell'_1, \ell'_2, \ell'_3, \ldots, \ell'_t \in \mathbb{N} \text{ such that} \tag{5.76}$$

$$\begin{aligned} ct([I, L'^{-1}(\ell'_1)], \quad [\theta_1]) \wedge \\ ct([L'^{-1}(\ell'_1), L'^{-1}(\ell'_2)], \quad [\theta_2]) \wedge \cdots \wedge \\ ct([L'^{-1}(\ell'_{t-1}), L'^{-1}(\ell'_t)], \quad [\theta_t]) \end{aligned} \tag{5.77}$$

$$\text{holds by Definition 34} \tag{5.78}$$

$$\Rightarrow \exists \ell_1, \ell_2, \ell_3, \ldots, \ell_t \in \mathbb{N} \text{ such that} \tag{5.79}$$

$$\begin{aligned} \ell_1 = abstract(\ell'_1) \wedge \\ \ell_2 = abstract(\ell'_2) \wedge \\ \ell_3 = abstract(\ell'_3) \wedge \cdots \wedge \\ \ell_t = abstract(\ell'_t) \wedge \\ ct([I, L^{-1}(\ell_1)], \quad [\theta_1]) \wedge \\ ct([L^{-1}(\ell_1), L^{-1}(\ell_2)], \quad [\theta_2]) \wedge \cdots \wedge \\ ct([L^{-1}(\ell_{t-1}), L^{-1}(\ell_t)], \quad [\theta_t]) \end{aligned} \tag{5.80}$$

$$\text{holds} \tag{5.81}$$

$$\Rightarrow TWA^{\mathbb{L}}(M,\Theta,T) = N \tag{5.82}$$

The case of Al_6 can be proved accordingly.

□

Based on Theorem 12, Corollary 1 studies the impact of adding more variables to the memory set.

Corollary 1 *If a TWA is precise w.r.t. a given model with a memory set $V_{\mathbb{L}}$, then for all other variable $v \in V \setminus V_{\mathbb{L}}$, it is also precise for set $V_{\mathbb{L}} \cup \{v\}$.*

Proof of Corollary 1 The smaller $V_{\mathbb{L}}$, the bigger the superset of nominal consistent traces determined by a TWA. If a TWA concludes F for a given $V_{\mathbb{L}}$, then the superset of nominal traces is empty. For any $V_{\mathbb{L}} \cup \{v\}$, this superset is also empty. Therefore, the precision is preserved.

□

Based on Corollary 1, Procedure 1 computes a minimal memory set. Firstly, it chooses the variable to add to $V_{\mathbb{L}}$ (Line 1–5). Starting with a set with one variable to identify the faulty states, variables are added until the algorithm becomes precise. A simulation for a TWA and a DES model M may generate a "witness", i.e. a faulty

Procedure 1(Add and Optimise Variables)

Input: TWA, M
Output: A minimal memory set $V_{\mathbb{L}}$

```
1  V_L := {f}
2  while TWA is not precise for M do
3      w := witness
4      Identify v in w
5      Add v to V_L
6  foreach v ∈ V_L do
7      Remove v from V_L
8      if TWA is not precise for M then
9          Reinsert v to V_L
10 return V_L
```

trace of the system together with the nominal trace on the simulation that exonerates the faulty trace (Line 2). Because the system is diagnosable, the simulation trace necessarily contains one or more ε-transitions that change the value of one or more state variables. Then, Procedure 1 chooses one of these variables, which will exclude this witness for later precision verification.

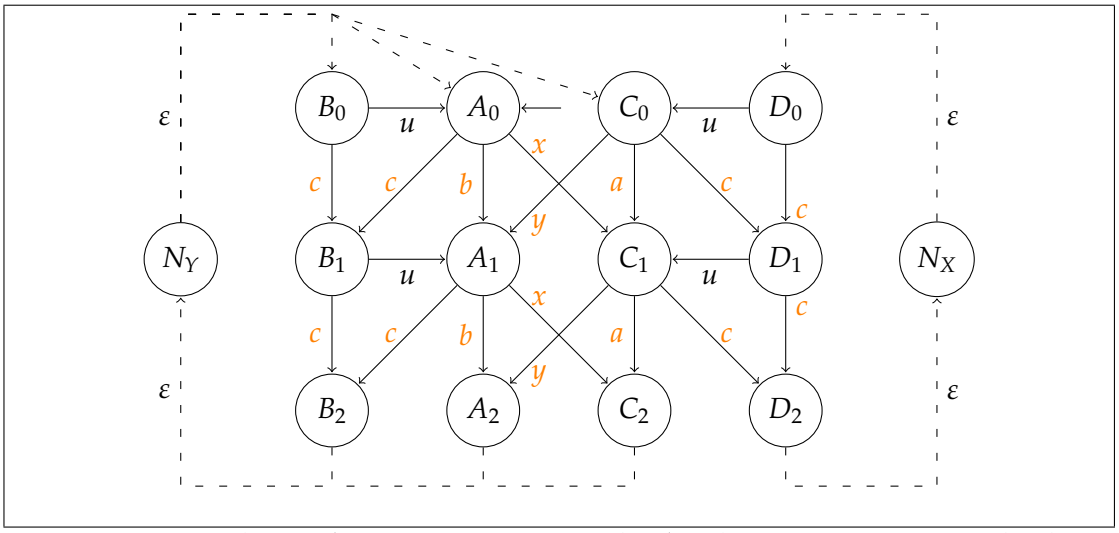

Figure 5.6: Simulation for M_4 in Fig. 5.1 and Al_5: the state partitioning leads to imprecision.

For instance, Fig. 5.6 shows the Al_5-simulation for M_4 with an imprecise state partitioning, i.e. states A, B, C are in N_Y, and state D is in N_X. Tab. 5.5 shows the diagnostic result of '$xacbbb$' using Al_5 where the time window size is 2.

- For the first time window xa, there exists a nominal trace that begins with the

Algorithm	Slice	ABS	Diagnosis
Δ	$xacbbb$	NA	F (precise)
Al_5	xa	$\{N_Y\}$	N (imprecise)
	cb	$\{N_Y\}$	
	bb	$\{N_Y\}$	

Table 5.5: Diagnostic results of Al_5 for M_4, and observations $\Theta = xacbbb$: ABS refers to the abstract belief state at the end of each time window.

initial state, and is compatible with this time window, i.e. $A \xrightarrow{x} C \xrightarrow{a} C$. In particular, $L(C)$ evaluates to N_Y. The ABS at the end of this time window is N_Y. Thus, the first time window leads to N_Y, and the next time window should continue from any state with the abstract state of N_Y.

- For the second time window cb, there exists a nominal trace that begins with a state in N_Y, and is compatible with this time window, i.e. $A \xrightarrow{c} B \xrightarrow{u} A \xrightarrow{b} A$. In particular, $L(A)$ evaluates to N_Y. The ABS at the end of this time window is N_Y. Thus, the second time window leads to N_Y, and the next time window should continue from any state with the abstract state of N_Y.

- For the third time window bb, there exists a nominal trace that begins with a state in N_Y, and is compatible with this time window, i.e. $A \xrightarrow{b} A \xrightarrow{b} A$. In particular, $L(A)$ evaluates to N_Y. The ABS at the end of this time window is N_Y.

Thus, the diagnostic output of Al_5 is N. However, the diagnostic result of using the exact diagnosis Δ is F. Consequently, '$xacbbb$' is a witness. This witness is caused by an ε-transition from C_2 to A_0, i.e. the trace of the first time window ends in C while the trace of the second time window starts with A. Furthermore, states A and C have different values of p. Thus, p needs to be added to the memory set.

In the second part of Procedure 1 , all variables are examined to remove the ones that are unnecessary to maintain the precision (Line 6–10). Notice that the procedure remains polynomial since each loop is applied at most a linear number of times. This procedure bears similarity to Brandán-Briones et al. [2008], which finds a minimal subset of observable events while ensuring diagnosability.

5.6 Summary

This chapter first reviews the IWAs, which are proposed in Chapter 4. IWAs is a class of window-based diagnostic algorithms. The most distinct characteristic is that

each IWA slices an observation sequence into multiple time windows in different ways, and remembers no information between time windows during the diagnostic process. The aim of IWAs is to improve the flexibility of diagnosis because time windows are diagnosed separately and in parallel. Also, the complexity of diagnosis is reduced because the size of the time windows is bounded. On the other hand, the precision of IWAs may be reduced as the links between the time windows are lost. For instance, this happens when a fault can be diagnosed only by observing two specific events; however, these two events are not guaranteed to appear in the same time window.

Inspired by IWAs, this chapter proposes TWAs, and presents two instances, namely Al_5 and Al_6. TWAs differ from IWAs that TWAs remember some of the knowledge about the current diagnosed states. Al_5 remembers the key information from the previous consecutive time window while Al_6 remembers the history from the previous overlapping time window. Although the TWAs are not computing the exact diagnosis, Chapter 3 proposes the theory of simulation to verify the precision of a non-exact diagnostic algorithm w.r.t. a DES model. This chapter demonstrates how to verify the precision of each TWA w.r.t. a DES model by building a simulation. Finally, this work discusses the symbolic implementation using BDD, and proposes an procedure that minimises the amount of information that a TWA needs to carry over.

In relation to intelligent alarm log handling, TWAs presented in this chapter reduce the diagnostic complexity compared to using exact diagnosis. Deriving the belief state using exact diagnosis is proven to be exponential w.r.t. the number of system states [Rintanen, 2007], which makes it impractical in a real-time and critical environment such as electricity distribution and telecommunication control. In contrast, a TWA reduces the computational complexity because using abstract states is very helpful to reduce the size of a belief state, and thus reduce memory use.

In conclusion, the contribution of this chapter is developing from the naive window-based diagnosis to more advanced diagnostic algorithms, which slice a sequence of observations and diagnose the time windows. This approach has the advantage that it does not require maintaining a precise estimate of the system state. The outline for the further work includes analysing the probability of the system states when resetting a time window. It is also non-trivial to study how to apply window-based diagnosis to non-diagnosable systems since the IWAs and TWAs focus on diagnosable systems so far. In general, the goal is to develop more advanced window-based diagnostic algorithms that reduces the computational complexity while maintaining the capability to verify the precision of diagnosis.

Chapter 6

Experimentation

This work proposes window-based diagnostic algorithms. Chapter 4 has presented four IWAs, and Chapter 5 has presented two TWAs. This chapter evaluates the performance of the IWAs and the TWAs, which is measured by the precision of diagnosis, computational time, maximum memory use, average memory use, and diagnostic distance. Diagnostic distance is defined as the number of observations between a fault occurrence and the fault diagnosis of a diagnostic algorithm. This work compares the above aspects with the exact diagnostic algorithm Al_0 encoded using BDD as proposed by Schumann [2007]. This chapter also examines the impact of the time window size on the performance of a window-based diagnostic algorithm.

This chapter is organised as follows. Section 6.1 describes the implementations that read a DES model with multiple components and an observation sequence to diagnose using the window-based diagnostic algorithms, i.e. IWAs and TWAs. The implementations also include the exact diagnostic algorithm Al_0 encoded by BDD. The theory of the exact diagnosis using BDD was proposed by Schumann [2007]. However, there is no corresponding implementation available. This work implements the exact diagnostic algorithm Al_0 encoded by BDD, which supports diagnosis on a DES model with multiple components. Section 6.2 describes the experimental models and scenarios for diagnosis. Section 6.3 reports the experimental settings for IWAs, the data gathered from the experiments, and the evaluations. Section 6.4 reports the experimental settings for TWA Al_5, the data gathered from the experiments, and the evaluations. Section 6.4 reports the experimental settings for TWA Al_6, the data gather from the experiments, and the evaluations. Section 6.6 concludes this chapter, and provides an outline for the future work. Appendix A shows the figures for the "robot" model as specified in Section 6.2.

6.1 Implementations for Experiments

This section describes the original diagnostic software program of this work, which is written in C++ 2011 standard. The theory of exact diagnosis using BDD was proposed by Schumann [2007]. However, there is no corresponding implementation available. This work implements the exact diagnostic algorithm Al_0 encoded by BDD, which supports diagnosis on a DES model with multiple components. Component-based modelling has the advantage to represent a large-scale system. For instance, m components with n states in each component can represent a system with n^m states. Notice that events with the same name are synchronised among all components, which applies to all observable, unobservable, and faulty events.

6.1.1 Input and Output of Diagnostic Software

The input of this diagnostic software program consists of two parts:

- the descriptions of each component of a DES model, and
- a sequence of events that are observed from the system.

The first part of the input is a set of text files where each file describes one component. The format of a file is based on the *des_comp* format used in the *Dia-Des Tool* developed by Yannick Pencolé from LAAS-CNRS [1]. This work includes the additional specifications of the nominal and faulty states of each component because this work considers faults at the state level, rather than the event level, as stated in Chapter 3. A *des_comp* file provides the specifications of a component, which is the described as follows.

- the name of a component,
- states,
- faulty events,
- unobservable events,
- observable events,
- faulty states,
- nominal states, and

[1] homepages.laas.fr/ypencole/diades/html/index.html

- transitions including the beginning state, the event name, and the ending state of each transition.

For TWA Al_5 and Al_6 only, the input further includes a set of text files where each file describes the abstract states of a component.

The second part of the input is a text file with a sequence of events that are observed from the system. In terms of diagnosis, if every trace that can explain the system behaviour ends in a faulty state, then there is no nominal explanation, and the system is faulty. The diagnostic output is "F" as soon as one component is faulty, and thus the system is diagnosed as faulty. The output is "N" if every component of the system is nominal. As stated in Assumption 6 of Chapter 3, this work considers one fault in a system at a time.

6.1.2 Diagnostic Software

This software program consists of 14 classes, which are visualised in the UML diagram in Fig. 6.1.

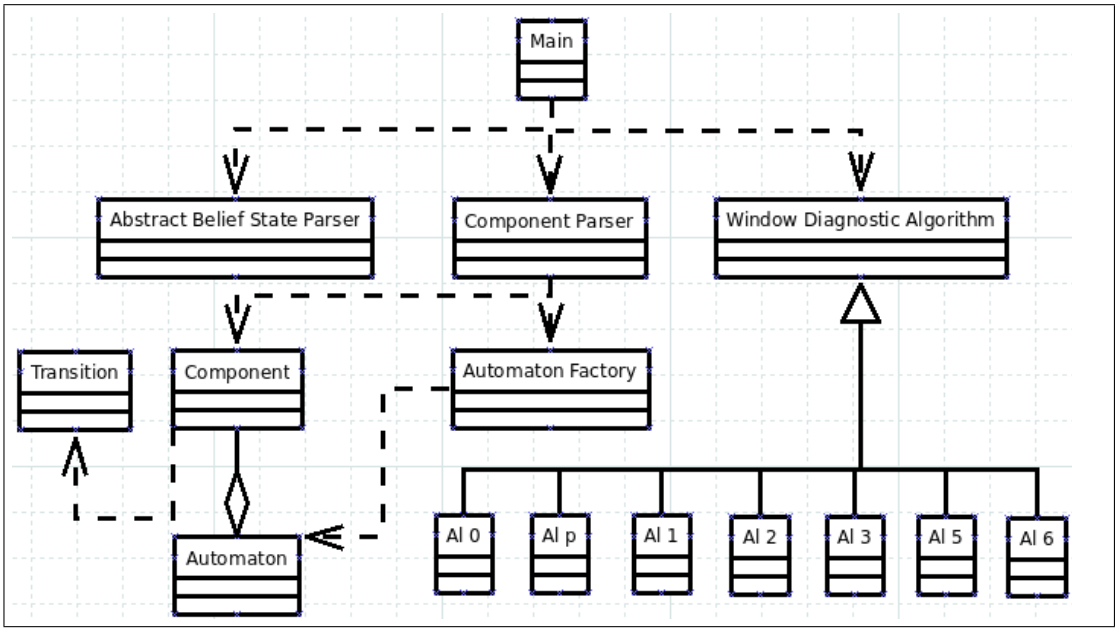

Figure 6.1: UML diagram for the diagnostic software

The 14 classes are described as follows.

- The `Component Parser` class reads a set of text files. Each file describes one component of a DES model, and is in the modified *des_comp* format as specified in Section 6.1.1.

- The `Component` class records all of the information about one component including states, faulty events, unobservable events, observable events, faulty states, nominal states, and transitions.
- The `Automaton` class records all of the information about the global DES model, which consists of the underlying components.
- The `Automaton Factory` class encodes all states, events, and transitions into BDD formulae. Also, this class generates an instance of an `Automaton`. This work uses BDD to encode the states, events, and transitions of each component in a DES model. This software program uses the *BUDDY Library* [2], which supports all of the required operations and management on BDD variables in C++. The BDD encodings have been reviewed in Section 2.1.2 of Chapter 2.
- The `Transition` class is a tool used by the `Automaton Factory` class when encoding transitions.
- The `Window Diagnostic Algorithm` class is a high-level class, which will be extended by the exact diagnostic algorithm Al_0, four IWAs, and two TWAs. This class diagnoses one time window, which is a slice of the observation sequence.
- The `Al 0` class extends the Window Diagnostic Algorithm class. It implements the exact diagnostic algorithm Al_0, which supports diagnosing a DES model with multiple components. The theory of diagnosis using BDD was proposed by Schumann [2007].
- The `Al p` class implements IWA Al_p, as defined in Section 4.2.1.
- The `Al 1` class implements IWA Al_1, as defined in Section 4.2.2.
- The `Al 2` class implements IWA Al_2, as defined in Section 4.2.3.
- The `Al 3` class implements IWA Al_3, as defined in Section 4.2.4.
- The `Abstract Belief State Parser` class is only applicable to TWA Al_5 and Al_6. This class reads the BDD variables that indicate the abstract states of each component.
- The `Al 5` class implements TWA Al_5, as defined in Section 5.2.
- The `Al 6` class implements TWA Al_6, as defined in Section 5.3.

[2] buddy.sourceforge.net/manual/main.html

- The `Main` class runs this program. This class also reads the entire observation sequence.

The processes of the software program are described as follows. Firstly, the `Main` class calls the `Component Parser` to read the descriptions for all components, and stores the information for each instance of `Component`. Secondly, `Automaton Factory` generates an instance of `Automaton`, which consists of the underlying components. Finally, the `Main` class reads a sequence of observations, and calls one of the diagnostic algorithms, i.e. Al_0, Al_1, Al_2, Al_3, Al_5, or Al_6 class.

6.2 Experimental Models and Scenarios for Diagnosis

There is no established benchmark for DES diagnosis in the literature. This work proposes a benchmark for DES diagnosis. This section explains the "robot" DES model that is used in the experiments. It then explains how to generate a scenario for diagnosis.

6.2.1 The "Robot" Model

This work establishes a DES model for a "robot". This "robot" model consists of four categories of components, namely,

- `Order Monitor`,
- `Fault Monitor`,
- `Customer`, and
- `Glass`.

There is only one instance of `Order Monitor` and one instance of `Fault Monitor`. In a "robot" model, $|c|$ denotes the pre-defined number of customers, and $|g|$ denotes the pre-defined number of glasses. This "robot" model also considers the situations where not every customer is ordering and not every glass is being used. Events with the same label are synchronised, which applies to all observable, unobservable, and faulty events.

The details of the four components are described as follows with visualisations. The visualisations illustrate the case of two customers and two glasses, i.e. $|c| = 2$ and $|g| = 2$. Notice that all of the figures mentioned in this section will appear in Appendix A.

- The `Order Monitor` component manages the sequencing of the orders from the customers. Fig. A.1 is a visualisation for the `Order Monitor` for maximum two customers and maximum two glasses. Observable events are `order_1` and `order_2` while the rest events are unobservable. The constraint is that once a customer order is received (`order_1` or `order_2`), a glass should be selected (`use_1` or `use_2`). However, exactly which glass has been selected is not known, nor observable. Then, the selected glass is given to the customer (`g_to_c` or `damaged_g_to_c`). It is not known, nor observable whether the selected glass is normal or damaged. The number of transitions with an `order` event is $|c|$, and the number of transitions with a `use` event is $|g|$.

- The `Fault Monitor` component tracks whether there is a damaged glass. Fig. A.2 is a visualisation for the `Fault Monitor` for maximum two glasses. The "robot" model specifies that either no glass is damaged, or there is one damaged glass. This specification is consistent with Assumption 6 of Chapter 3, which states that there is at most one fault in the system at any time. The number of transitions with a `damage_g` event is $|g|$.

- The `Customer` component models how a customer places an order, and receives a glass. Whether a customer leaves a tip or not depends on the condition of the glass that is used. The customer leaves a tip if the glass is normal. Otherwise, the glass is damaged, and the customer departs without leaving a tip. Fig. A.3 and Fig. A.4 are the visualisations for two instances of the `Customer` component. Notice that when a customer is in the `waiting` state, the `Customer` component is not affected by any `g_to_c` event, or `damaged_g_to_c` event. This is because the loops with `g_to_c` and `damaged_g_to_c` events are used to model the sequencing of multiple orders and glasses, i.e. a glass is given to one customer while the other customers are not affected.

- The `Glass` component models the fact that a glass is selected after receiving an order from a customer. Then, the glass is filled with drink and is given to the customer. However, the glass may or may not be damaged. After the customer leaves, the glass is washed. Fig. A.5 and Fig. A.6 are the visualisations for two instances of the `Glass` component.

The "robot" model describes the processes that a "robot" receives orders from one or two customers. It then selects a glass for every customer order, and gives the glass to the customer that is filled with drink. Exactly which glass is selected is not known, nor observable. At any time, at most one glass may be damaged, which is in accordance to Assumption 6 of Chapter 3, i.e. this work considers one fault in a

system at a time. The "robot" receives tips from a customer if the glass is normal, and does not receive tips if the glass is damaged. Finally, the "robot" washes all of the used glasses. Exactly which glass is washed is observable.

Diagnosis of the "robot" system aims to identify exactly which glass is damaged. In general, diagnosis of the "robot" system requires multiple rounds of order-tip-wash. For example, if the scenario is `order_1`, `order_2`, `no_tip`, `tip`, `wash_3`, `wash_4`, then there are two orders, which are received from customer 1 and 2. Also, glass 3 and 4 are used and washed. Based on this scenario, a diagnoser infers that either glass 3 or 4 is damaged. However, a diagnoser requires more observations in order to determine exactly which glass is damaged. For example, if this scenario continues with `order_1`, `no_tip`, `wash_3`, then a diagnoser infers that glass 3 has been used and washed. In particular, a diagnoser determines that glass 3 is damaged based on `no_tip`, `wash_3`. In general, a diagnoser observes `no_tip`, and then analyses which glasses have been used and washed so that it narrows down the set of faulty glasses to a single faulty glass.

6.2.2 Generating Scenarios for Diagnosis

The work also implements a scenario script to generate the diagnostic scenarios. Each diagnostic scenario consists of a specific number of *rounds*. Each round consists of three groups of observations, namely,

- `order`,
- `tip` or `no_tip`, and
- `wash`.

When generating a scenario, the scenario script specifies the index of the round where a glass is damaged, denoted as r. Thus, the first $(r-1)$ rounds are free of damaged glasses. Then, the scenario script records which glass is damaged, i.e. starting from the r-th round, it is possible that a damaged glass is selected and given to a customer. The three groups of observations are generated as follows.

- The first group is a random number of customer orders. The number of customer orders must be within the limit of the maximum number of customers. For example, if the maximum number of customers is 2, then it is allowed to generate at most two `order_i` observations, i.e. `order_1` and `order_2`.
- The second group is `tip` or `no_tip` observations depending on the condition of a selected glass. The total number of `tip` and `no_tip` observations is the same

as the number of orders in the first group. Since the "robot" model specifies the number of glasses as $|g|$, the scenario script randomly selects one glass from $|g|$ glasses for an order, denoted as `glass_i`. Notice that the scenario script only simulates how the "robot" model operates. From the observations of the "robot" model, it is not known exactly which glass is used, nor exactly which glass is damaged, i.e. `glass_i` is not observable. If a normal glass is selected, then the scenario script generates observable `tip_i` w.r.t. its `order_i` from the first group. Otherwise, a damaged glass is used, and the scenario script generates observable `no_tip_i` w.r.t. its `order_i`.

- The third group is `wash_i` observations w.r.t. the `glass_i` from the second group.

Using the scenario script, this work simulates a diagnostic scenario where a fault occurs in the "robot" model. This work also records the observation index for the fault occurrence in a scenario. Section 6.3 will use the scenario script and the recorded data for experiments and evaluations. Although the "robot" model is not diagnosable by Definition 8 of Chapter 3, this work conducts experiments on Al_1, Al_2, Al_3, Al_5, and Al_6 using the scenario script to generate a diagnostic scenario with a fault. If the exact diagnostic algorithm Al_0 diagnoses a scenario as faulty, then this scenario is diagnosable and it is retained for experiments.

6.3 Experimental Settings, Results, and Evaluations for IWAs Al_1, Al_2, and Al_3

This work conducts experiments on IWAs Al_1, Al_2, and Al_3. It then evaluates the performance of each IWA, and compares the experimental results with those of the exact diagnostic algorithm Al_0.

6.3.1 Experimental Settings

This work conducts experiments on the "robot" model with 14 glasses, 9 customers, a fault monitor, and an order monitor. Section 6.2 has specified that there are 6 states in the `Glass` component, 4 states in the `Customer` component, 3 states in the `Order Monitor` component, and 2 states in the `Fault Monitor` component. Using Al_0, the number of states in a belief state is up to 1.23256×10^{17}, i.e. $6^{14} \times 4^9 \times 3 \times 2$.

Using the scenario script described in Section 6.2, this work generates 500 scenarios. Each diagnostic algorithm is evaluated in terms of the following five aspects.

- The first aspect is the precision of diagnosis. Al_0 is an implementation of the exact diagnosis encoded by BDD. Al_0 diagnoses all of the 500 scenarios as faulty. The diagnostic results of a window-based diagnostic algorithm will be compared with those of Al_0.
- The run time of diagnosing a scenario is measured in seconds. Run time does not include the time for the following operations:
  - initialising the C++ program,
  - reading the specifications of all components and a diagnostic scenario, and
  - BDD encodings for states, events, transitions, and synchronisations.
- The peak memory use is measured by the maximum number of BDD nodes among belief states.
- The average memory use is measured by the average number of BDD nodes among belief states.
- The final aspect is the number of observations between a fault occurrence and the fault diagnosis of a diagnostic algorithm, denoted as *diagnostic distance*. This aspect provides an insight on the minimum number of required observations to diagnose a fault. Section 6.2.2 explains that the scenario script generates a set of scenarios where the index of a fault occurrence is recorded for each scenario.

The experiments were conducted on a Macintosh OS 10.9.5 computer with a 2 GHz quad-core CPU and 8 GB memory. Four sets of experiments were run on 500 scenarios. Recall that Chapter 4 denotes a time window size as k, the index of the starting observation in a scenario as i, and the number of skipped time windows of Al_3 as d. Let the total number of observations in a scenario be n. The settings of each experiment are described as follows.

- Experiment 1: the exact diagnostic algorithm Al_0;
- Experiment 2: Al_1 where $k = 150, 200, 250$, and $i = 0$;
- Experiment 3: Al_2 where $k = 150, 200, 250$, and $i = 0$;
- Experiment 4: Al_3 where $k = 150, 200, 250, 300, 350$, $i = \frac{\lfloor \frac{n}{k} \rfloor \times k}{2}$, and $d = k$.

Notice that the setting for experiment 4 states that the starting index is the middle of a diagnostic scenario. The aim is to test the diagnostic capability of Al_3 when skipping a specified number of observations and diagnosing the rest observations.

This work evaluates Al_1, Al_2, and Al_3 by comparing their results with those of Al_0.

- Section 6.3.2 will compare the results of experiment 1 with those of experiment 2;
- Section 6.3.3 will compare the results of experiment 1 with those of experiment 3;
- Section 6.3.4 will compare the results of experiment 1 with those of experiment 4.

This work also examines the impact of different time window sizes on Al_1, Al_2, and Al_3.

6.3.2 Comparison between Experiment 1 and 2

This work compares the experimental results of Al_1 ($k = 150, 200, 250$) with those of Al_0, and evaluates their performance in five aspects as specified in Section 6.3.1.

- Precision:

  The exact diagnostic algorithm Al_0 diagnoses all of the 500 scenarios as faulty. The average diagnostic distance of using Al_0 is 72, which provides a valuable indication for the time window size selection. This set of experiments runs on Al_1 with $k = 150, 200, 250$. However, Al_1 ($k = 150, 200$) are unable to precisely diagnose some of the 500 scenarios. The percentage of precise diagnosis of Al_1 ($k = 150$) is 92.80%, and the percentage of precise diagnosis of Al_1 ($k = 200$) is 98.00%. Al_1 ($k = 250$) precisely diagnoses the fault in each of the 500 scenarios.

- Run time:

  Fig. 6.2 compares the run time of the first 50 scenarios between Al_0 and Al_1 ($k = 250$) measured in seconds. This plot shows the time differences for the first 50 scenarios. Notice that this set of experiments was run on 500 scenarios. The average time increase of using Al_1 ($k = 250$) is 60.63%.

- Peak memory use:

  Fig. 6.3 compares the maximum number of BDD nodes in a belief state of the first 50 scenarios. Al_1 ($k = 250$) consistently has larger peak memory use than Al_0 does. After "resetting" to every nominal state, diagnosing observed events in a new time window increases the size of the global belief state and the number of BDD nodes to represent the global belief state. The average increase of using Al_1 ($k = 250$) is 150.49%.

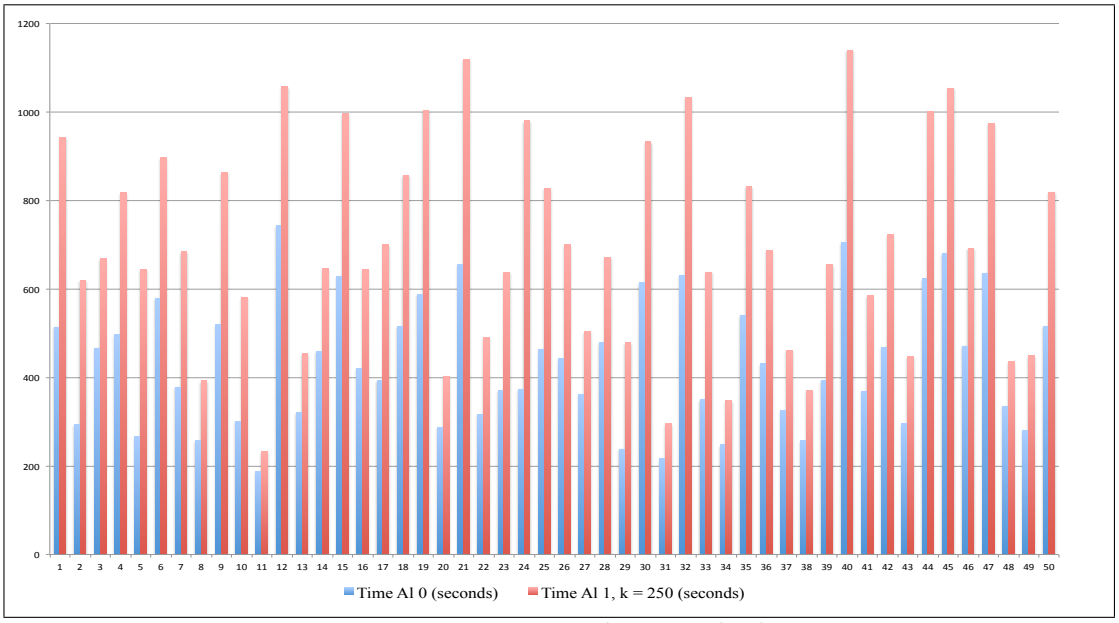

Figure 6.2: Run time comparison between Al_0 and Al_1 ($k = 250$): the x-axis shows the scenario IDs, and the y-axis shows the run time in seconds

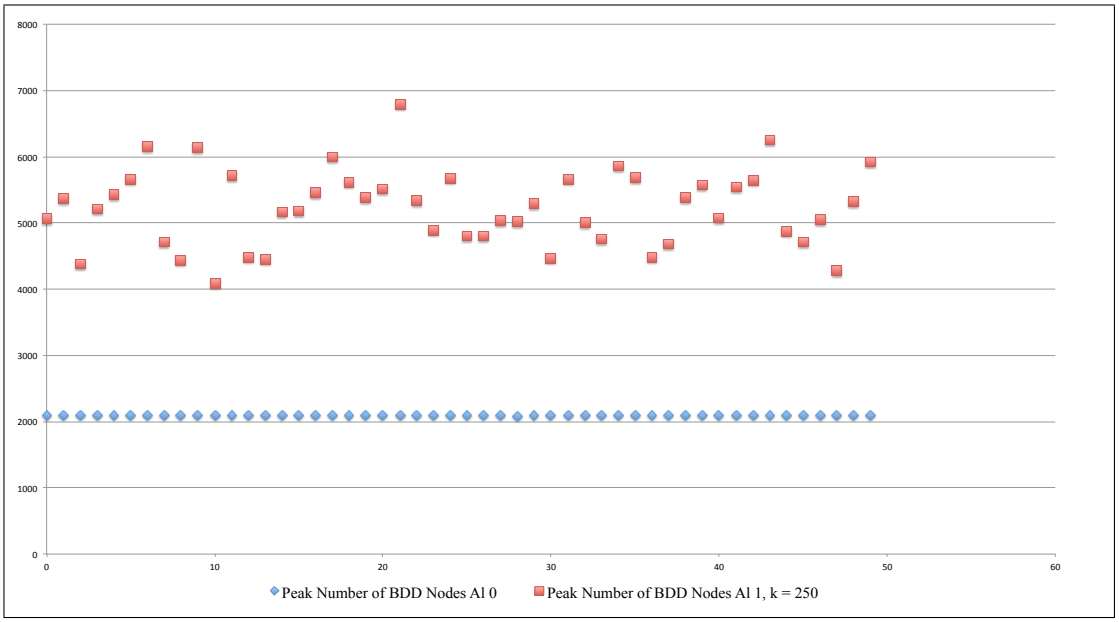

Figure 6.3: Peak number of BDD nodes in a belief state: comparison between Al_0 and Al_1 ($k = 250$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

- Average memory use:

  Fig. 6.4 compares the average number of BDD nodes in a belief state. Al_1 ($k = 250$) has larger average memory use than Al_0 does. The average increase of using Al_1 ($k = 250$) is 42.40%.

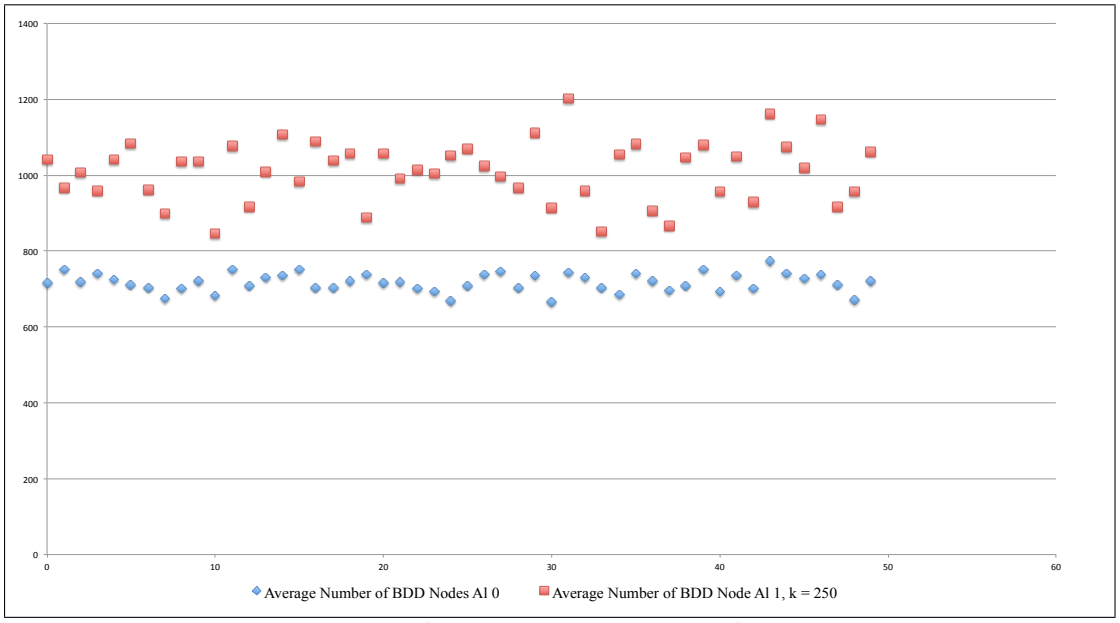

Figure 6.4: Average number of BDD nodes in a belief state: comparison between Al_0 and Al_1 ($k = 250$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

- Diagnostic distance:

  The average number of observations in one scenario is 1490. The average diagnostic distance of using Al_1 ($k = 250$) is 199. In comparison, the average diagnostic distance of using Al_0 is 72.

In summary, although the computational time is increased compared to using Al_0, this set of experiments is helpful to find a small time window size k that is able to precisely diagnose a fault. For a "robot" model with 14 glasses and 9 customers, Al_1 ($k = 250$) precisely diagnoses all 500 scenarios. However, Al_1 ($k = 150, 200$) are unable to precisely diagnose some of the 500 scenarios. The percentage of precise diagnosis of Al_1 ($k = 150$) is 92.80%, and the percentage of precise diagnosis of Al_1 ($k = 200$) is 98.00%. Therefore, this set of experiments provides a useful indication on how small the value of k can be while being able to precisely diagnose most scenarios of the "robot" model with 14 glasses and 9 customers.

6.3.3 Comparison between Experiment 1 and 3

This work compares the experimental results of Al_2 ($k = 150, 200, 250$) with those of Al_0, and evaluates their performance in five aspects as specified in Section 6.3.1.

- Precision:

The exact diagnostic algorithm Al_0 diagnoses all of the 500 scenarios as faulty. The average diagnostic distance of using Al_0 is 72, which provides a valuable indication for the time window size selection. This set of experiments runs on Al_2 with $k = 150, 200, 250$. However, Al_2 ($k = 150, 200$) are unable to precisely diagnose some of the 500 scenarios. The percentage of precise diagnosis of Al_2 ($k = 150$) is 96.20%, and the percentage of precise diagnosis of Al_2 ($k = 200$) is 99.00%. Al_2 ($k = 250$) precisely diagnoses the fault in each of the 500 scenarios.

- Run time:

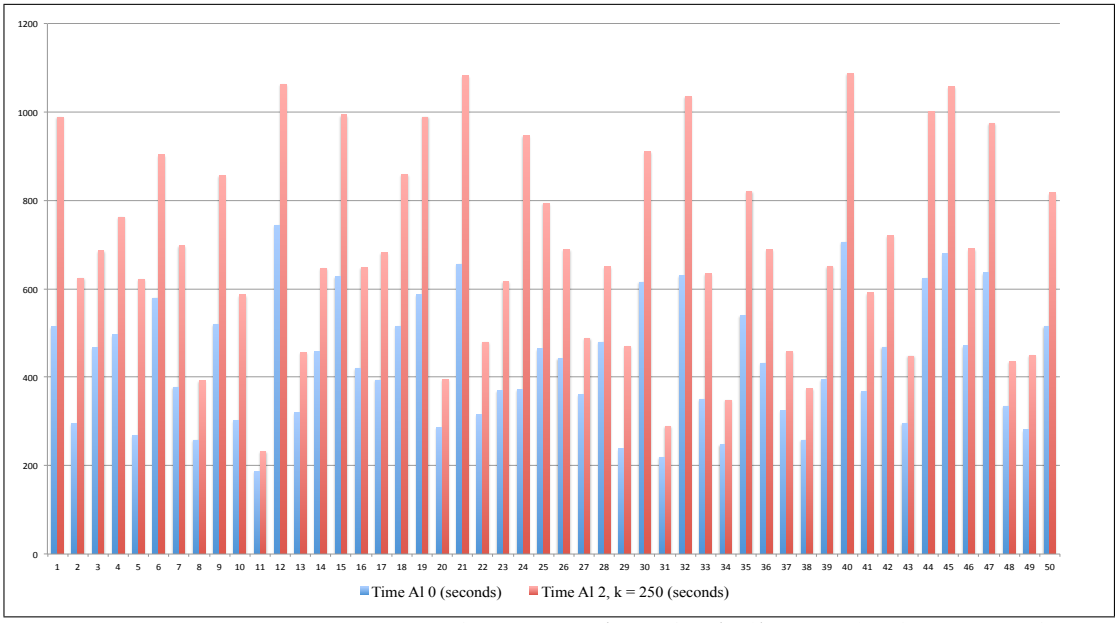

Figure 6.5: Run time comparison between Al_0 and Al_2 ($k = 250$): the x-axis shows the scenario IDs, and the y-axis shows the run time in seconds

Fig. 6.5 compares the run time of the first 50 scenarios between Al_0 and Al_2 ($k = 250$) measured in seconds. This plot shows the time differences for the first 50 scenarios. Notice that this set of experiments was run on 500 scenarios. The average time increase of using Al_2 ($k = 250$) is 60.20%.

- Peak memory use:

Fig. 6.6 compares the maximum number of BDD nodes in a belief state for the first 50 scenarios. Al_2 ($k = 250$) predominantly has larger peak memory use than Al_0 does. After "resetting" to every nominal state, diagnosing observed events in a new time window increases the size of the global belief state and the number of BDD nodes to represent the global belief state. The average increase of using Al_2 ($k = 250$) is 150.49%.

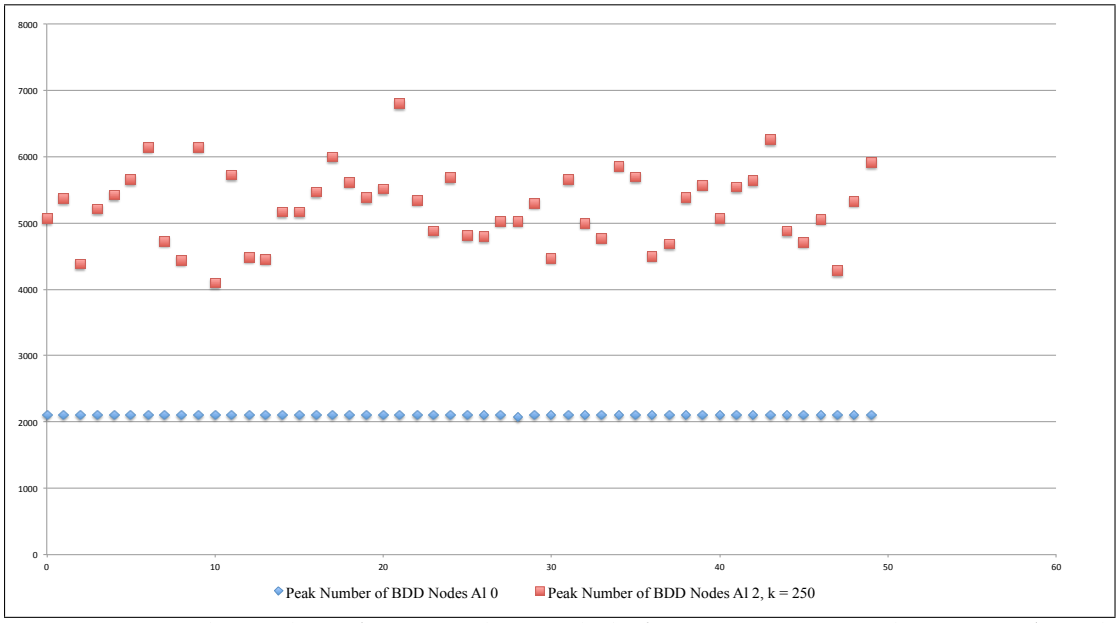

Figure 6.6: Peak number of BDD nodes in a belief state: comparison between Al_0 and Al_2 ($k = 250$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

- Average memory use:

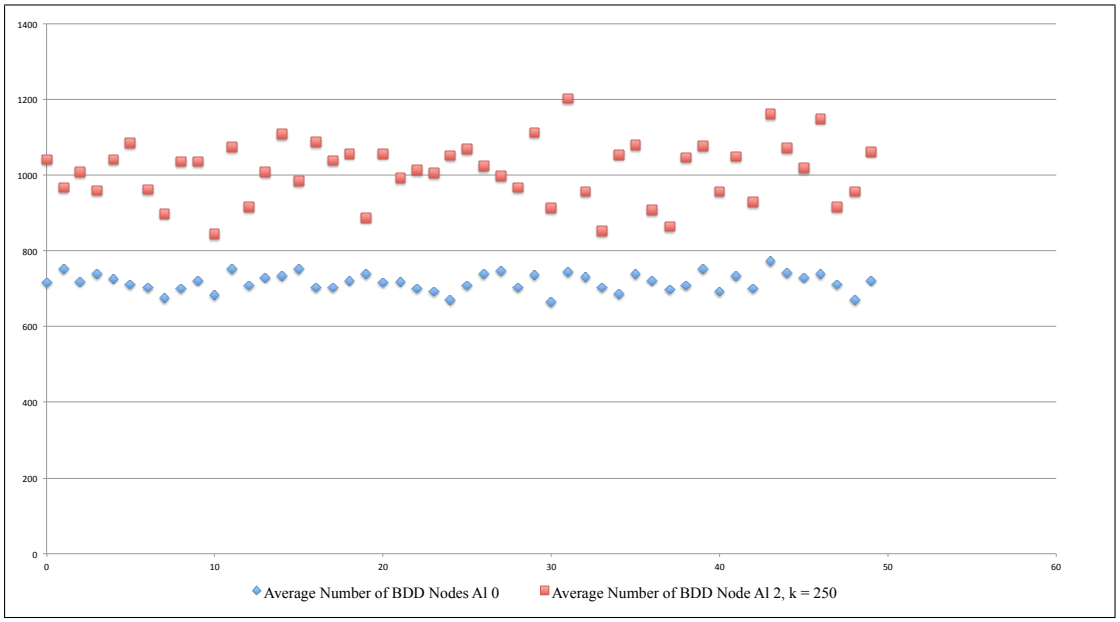

Figure 6.7: Average number of BDD nodes in a belief state: comparison between Al_0 and Al_2 ($k = 250$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

Fig. 6.7 compares the average number of BDD nodes in a belief state. Al_2 ($k = 250$) has larger memory use than Al_0 does. The average increase of using

Al_2 ($k = 250$) is 42.40%.

- Diagnostic distance:

  The average number of observations in one scenario is 1490. The average diagnostic distance of using Al_2 ($k = 250$) is 196. In comparison, the average diagnostic distance of using Al_0 is 72.

In summary, although the computational time is increased compared to using Al_0, the result of this set of experiments indicates that Al_2 ($k = 250$) precisely diagnoses all 500 scenarios of a "robot" model with 14 glasses and 9 customers. In particular, the result of Al_2 ($k = 250$) is consistent with Theorem 9 of Chapter 4 since the time windows of Al_2 ($k = 250$) *cover* those of Al_1 ($k = 250$). This set of experiments also runs on Al_2 ($k = 150, 200$). However, Al_2 ($k = 150, 200$) are unable to precisely diagnose some of the 500 scenarios. Therefore, this set of experiments provides a useful indication on how small the value of k can be while being able to precisely diagnose most scenarios of the "robot" model with 14 glasses and 9 customers.

6.3.4 Comparison between Experiment 1 and 4

The settings for Al_3 are $k = 150, 200, 250, 300, 350$, $i = \frac{\lfloor \frac{n}{k} \rfloor \times k}{2}$, and $d = k$ where n is the total number of observations in a scenario, and k is the size of a time window. In other words, Al_3 skips the first half of a scenario, and starts diagnosis. The average number of observations in one scenario is 1493. This work compares the experimental results of Al_3 ($k = 150, 200, 250, 300, 350$) with those of Al_0, and evaluates their performance in five aspects as specified in Section 6.3.1.

- Precision:

  The exact diagnostic algorithm Al_0 diagnoses all of the 500 scenarios as faulty. This set of experiments runs on Al_3 with $k = 150, 200, 250, 300, 350$. However, Al_3 ($k = 150, 200, 250, 300$) are unable to precisely diagnose some of the 500 scenarios. Al_3 ($k = 350$) precisely diagnoses the fault in each of the 500 scenarios.

- Run time:

  Fig. 6.8 compares the run time of the first 50 scenarios between Al_0 and Al_3 ($k = 150$) measured in seconds. This plot shows the time reduction for the first 50 scenarios. Notice that this set of experiments was run on 500 scenarios. The average time difference of using Al_3 ($k = 150$) is -46.06%.

  Fig. 6.9 compares the run time of the first 50 scenarios between Al_0 and Al_3 ($k = 200$) measured in seconds. This plot shows the time reduction for the first

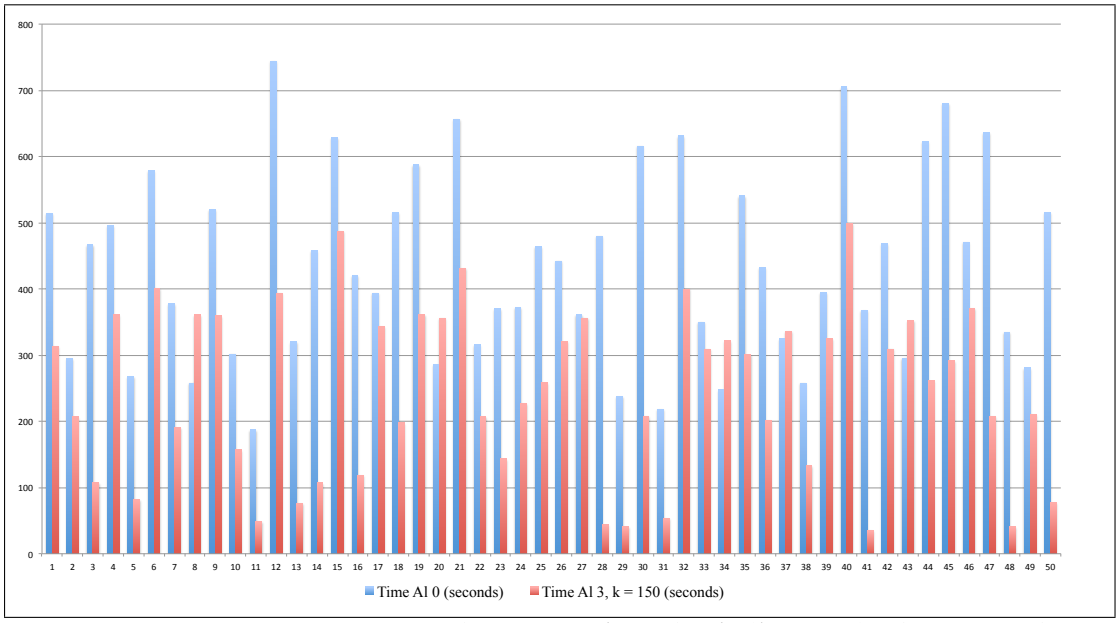

Figure 6.8: Run time comparison between Al_0 and Al_3 ($k = 150$): the x-axis shows the scenario IDs, and the y-axis shows the run time in seconds

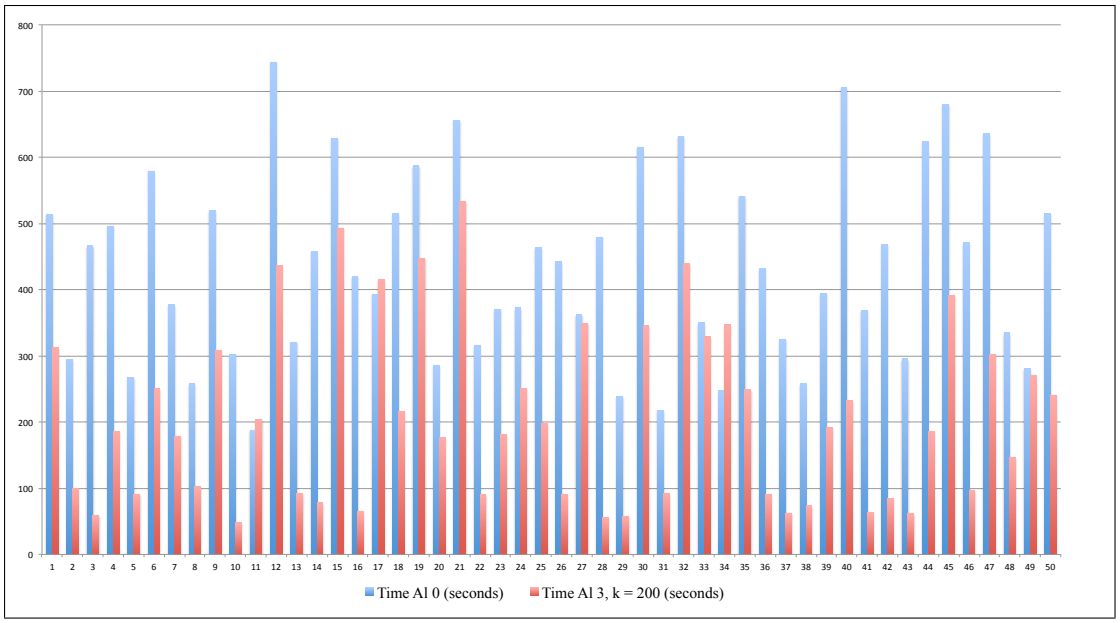

Figure 6.9: Run time comparison between Al_0 and Al_3 ($k = 200$): the x-axis shows the scenario IDs, and the y-axis shows the run time in seconds

50 scenarios. Notice that this set of experiments was run on 500 scenarios. The average time difference of using Al_3 ($k = 200$) is -58.11%.

Fig. 6.10 compares the run time of the first 50 scenarios between Al_0 and Al_3 ($k = 350$) measured in seconds. This plot shows the time reduction for the first

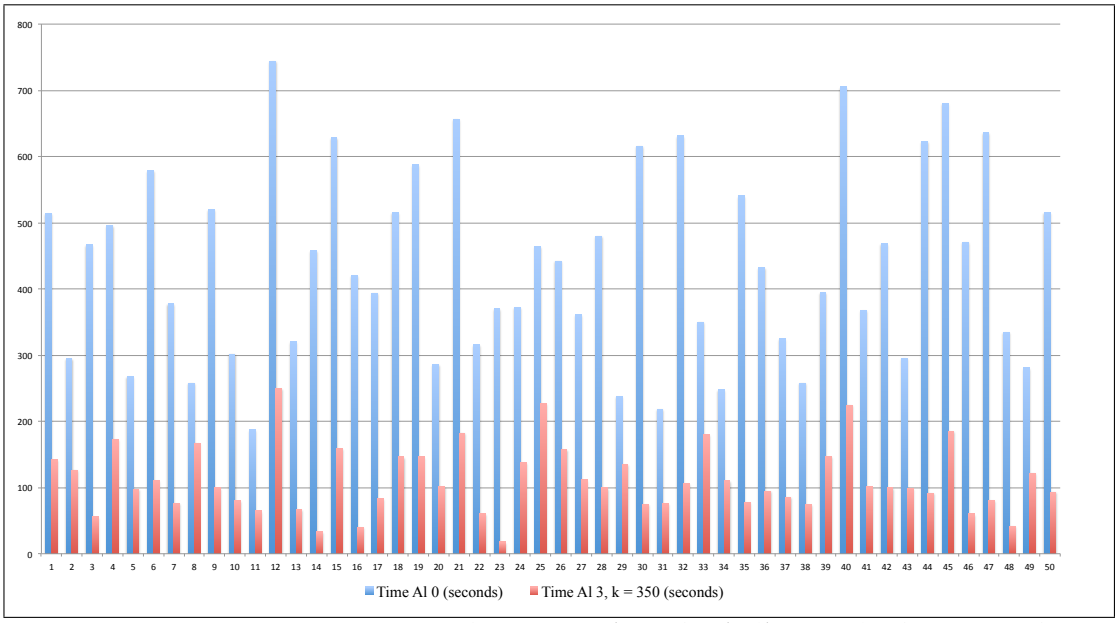

Figure 6.10: Run time comparison between Al_0 and Al_3 ($k = 350$): the x-axis shows the scenario IDs, and the y-axis shows the run time in seconds

50 scenarios. Notice that this set of experiments was run on 500 scenarios. The average time difference of using Al_3 ($k = 350$) is -74.20%.

- Peak memory use:

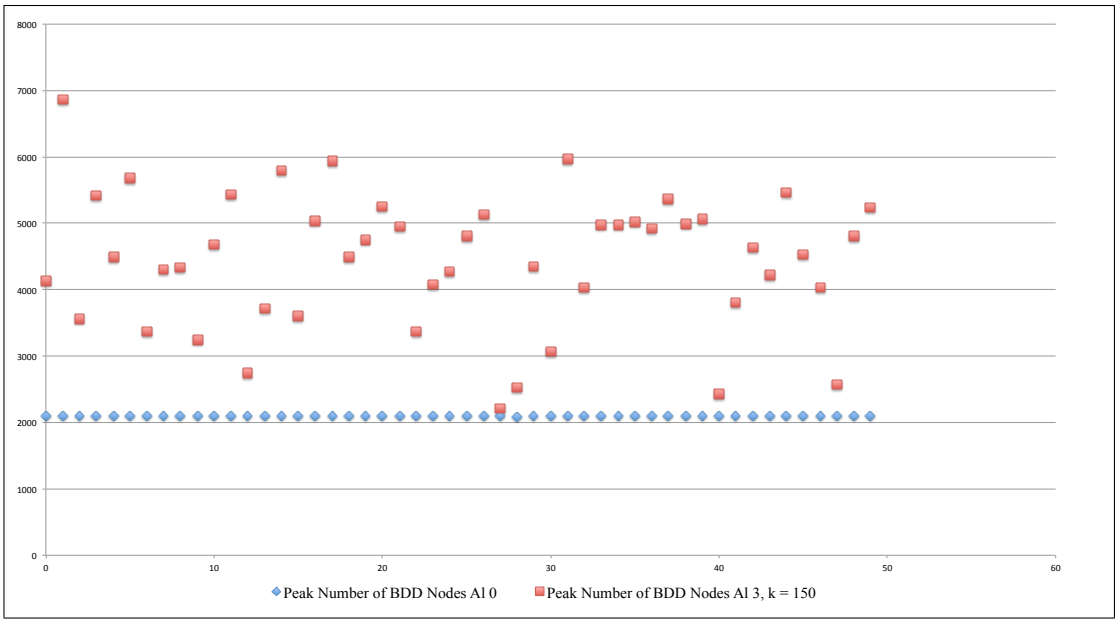

Figure 6.11: Peak number of BDD nodes in a belief state: comparison between Al_0 and Al_3 ($k = 150$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

Fig. 6.11 compares the maximum number of BDD nodes in a belief state for the first 50 scenarios. Al_3 ($k = 150$) predominantly has larger peak memory use than Al_0 does. After "resetting" to every nominal state in the global model, diagnosing the observed events in a new time window increases the size of the global belief state and the number of BDD nodes to represent the global belief state. The average increase of using Al_3 ($k = 150$) is 110.80%.

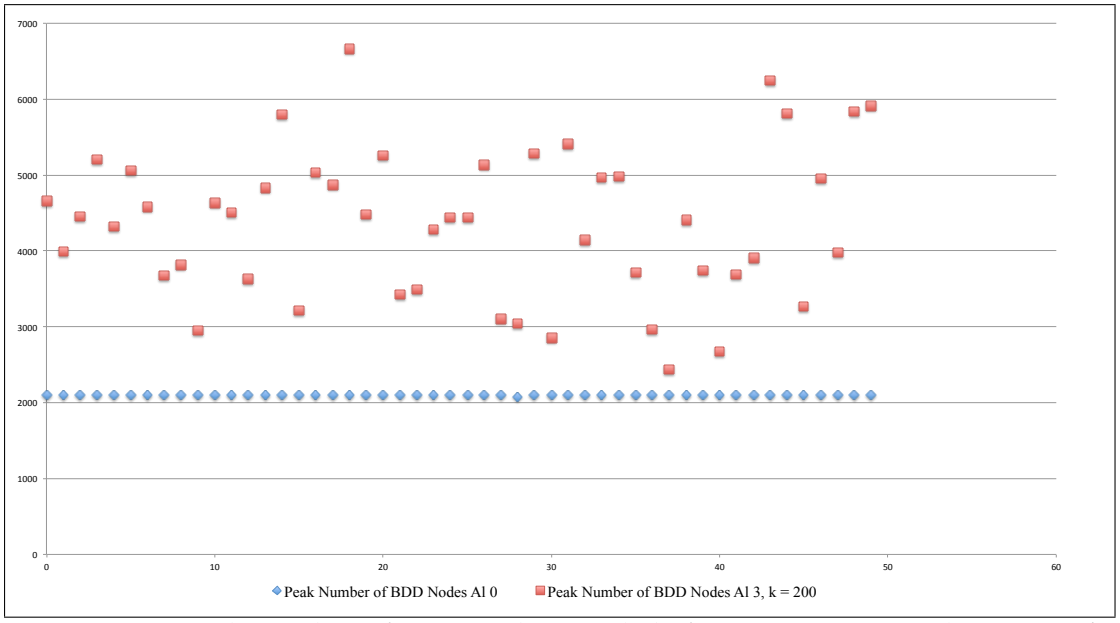

Figure 6.12: Peak number of BDD nodes in a belief state: comparison between Al_0 and Al_3 ($k = 200$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

Fig. 6.12 compares the maximum number of BDD nodes in a belief state for the first 50 scenarios. Al_3 ($k = 200$) predominantly has larger peak memory use than Al_0 does. The average increase of using Al_3 ($k = 200$) is 103.29%.

Fig. 6.13 compares the maximum number of BDD nodes in a belief state for the first 50 scenarios. Al_3 ($k = 350$) predominantly has larger peak memory use than Al_0 does. The average increase of using Al_3 ($k = 350$) is 94.04%.

- Average memory use:

  Fig. 6.14 compares the average number of BDD nodes in a belief state. The average memory use of Al_3 ($k = 150$) is consistently lower than that of Al_0. After a "reset", diagnosing observations in a new time window helps to reduce the size of the belief state from the peak. The average difference of using Al_3 ($k = 150$) is -56.63%.

  Fig. 6.15 compares the average number of BDD nodes in a belief state. The

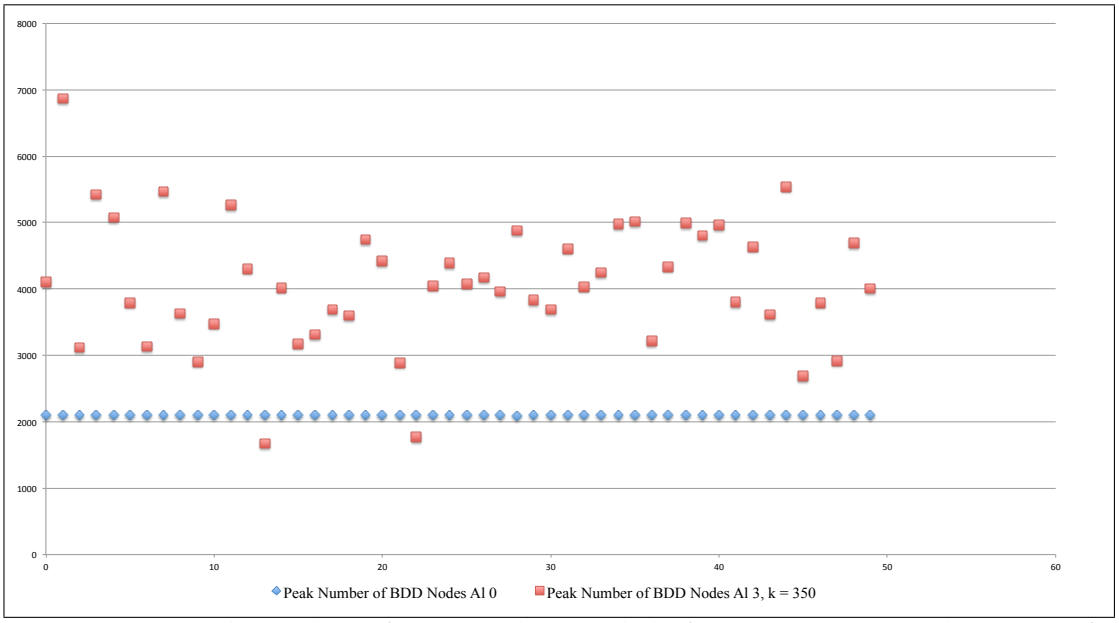

Figure 6.13: Peak number of BDD nodes in a belief state: comparison between Al_0 and Al_3 ($k = 350$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

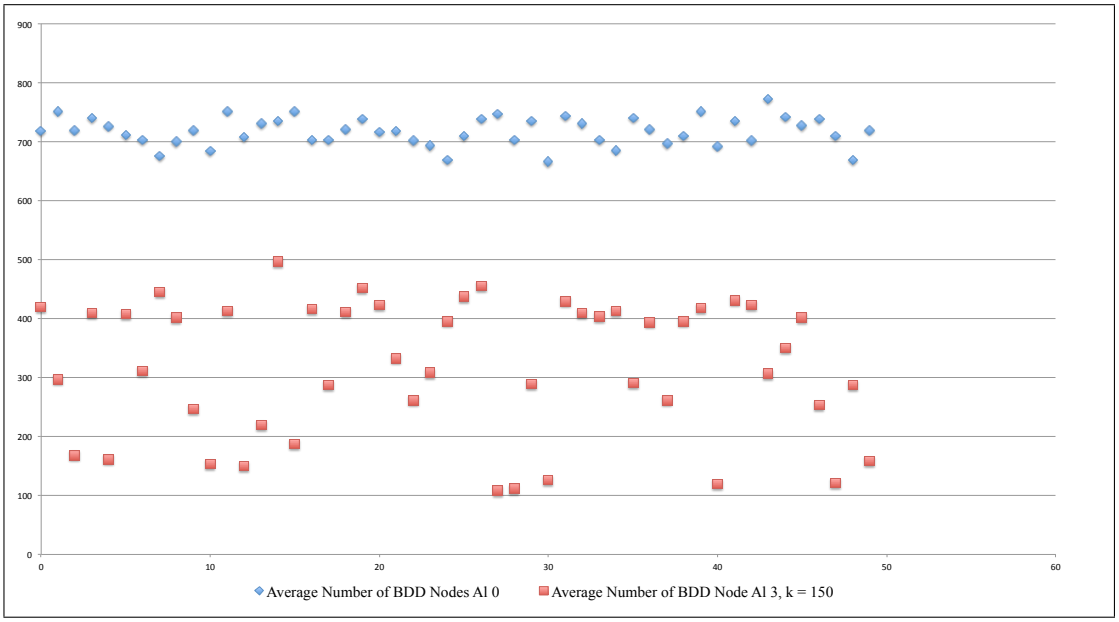

Figure 6.14: Average number of BDD nodes in a belief state: comparison between Al_0 and Al_3 ($k = 150$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

average memory use of Al_3 ($k = 200$) is consistently lower than that of Al_0. The average difference of using Al_3 ($k = 200$) is -60.17%.

Fig. 6.16 compares the average number of BDD nodes in a belief state. The

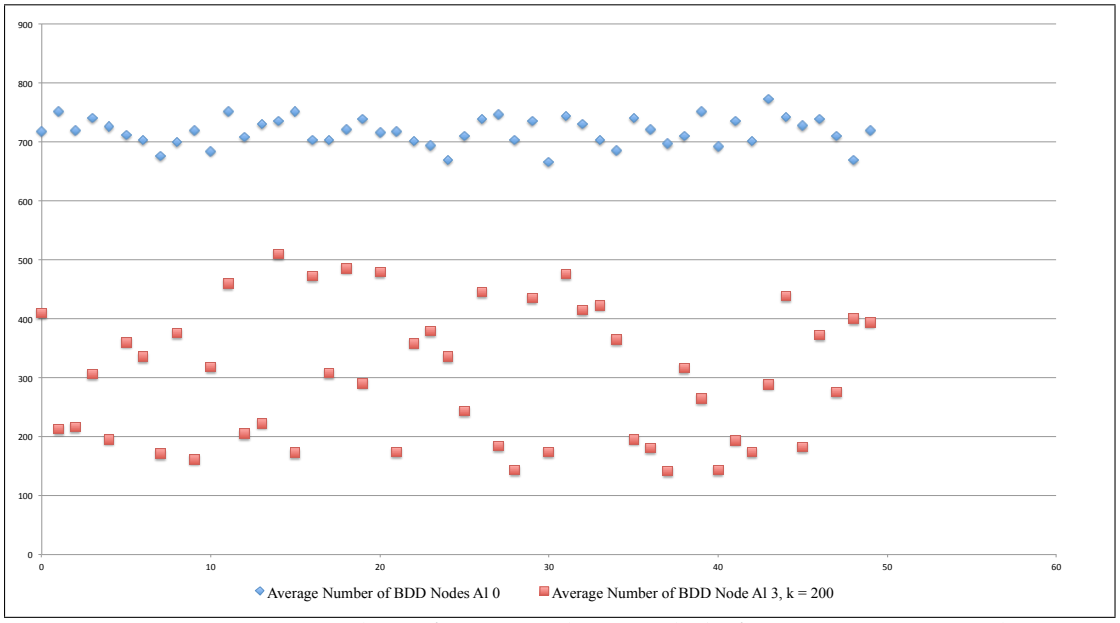

Figure 6.15: Average number of BDD nodes in a belief state: comparison between Al_0 and Al_3 ($k = 200$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

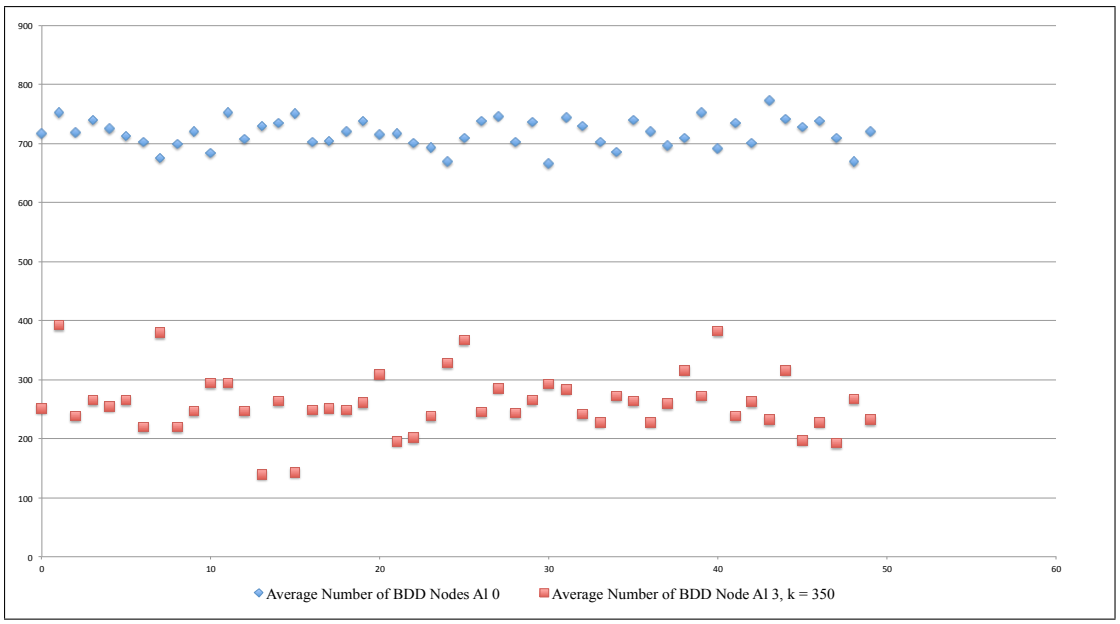

Figure 6.16: Average number of BDD nodes in a belief state: comparison between Al_0 and Al_3 ($k = 350$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

average memory use of Al_3 ($k = 350$) is consistently lower than that of Al_0. The average difference of using Al_3 ($k = 350$) is -63.65%.

- Diagnostic distance:

The average number of observations in one scenario is 1493. The average diagnostic distance using Al_3 ($k = 150$) is 428; the average diagnostic distance using Al_3 ($k = 200$) is 340; the average diagnostic distance using Al_3 ($k = 350$) is 431. In comparison, the average diagnostic distance using Al_0 is 72. It should be noted that the settings for Al_3 specify that the first half of each scenario is skipped. Although Al_3 ($k = 350$) has a larger diagnostic distance than Al_0 does, Al_3 ($k = 350$) is still able to precisely diagnose the fault.

In summary, this set of experiments indicates that Al_3 ($k = 350$) precisely diagnoses all of the 500 scenarios of a "robot" model with 14 glasses and 9 customers. Also, the run time and the average memory use are consistently reduced compared to using Al_0. This set of experiments also runs on Al_3 ($k = 150, 200, 250, 300$). However, Al_3 ($k = 150, 200, 250, 300$) are unable to precisely diagnose some of the 500 scenarios.

6.3.5 Summary for Experiments on IWAs Al_1, Al_2, and Al_3

The above experiments on IWAs Al_1, Al_2, and Al_3 examine the impact of a time window size on precision, run time, peak memory use, average memory use, and diagnostic distance. The results show that Al_3 has promising performance in comparison to the exact diagnostic algorithm Al_0. In particular, Al_3 ($k = 350$) precisely diagnoses all of the 500 scenarios of a "robot" model with 14 glasses and 9 customers. Also, the run time and the average memory use are consistently reduced compared to using Al_0. The results of Al_3 ($k = 350$) indicate that a greater value of k, i.e. fewer time windows, lead to shorter computational time, as well as lower peak and average memory use than using a smaller value of k. In conclusion, to diagnose a component-based DES model, Al_3 has good diagnostic performance. As an IWA, Al_3 can skip a specified number of observations and start diagnosis.

6.4 Experimental Model, Settings, Results, and Evaluations for Al_5

This section explains the DES model that is used in the set of experiments for TWA Al_5. The scenarios are generated using the same scenario script as described in Section 6.2.2. This work conducts experiments on Al_5. It then evaluates the performance of Al_5, and compares the experimental results with those of the exact diagnostic algorithm Al_0.

6.4.1 Experimental Model and Settings for Al_5

This work conducts experiments on "robot" model 2, which is based on the "robot" model of Section 6.3.1. Only the `Customer` component is modified, which includes three waiting states while the other components are the same as in Section 6.3.1.

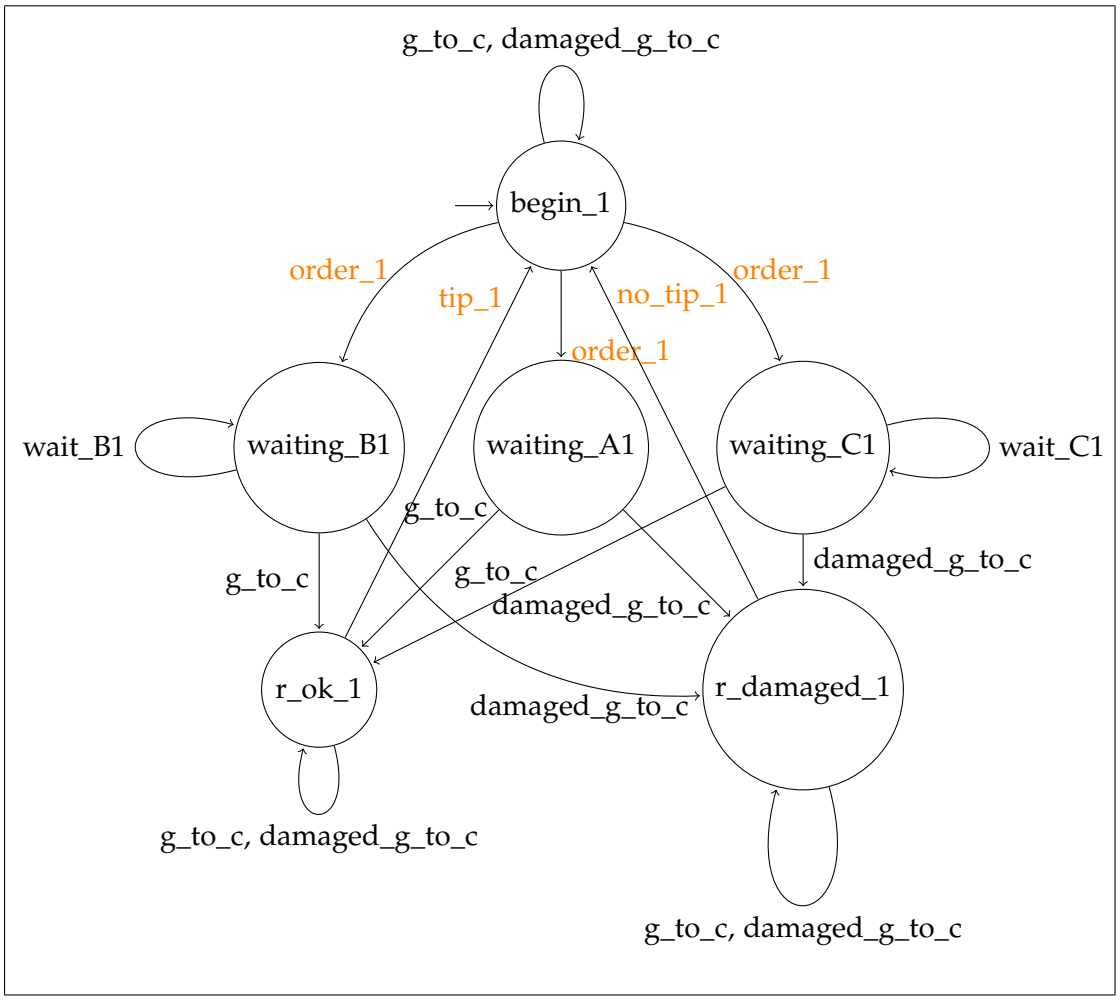

Figure 6.17: Modified Customer 1: the state "r_ok_1" means that Customer 1 has received a normal glass, and the state "r_damaged_1" means that Customer 1 has received a damaged glass; the event "g_to_c" means giving a normal glass to a customer, and the event "damaged_g_to_c" means giving a damaged glass to a customer; "order_1", "tip_1", and "no_tip_1" are the observable events while the other events are unobservable; the loops with "g_to_c" and "damaged_g_to_c" are used to model the sequencing of multiple orders and glasses.

Fig. 6.17 and Fig. 6.18 illustrate two instances of the modified `Customer` component. The abstract states in the `Customer` component are described as follows.

- {`begin_i`};

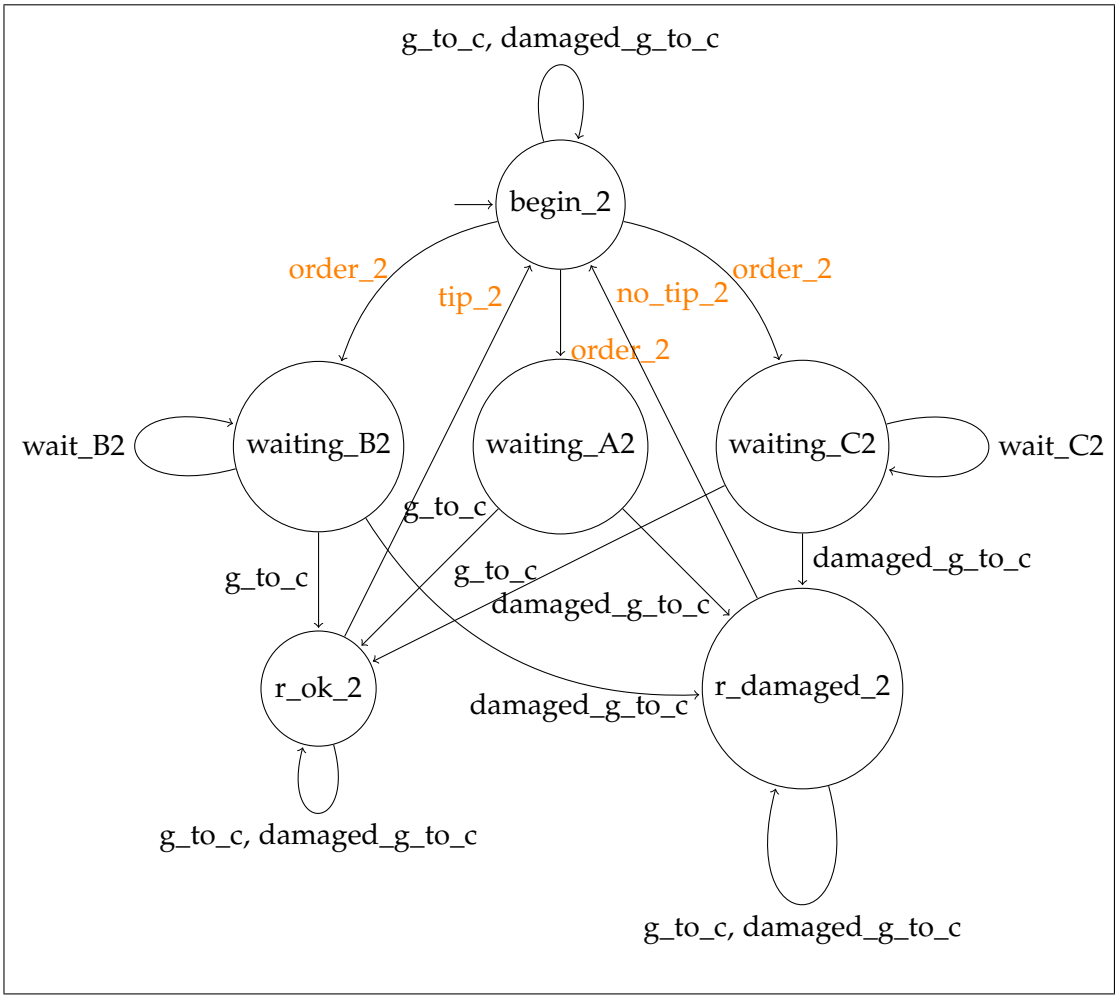

Figure 6.18: Modified Customer 2: the state "r_ok_2" means that Customer 2 has received a normal glass, and the state "r_damaged_2" means that Customer 2 has received a damaged glass; the event "g_to_c" means giving a normal glass to a customer, and the event "damaged_g_to_c" means giving a damaged glass to a customer; "order_2", "tip_2", and "no_tip_2" are the observable events while the other events are unobservable; the loops with "g_to_c" and "damaged_g_to_c" are used to model the sequencing of multiple orders and glasses.

- {`waiting_Ai`, `waiting_Bi`, `waiting_Ci`};
- {`r_ok_i`};
- {`r_damaged_i`} where 'i' is the index for each `Customer` component.

The abstract states in the other components are formed by the individual states. This work conducts three sets of experiments on "robot" model 2 with 14 glasses, 9 customers, a fault monitor, and an order monitor. Using the scenario script described in Section 6.2, this work generates a set of 500 scenarios, and runs the experiments.

Recall that Chapter 5 denotes a time window size as k. The settings of each set of experiments are described as follows.

- Experiment 1: the exact diagnostic algorithm Al_0;
- Experiment 2: Al_5 where $k = 25$;
- Experiment 3: Al_5 where $k = 50$.

Section 6.4.2 will compare the results of experiment 1 with those of experiment 2. Section 6.4.3 will compare the results of experiment 1 with those of experiment 3.

6.4.2 Comparison between Al_0 and Al_5 ($k = 25$)

This work compares the experimental results of Al_5 ($k = 25$) with those of Al_0, and evaluates the performance of Al_5 ($k = 25$) in five aspects as specified in Section 6.3.1.

- Precision:

  The exact diagnostic algorithm Al_0 diagnoses all of the 500 scenarios as faulty. Al_5 ($k = 25$) also diagnoses the fault in each of the 500 scenarios.

- Run time:

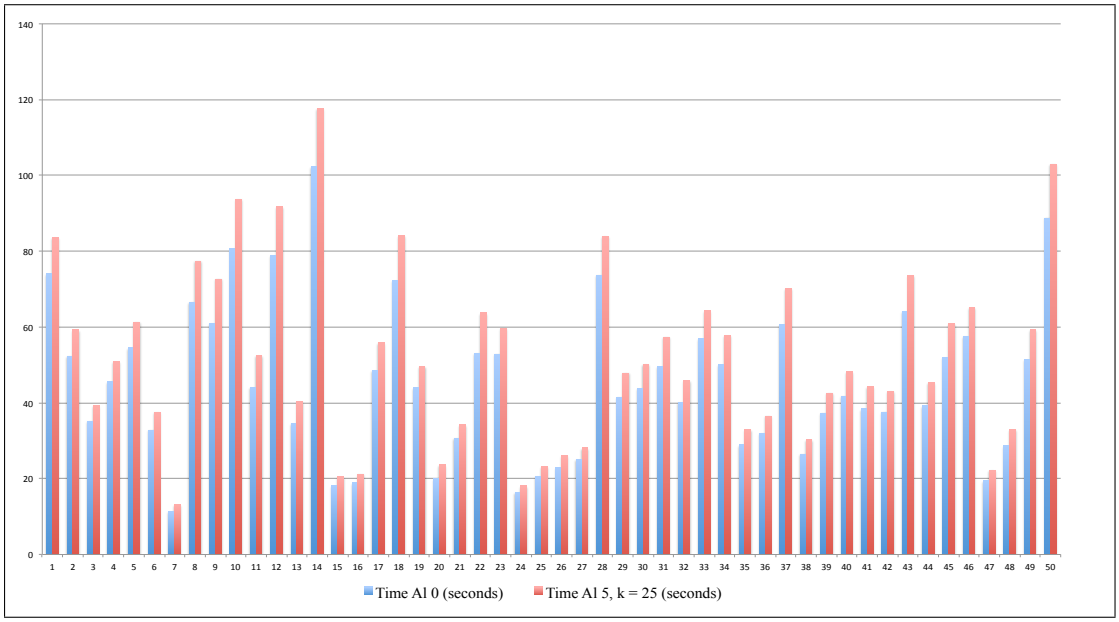

Figure 6.19: Run time comparison between Al_0 and Al_5 ($k = 25$): the x-axis shows the scenario IDs, and the y-axis shows the run time in seconds

Fig. 6.19 compares the run time of the first 50 scenarios between Al_0 and Al_5 ($k = 25$) measured in seconds. This plot shows the time increase for the first

50 scenarios. Notice that this set of experiments was run on 500 scenarios. The average time increase of using Al_5 ($k = 25$) is 16.00%.

- Peak memory use:

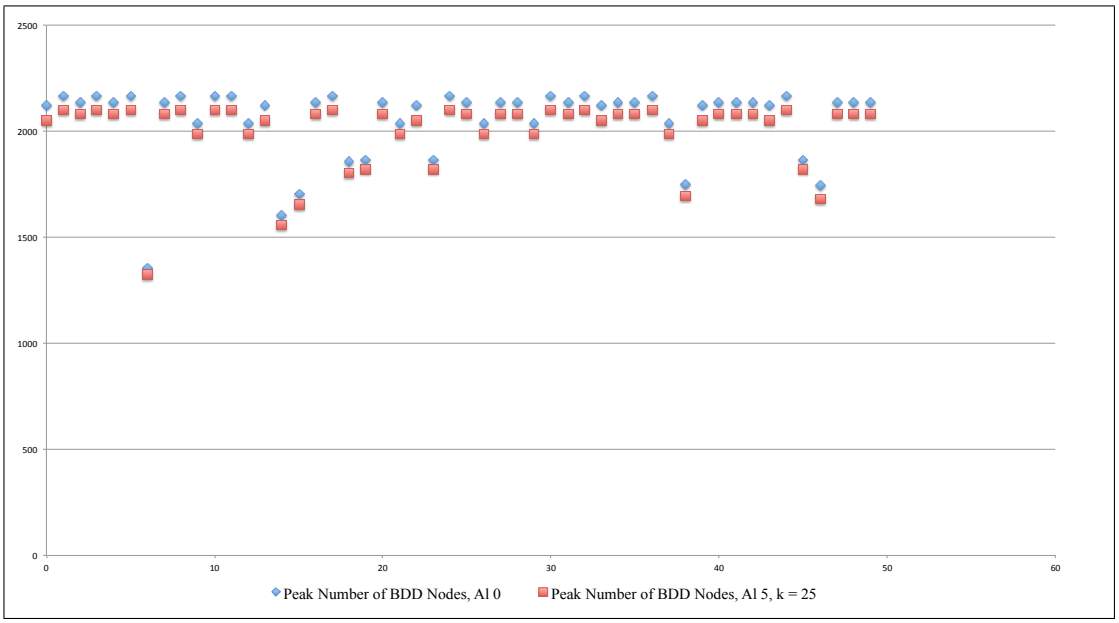

Figure 6.20: Peak number of BDD nodes in a belief state: comparison between Al_0 and Al_5 ($k = 25$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

Fig. 6.20 compares the maximum number of BDD nodes in a belief state for the first 50 scenarios. Al_5 ($k = 25$) consistently has lower peak memory use than Al_0 does. This is because Al_5 ($k = 25$) uses abstract belief states in the global model, which reduces the size of the belief state compared to using Al_0. The average difference of using Al_5 ($k = 25$) is -3.05%.

- Average memory use:

  Fig. 6.21 compares the average number of BDD nodes in a belief state. The average memory use of Al_5 ($k = 25$) is consistently lower than that of Al_0. This result is consistent with that of the peak memory use. The average difference of using Al_5 ($k = 25$) is -9.19%.

- Diagnostic distance:

  The diagnostic distance of using Al_5 ($k = 25$) is consistently the same as that of using Al_0. The average diagnostic distance is 74. Notice that the average number of observations in one scenario is 1498.

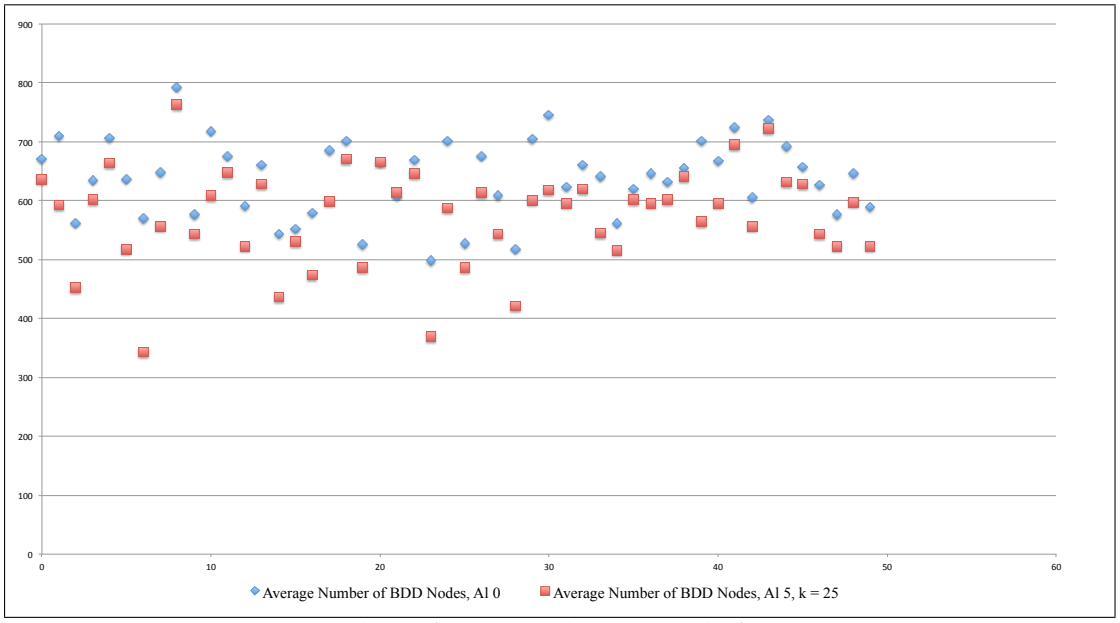

Figure 6.21: Average number of BDD nodes in a belief state: comparison between Al_0 and Al_5 ($k = 25$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

In summary, this set of experiments indicates that Al_5 ($k = 25$) precisely diagnoses "robot" model 2 with 14 glasses and 9 customers. Although the computational time is increased compared to using Al_0, both peak and average memory use are reduced. Also, the selected abstract states lead to a precise TWA Al_5 ($k = 25$), i.e. Al_5 ($k = 25$) is precise to diagnose "robot" model 2. Therefore, the results of this set of experiments show that the benefits of TWA Al_5 are consistent with the theory presented in Chapter 5.

6.4.3 Comparison between Al_0 and Al_5 ($k = 50$)

This work compares the experimental results of Al_5 ($k = 50$) with those of Al_0, and evaluates the performance of Al_5 ($k = 50$) in five aspects as specified in Section 6.3.1.

- Precision:

  The exact diagnostic algorithm Al_0 diagnoses all of the 500 scenarios as faulty. Al_5 ($k = 50$) also diagnoses the fault in each of the 500 scenarios.

- Run time:

  Fig. 6.22 compares the run time of the first 50 scenarios between Al_0 and Al_5 ($k = 50$) measured in seconds. This plot shows the time increase for the first 50 scenarios. Notice that this set of experiments was run on 500 scenarios. The average time increase of using Al_5 ($k = 50$) is 14.15%.

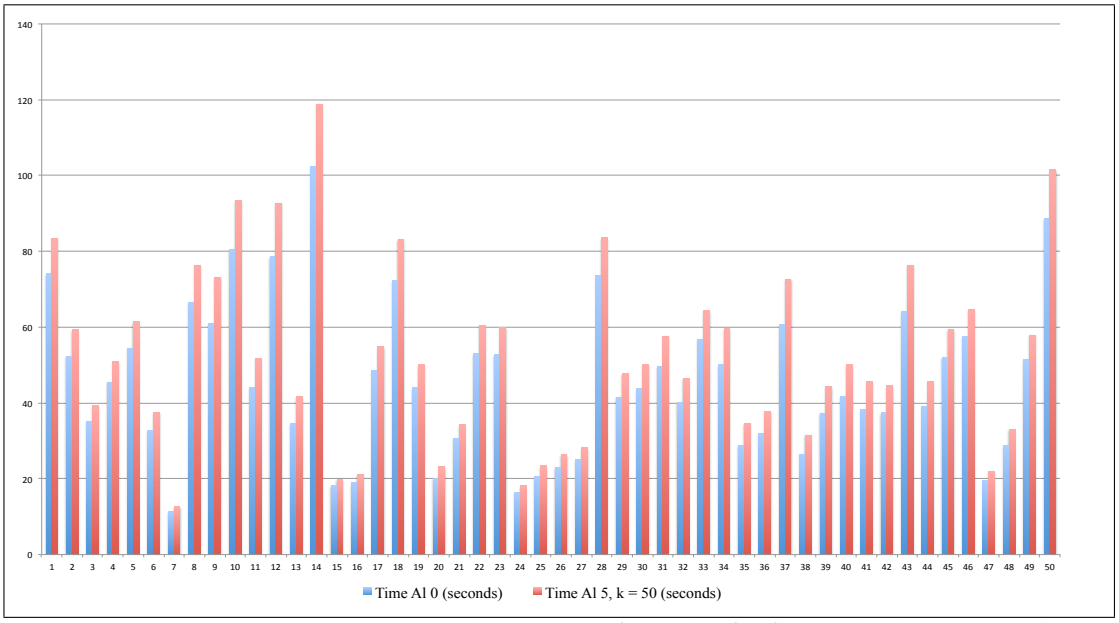

Figure 6.22: Run time comparison between Al_0 and Al_5 ($k = 50$): the x-axis shows the scenario IDs, and the y-axis shows the run time in seconds

- Peak memory use:

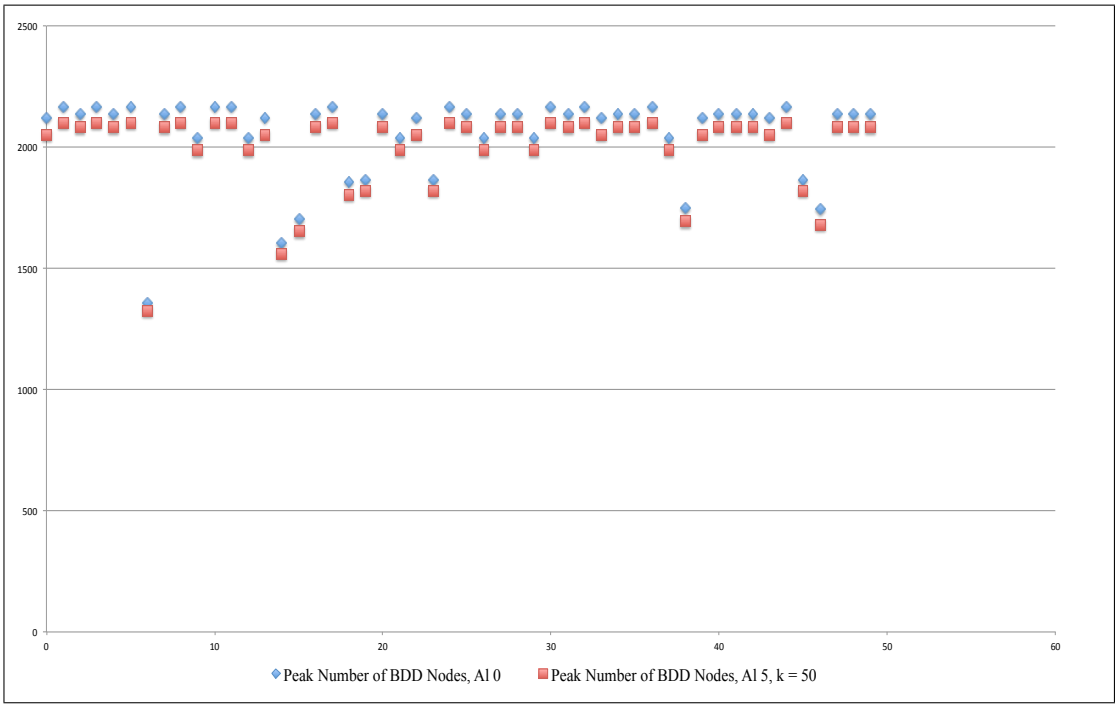

Figure 6.23: Peak number of BDD nodes in a belief state: comparison between Al_0 and Al_5 ($k = 50$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

Fig. 6.23 compares the maximum number of BDD nodes in a belief state for the

first 50 scenarios. Al_5 ($k = 50$) consistently has lower peak memory use than Al_0 does. This is because Al_5 ($k = 50$) uses abstract belief states in the global model, which reduces the size of the belief state compared to using Al_0. The average difference of using Al_5 ($k = 50$) is -3.05%.

- Average memory use:

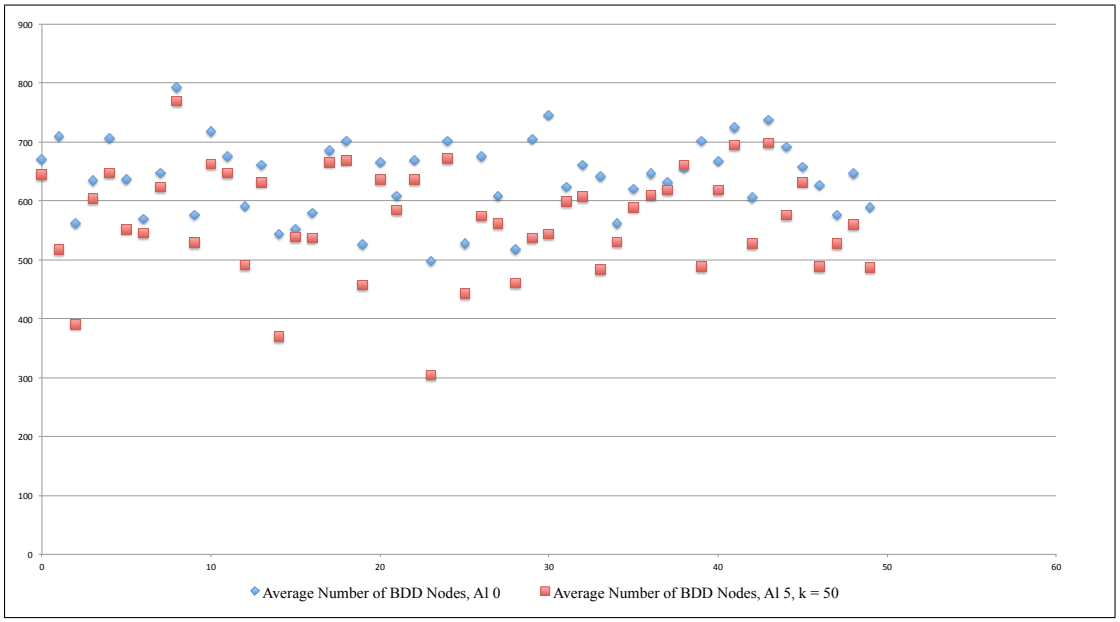

Figure 6.24: Average number of BDD nodes in a belief state: comparison between Al_0 and Al_5 ($k = 50$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

Fig. 6.24 compares the average number of BDD nodes in a belief state. The average memory use of Al_5 ($k = 50$) is consistently lower than that of Al_0. This result is consistent with the peak memory use. The average difference of using Al_5 ($k = 50$) is -9.63%.

- Diagnostic distance:

  The diagnostic distance of using Al_5 ($k = 50$) is consistently the same as that of using Al_0. The average diagnostic distance is 74. Notice that the average number of observations in one scenario is 1498.

In summary, this set of experiments indicates that Al_5 ($k = 50$) precisely diagnoses "robot" model 2 with 14 glasses and 9 customers. Although the computational time is increased compared to using Al_0, both peak and average memory use are reduced. Also, the selected abstract states lead to a precise TWA Al_5 ($k = 50$), i.e. Al_5 ($k = 50$) is precise to diagnose "robot" model 2. Therefore, the results of this

set of experiments show that the benefits of TWA Al_5 are consistent with the theory presented in Chapter 5.

6.5 Experimental Model, Settings, Results, and Evaluations for Al_6

This section explains the DES model that is used in the set of experiments for Al_6. This work conducts experiments on TWA Al_6. It then evaluates the performance of Al_6, and compares the experimental results with those of the exact diagnostic algorithm Al_0.

6.5.1 Experimental Model and Settings for Al_6

This section conducts experiments on "robot" model 3, which is based on the "robot" model as specified in Section 6.4.1. Only the `Customer` component is modified, which includes an additional state, called `no_order`.

Fig. 6.25 and Fig. 6.26 illustrate two instances of the modified `Customer` component. The abstract states in the `Customer` component are described as follows.

- {`begin_i`, `no_order_i`};
- {`waiting_Ai`, `waiting_Bi`, `waiting_Ci`};
- {`r_ok_i`};
- {`r_damaged_i`} where 'i' is the index for each `Customer` component.

The other components are the same as in Section 6.3.1, and the abstract states in the other components are formed by the individual states. This work conducts experiments on "robot" model 3 with 14 glasses, 9 customers, a fault monitor, and an order monitor. This work generates a set of 500 scenarios based on "robot" model 3, and runs the experiments. Recall that Chapter 5 denotes a time window size as k. The settings of each experiment are described as follows.

- Experiment 1: the exact diagnostic algorithm Al_0;
- Experiment 2: Al_5 where $k = 30$;
- Experiment 3: Al_5 where $k = 60$;
- Experiment 4: Al_6 where $k = 30$;
- Experiment 5: Al_6 where $k = 60$.

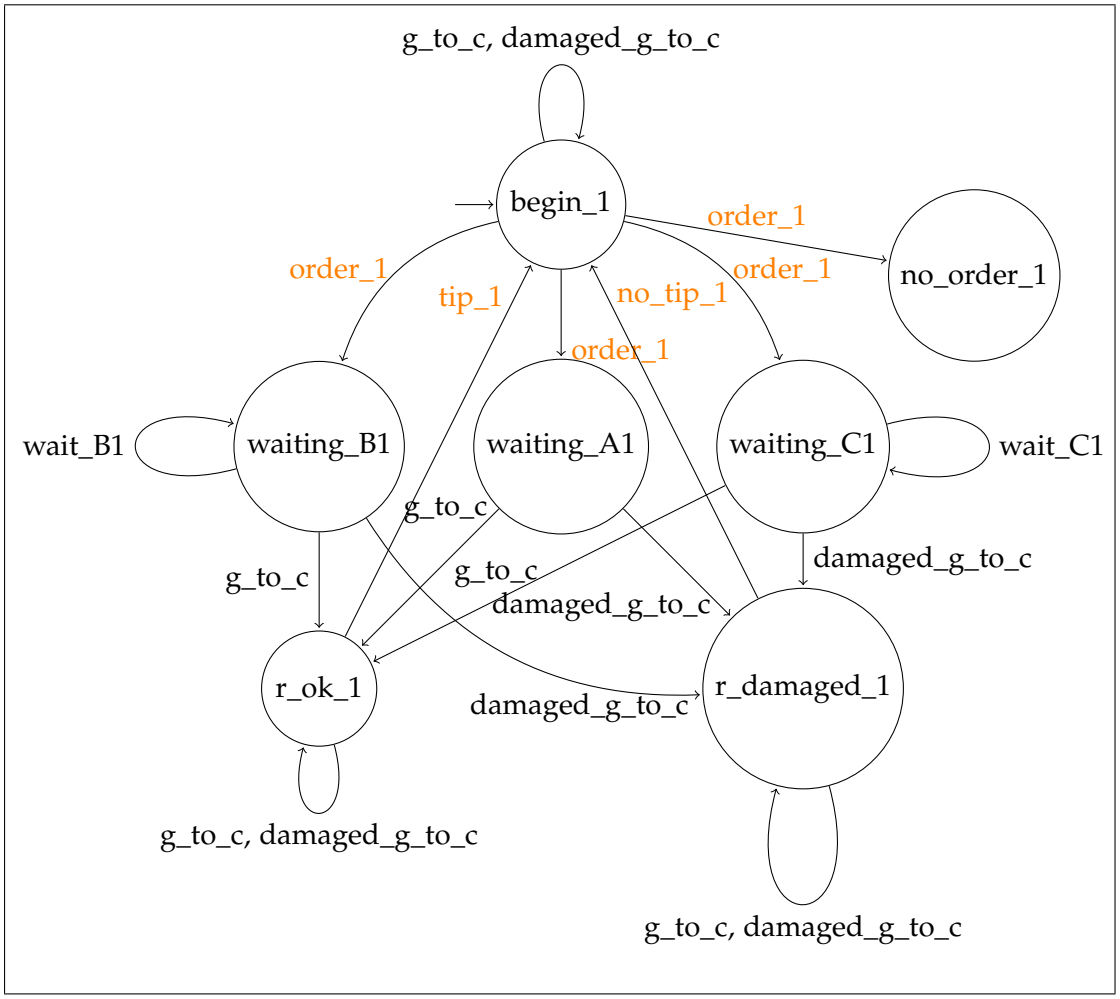

Figure 6.25: Modified Customer 1: the state "r_ok_1" means that Customer 1 has received a normal glass, and the state "r_damaged_1" means that Customer 1 has received a damaged glass; the event "g_to_c" means giving a normal glass to a customer, and the event "damaged_g_to_c" means giving a damaged glass to a customer; "order_1", "tip_1", and "no_tip_1" are the observable events while the other events are unobservable; the loops with "g_to_c" and "damaged_g_to_c" are used to model the sequencing of multiple orders and glasses.

Section 6.5.2 will compare the results of experiment 1 with those of experiment 2. Section 6.5.3 will compare the results of experiment 1 with those of experiment 3. Section 6.5.4 will compare the results of experiment 1 with those of experiment 4. Section 6.5.5 will compare the results of experiment 1 with those of experiment 5.

6.5.2 Comparison between Al_0 and Al_5 ($k = 30$)

This work compares the experimental results of Al_5 ($k = 30$) with those of Al_0, and evaluates the performance of Al_5 ($k = 30$) in four aspects, i.e. precision, run time,

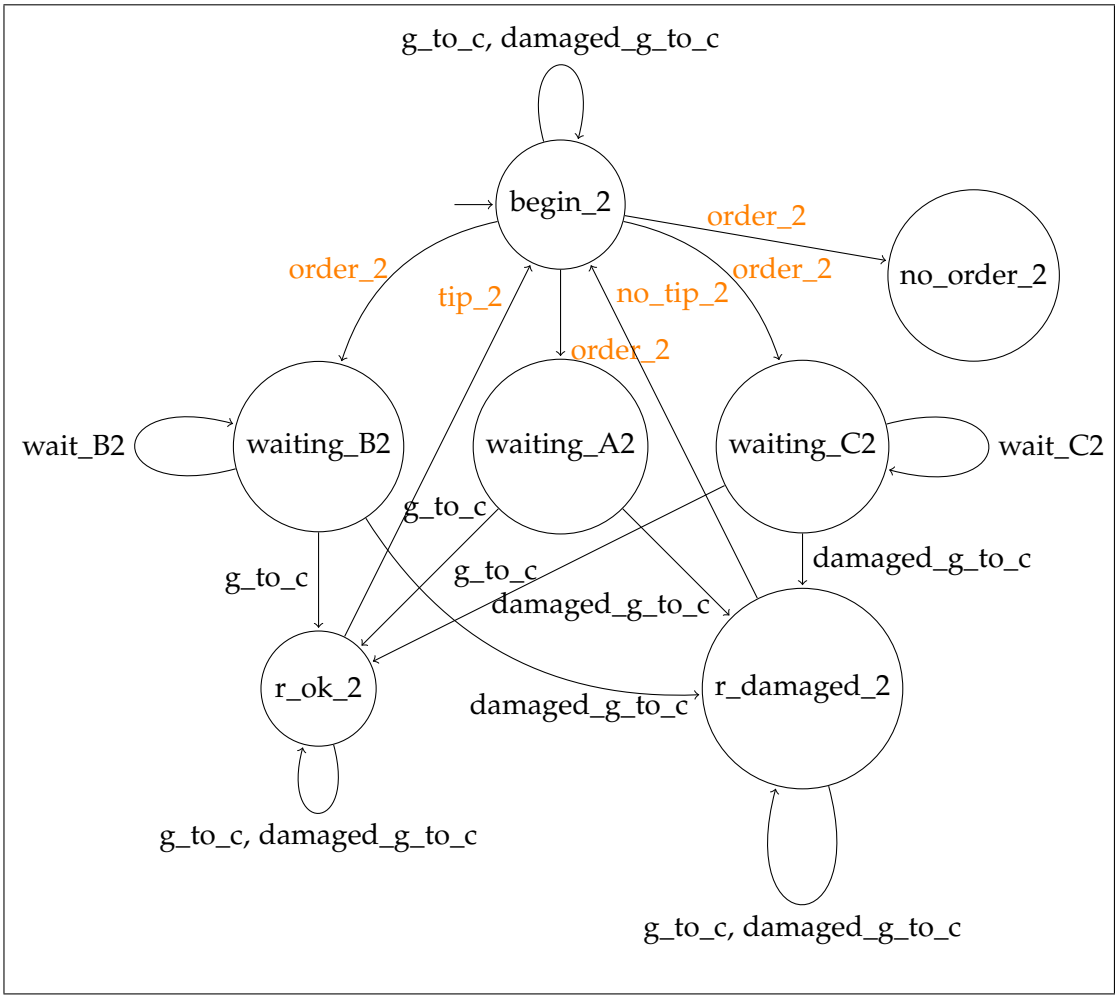

Figure 6.26: Modified Customer 2: the state "r_ok_2" means that Customer 2 has received a normal glass, and the state "r_damaged_2" means that Customer 2 has received a damaged glass; the event "g_to_c" means giving a normal glass to a customer, and the event "damaged_g_to_c" means giving a damaged glass to a customer; "order_2", "tip_2", and "no_tip_2" are the observable events while the other events are unobservable; the loops with "g_to_c" and "damaged_g_to_c" are used to model the sequencing of multiple orders and glasses.

peak memory use, and average memory use.

- Precision:

  The exact diagnostic algorithm Al_0 diagnoses all of the 500 scenarios as faulty. However, Al_5 ($k = 30$) cannot precisely diagnose every fault in the 500 scenarios. As a result, it is not possible to evaluate Al_5 ($k = 30$) in terms of diagnostic distance.

- Run time:

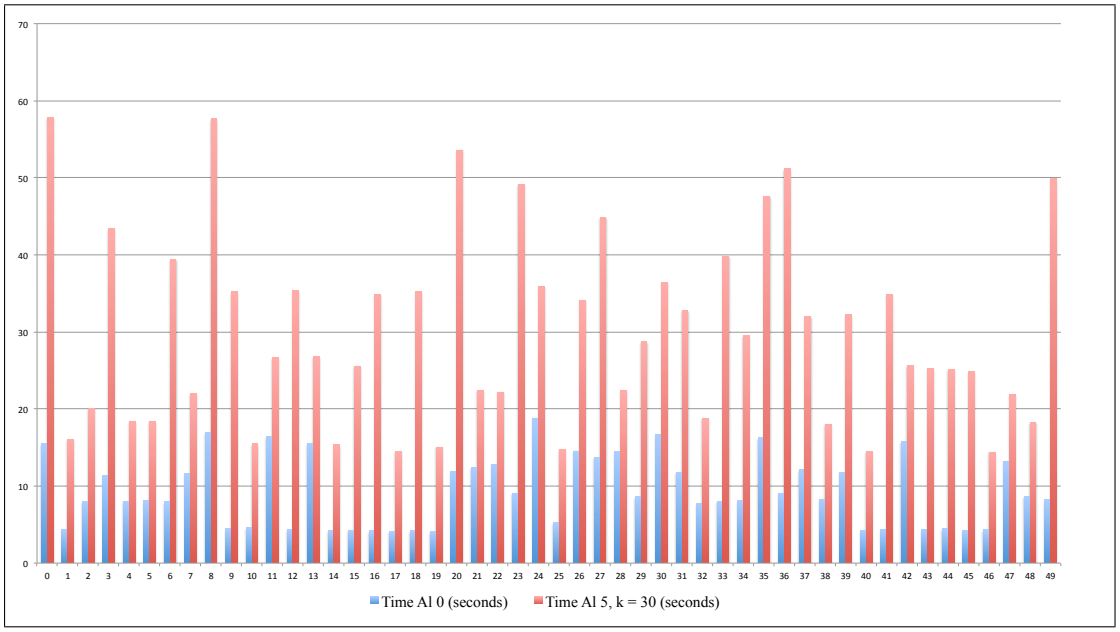

Figure 6.27: Rum time comparison between Al_0 and Al_5 ($k = 30$): the x-axis shows the scenario IDs, and the y-axis shows the run time in seconds

Fig. 6.27 compares the run time of the first 50 scenarios between Al_0 and Al_5 ($k = 30$) measured in seconds. This plot shows the time increase for the first 50 scenarios. This is because Al_5 ($k = 30$) may analyse all observations in a scenario, but is unable to diagnose the fault. Notice that this set of experiments was run on 500 scenarios. The average time increase of using Al_5 ($k = 30$) is 336.52%.

- Peak memory use:

  Fig. 6.28 compares the maximum number of BDD nodes in a belief state for the first 50 scenarios. Al_5 ($k = 30$) predominantly has higher peak memory use than Al_0 does. This is because Al_5 ($k = 30$) may analyse all observations in a scenario, but is unable to diagnose the fault. Notice that this set of experiments was run on 500 scenarios. The average increase of using Al_5 ($k = 30$) is 85.05%.

- Average memory use:

  Fig. 6.29 compares the average number of BDD nodes in a belief state. Al_5 ($k = 30$) predominantly has higher peak memory use than Al_0 does. This is because Al_5 ($k = 30$) may analyse all observations in a scenario, but is unable to diagnose the fault. Notice that this set of experiments was run on 500 scenarios. The average increase of using Al_5 ($k = 30$) is 55.00%.

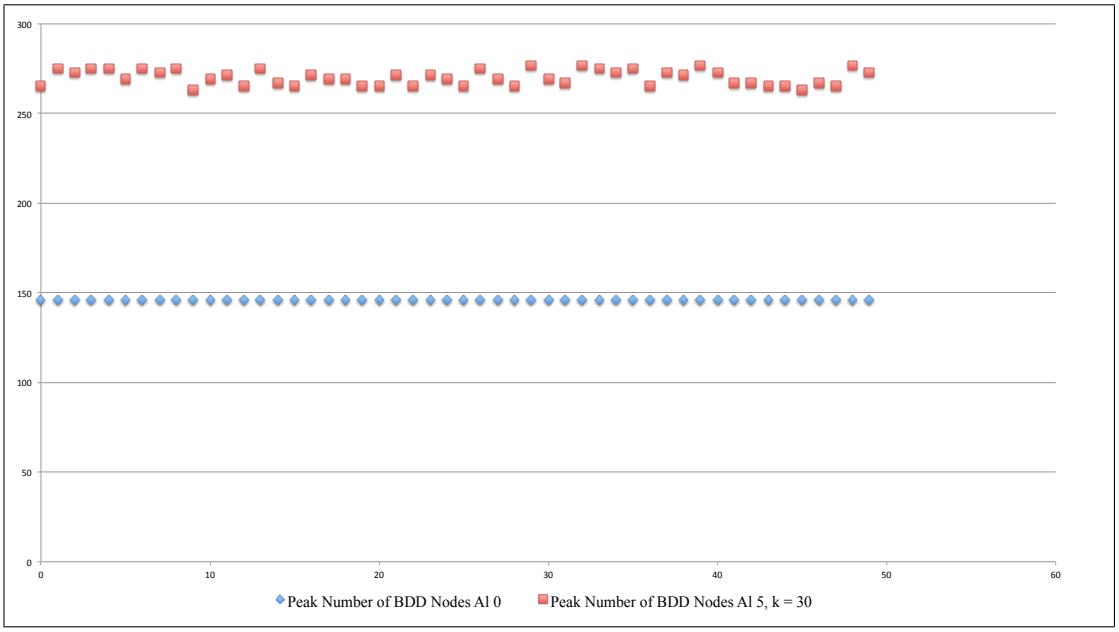

Figure 6.28: Peak number of BDD nodes in a belief state: comparison between Al_0 and Al_5 ($k = 30$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

In summary, this set of experiments indicates that Al_5 ($k = 30$) is unable to precisely diagnose "robot" model 3 with 14 glass and 9 customers. Thus, it is necessary for Section 6.5.4 and Section 6.5.5 to evaluate the performance of Al_6 ($k = 30, 60$) w.r.t. "robot" model 3.

6.5.3 Comparison between Al_0 and Al_5 ($k = 60$)

This work compares the experimental results of Al_5 ($k = 60$) with those of Al_0, and evaluates the performance of Al_5 ($k = 60$) in four aspects, i.e. precision, run time, peak memory use, and average memory use.

- Precision:

  The exact diagnostic algorithm Al_0 diagnoses all of the 500 scenarios as faulty. However, Al_5 ($k = 60$) cannot precisely diagnose every fault in the 500 scenarios. As a result, it is not possible to evaluate Al_5 ($k = 60$) in terms of diagnostic distance.

- Run time:

  Fig. 6.30 compares the run time of the first 50 scenarios between Al_0 and Al_5 ($k = 60$) measured in seconds. This plot shows the time increase for the first 50 scenarios. This is because Al_5 ($k = 60$) may analyse all observations in a

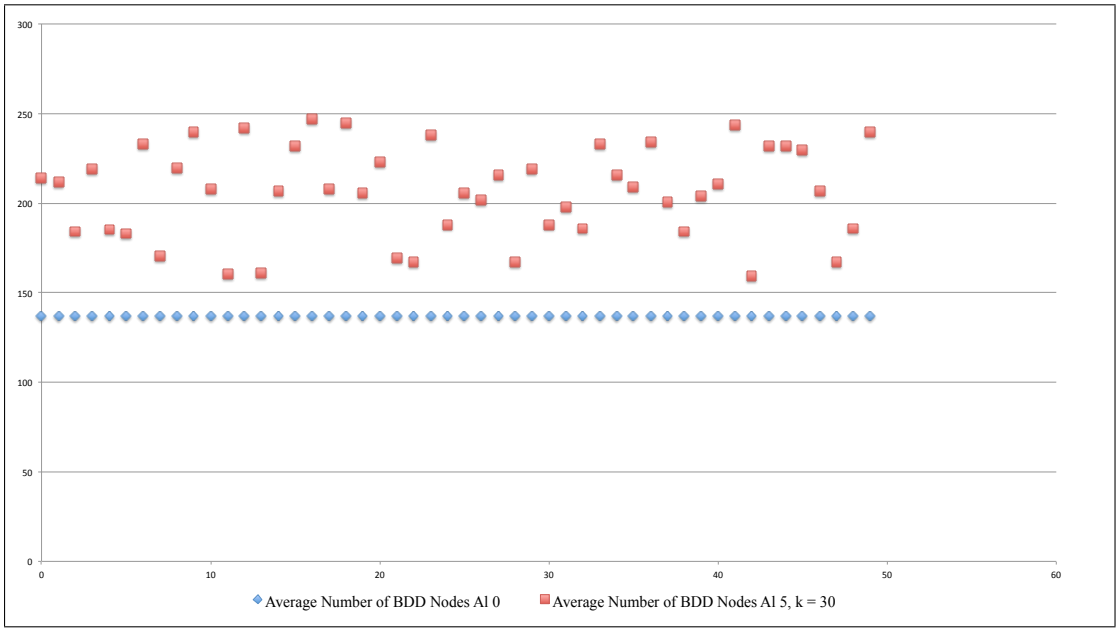

Figure 6.29: Average number of BDD nodes in a belief state: comparison between Al_0 and Al_5 ($k = 30$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

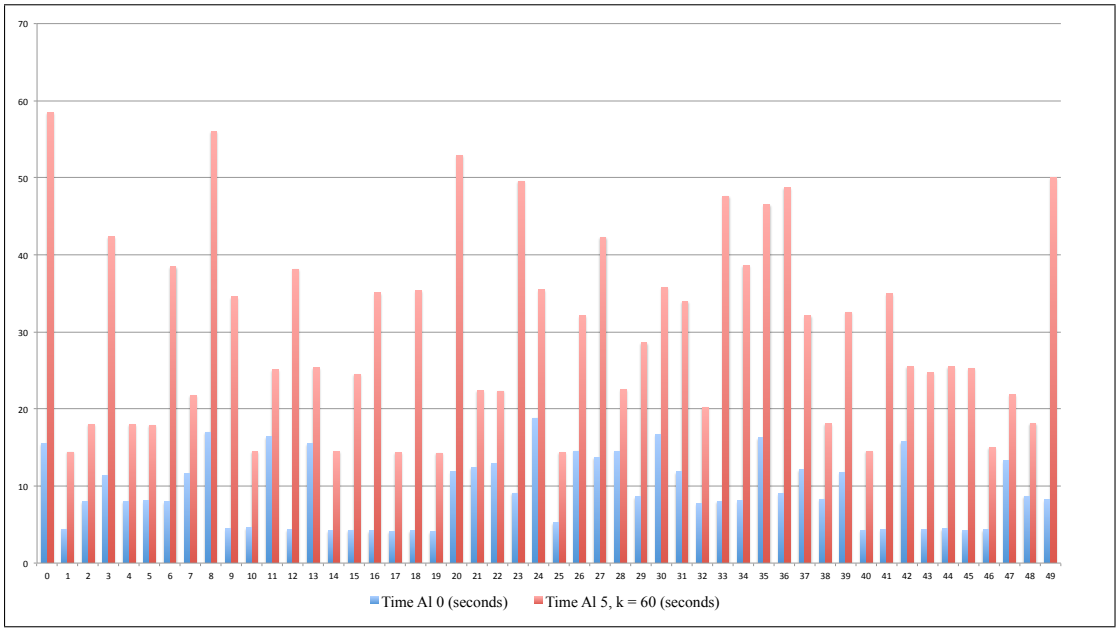

Figure 6.30: Rum time comparison between Al_0 and Al_5 ($k = 60$): the x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes

scenario, but is unable to diagnose the fault. Notice that this set of experiments was run on 500 scenarios. The average time increase of using Al_5 ($k = 60$) is 336.52%.

- Peak memory use:

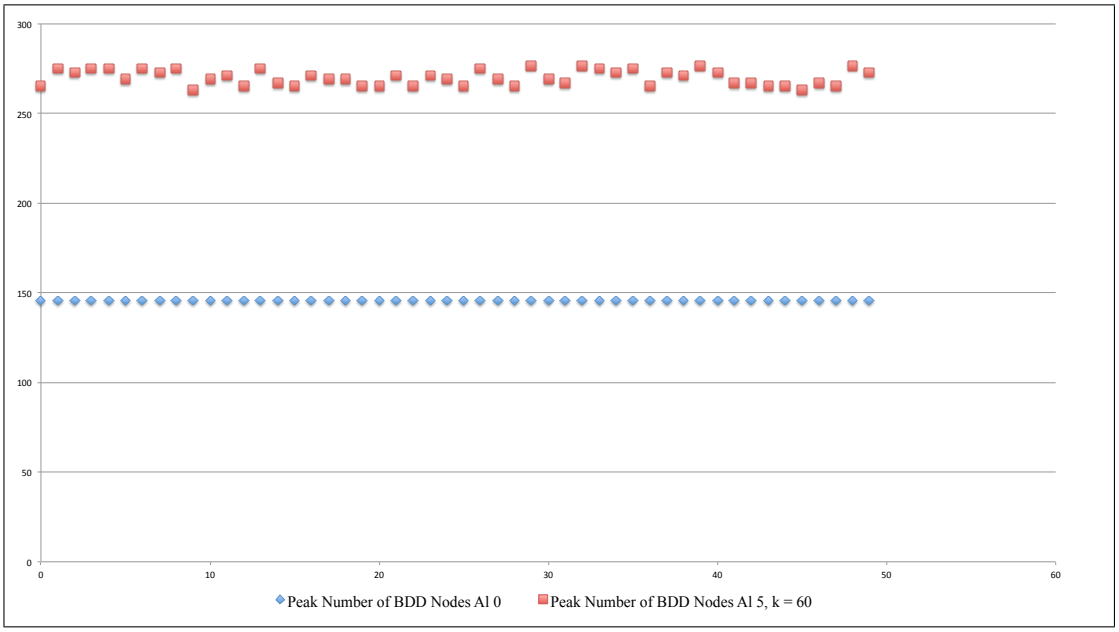

Figure 6.31: Peak number of BDD nodes in a belief state: comparison between Al_0 and Al_5 ($k = 60$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

Fig. 6.31 compares the maximum number of BDD nodes in a belief state for the first 50 scenarios. Al_5 ($k = 60$) predominantly has higher peak memory use than Al_0 does. This is because Al_5 ($k = 60$) may analyse all observations in a scenario, but is unable to diagnose the fault. Notice that this set of experiments was run on 500 scenarios. The average increase of using Al_5 ($k = 60$) is 85.05%.

- Average memory use:

Fig. 6.32 compares the average number of BDD nodes in a belief state. Al_5 ($k = 60$) predominantly has higher peak memory use than Al_0 does. This is because Al_5 ($k = 60$) may analyse all observations in a scenario, but is unable to diagnose the fault. Notice that this set of experiments was run on 500 scenarios. The average increase of using Al_5 ($k = 60$) is 50.74%.

In summary, this set of experiments indicates that Al_5 ($k = 60$) is unable to precisely diagnose "robot" model 3 with 14 glass and 9 customers. Thus, it is necessary for Section 6.5.4 and Section 6.5.5 to evaluate the performance of Al_6 ($k = 30, 60$) w.r.t. "robot" model 3.

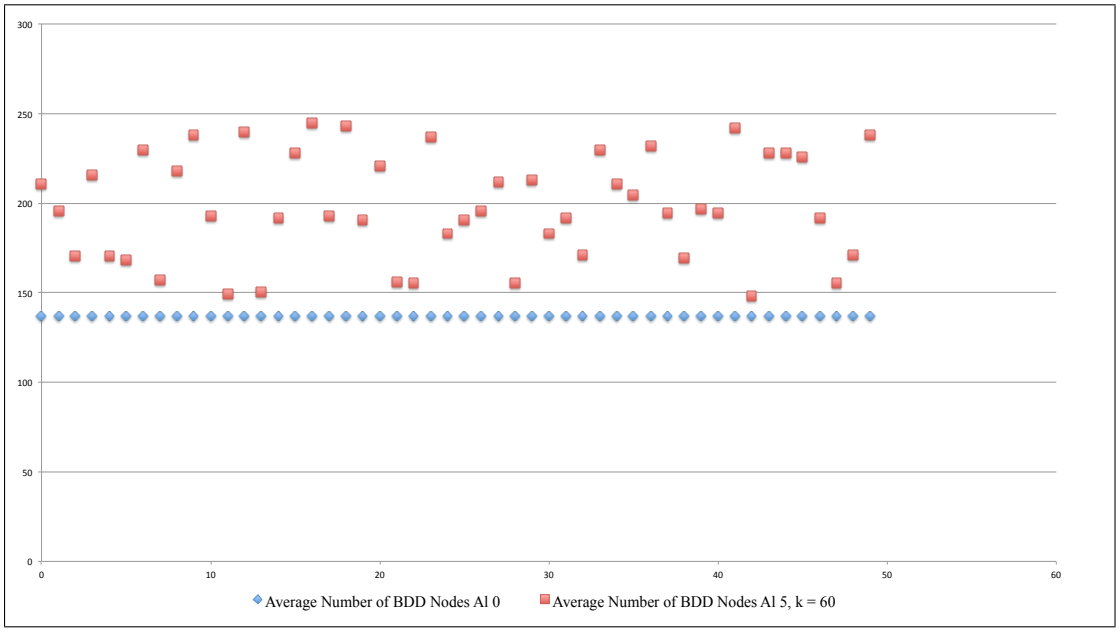

Figure 6.32: Average number of BDD nodes in a belief state: comparison between Al_0 and Al_5 ($k = 60$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

6.5.4 Comparison between Al_0 and Al_6 ($k = 30$)

This work compares the experimental results of Al_6 ($k = 30$) with those of Al_0, and evaluates the performance of Al_6 ($k = 30$) in five aspects as specified in Section 6.3.1.

- Precision:

  The exact diagnostic algorithm Al_0 diagnoses all of the 500 scenarios as faulty. Al_6 ($k = 30$) also precisely diagnoses the fault in each of the 500 scenarios.

- Run time:

  Fig. 6.33 compares the run time of the first 50 scenarios between Al_0 and Al_6 ($k = 30$) measured in seconds. This plot shows the time increase for the first 50 scenarios. Notice that this set of experiments was run on 500 scenarios. The average time increase of using Al_6 ($k = 30$) is 339.39%.

- Peak memory use:

  Fig. 6.34 compares the maximum number of BDD nodes in a belief state for the first 50 scenarios. Al_6 ($k = 30$) consistently has lower peak memory use than Al_0 does. This is because Al_6 ($k = 30$) uses abstract belief states in the global model, which reduces the size of the belief state compared to using Al_0. Notice that this set of experiments was run on 500 scenarios. The average difference of using Al_6 ($k = 30$) is -18.49%.

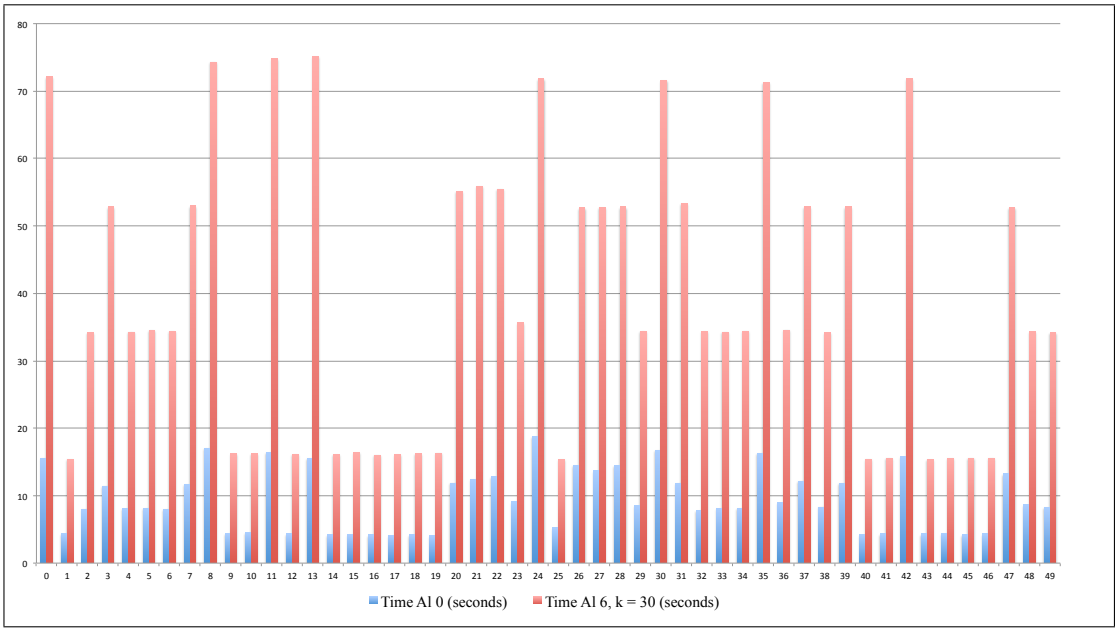

Figure 6.33: Rum time comparison between Al_0 and Al_6 ($k = 30$): the x-axis shows the scenario IDs, and the y-axis shows the run time in seconds

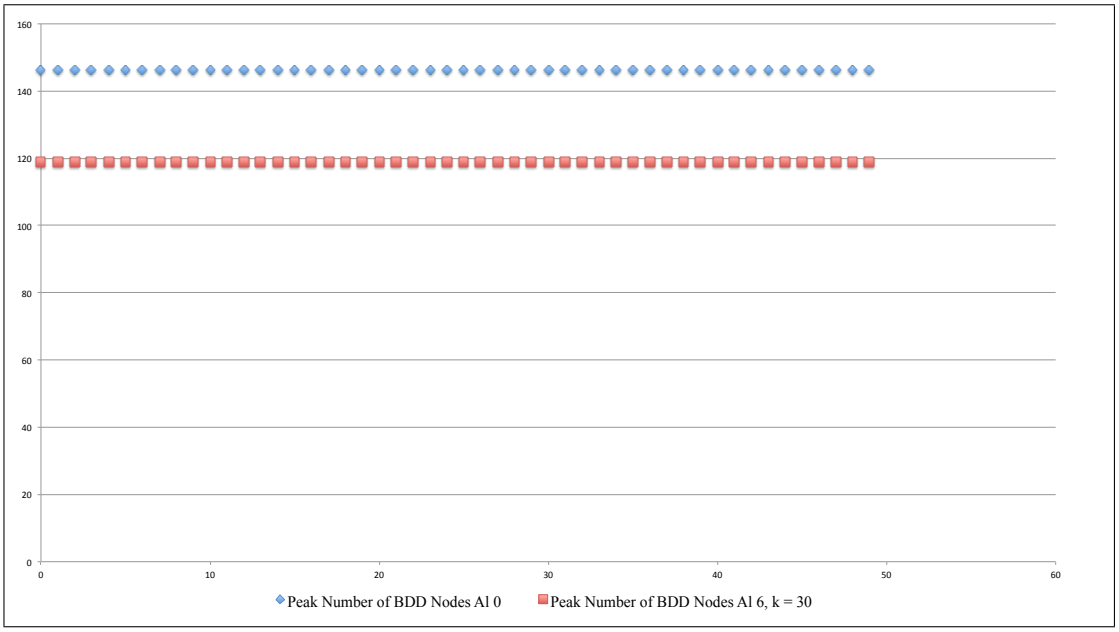

Figure 6.34: Peak number of BDD nodes in a belief state: comparison between Al_0 and Al_6 ($k = 30$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

- Average memory use:

  Fig. 6.35 compares the average number of BDD nodes in a belief state. The average memory use of Al_6 ($k = 30$) is consistently lower than that of Al_0. This

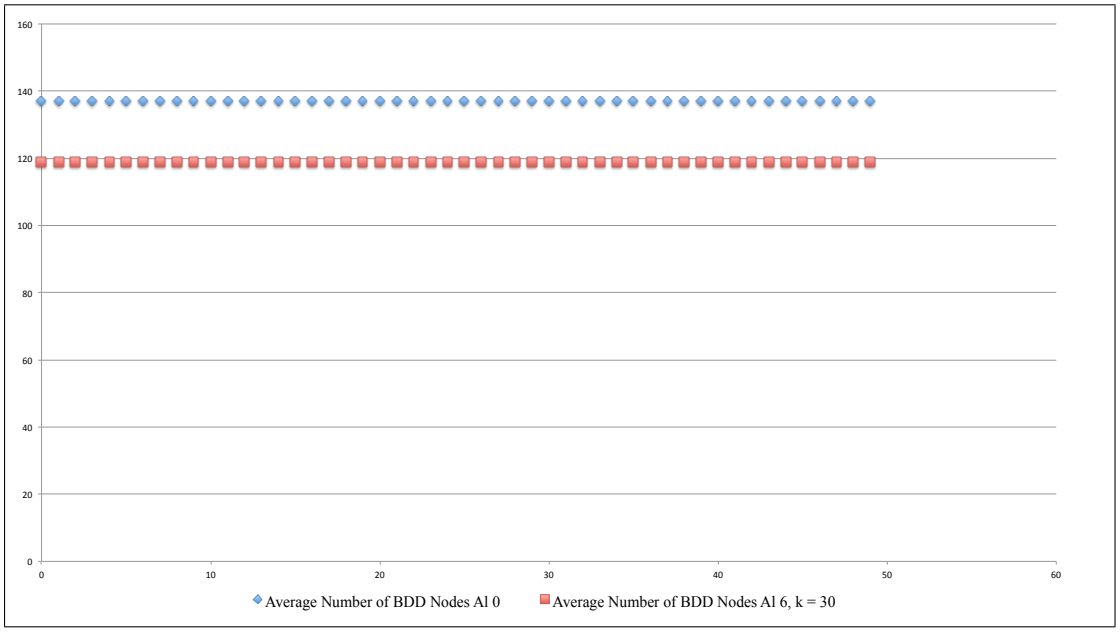

Figure 6.35: Average number of BDD nodes in a belief state: comparison between Al_0 and Al_6 ($k = 30$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

result is consistent with the peak memory use. The average difference of using Al_6 ($k = 30$) is -13.14%.

- Diagnostic distance:

  The diagnostic distance of using Al_6 ($k = 30$) is 0, which is consistently the same as that of using Al_0.

In summary, this set of experiments indicates that Al_6 ($k = 30$) precisely diagnoses "robot" model 3. Although the computational time is increased compared to using Al_0, both the peak and average memory use are reduced. Also, the selected abstract states lead to a precise TWA Al_6 ($k = 30$), i.e. Al_6 ($k = 30$) is precise to diagnose "robot" model 3. Notice that Section 6.5.2 shows that Al_5 ($k = 30$) is not always precise to diagnose "robot" model 3 using the same abstract states. In contrast, Al_6 is precise to diagnose "robot" model 3 without requiring more detailed abstract states. Therefore, this set of experiments shows that the benefits of TWA Al_6 are consistent with the theory presented in Chapter 5.

6.5.5 Comparison between Al_0 and Al_6 ($k = 60$)

This work compares the experimental results of Al_6 ($k = 60$) with those of Al_0, and evaluates the performance of Al_6 ($k = 60$) in five aspects as specified in Section 6.3.1.

- Precision:

  The exact diagnostic algorithm Al_0 diagnoses all of the 500 scenarios as faulty. Al_6 ($k = 60$) also precisely diagnoses the fault in each of the 500 scenarios.

- Run time:

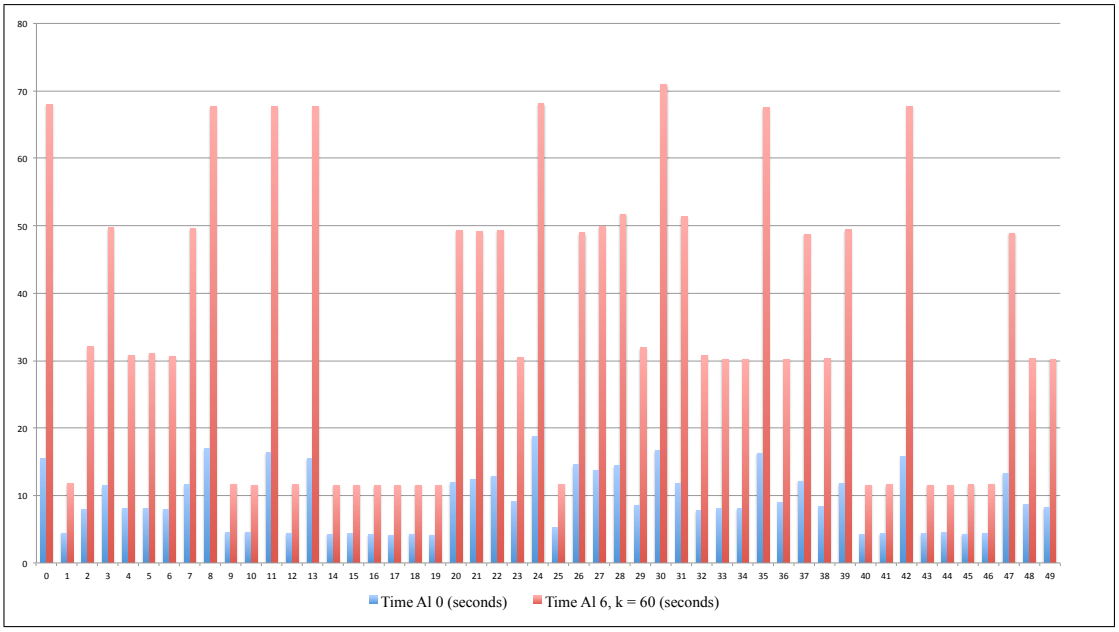

Figure 6.36: Rum time comparison between Al_0 and Al_6 ($k = 60$): the x-axis shows the scenario IDs, and the y-axis shows the run time in seconds

  Fig. 6.36 compares the run time of the first 50 scenarios between Al_0 and Al_6 ($k = 60$) measured in seconds. This plot shows the time increase for the first 50 scenarios. Notice that this set of experiments was run on 500 scenarios. The average time increase of using Al_6 ($k = 60$) is 288.56%.

- Peak memory use:

  Fig. 6.37 compares the maximum number of BDD nodes in a belief state for the first 50 scenarios. Al_6 ($k = 60$) consistently has lower peak memory use than Al_0 does. This is because Al_6 ($k = 60$) uses abstract belief states in the global model, which reduces the size of the belief state compared to using Al_0. Notice that this set of experiments was run on 500 scenarios. The average difference of using Al_6 ($k = 60$) is -18.49%.

- Average memory use:

  Fig. 6.38 compares the average number of BDD nodes in a belief state. The average memory use of Al_6 ($k = 60$) is consistently lower than that of Al_0. This

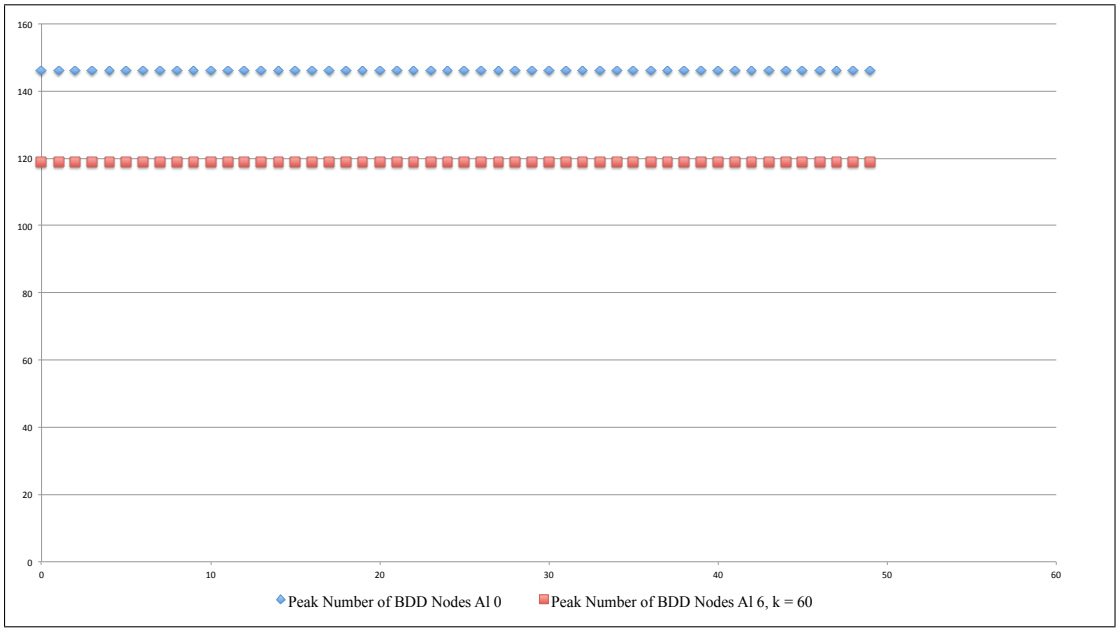

Figure 6.37: Peak number of BDD nodes in a belief state: comparison between Al_0 and Al_6 ($k = 60$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

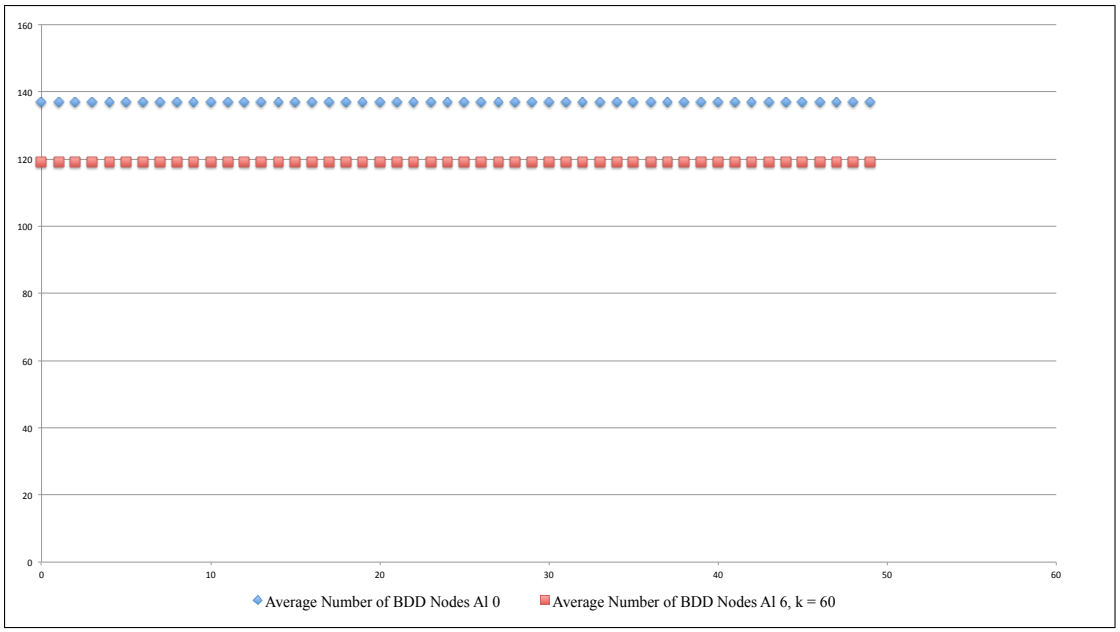

Figure 6.38: Average number of BDD nodes in a belief state: comparison between Al_0 and Al_6 ($k = 60$). The x-axis shows the scenario IDs, and the y-axis shows the number of BDD nodes.

result is consistent with the peak memory use. The average difference of using Al_6 ($k = 60$) is -13.14%.

- Diagnostic distance:

  The diagnostic distance of using Al_6 ($k = 60$) is 0, which is consistently the same as that of using Al_0.

In summary, this set of experiments indicates that Al_6 ($k = 60$) precisely diagnoses "robot" model 3. Although the computational time is increased compared to using Al_0, both the peak and average memory use are reduced. Also, the selected abstract states lead to a precise TWA Al_6 ($k = 60$), i.e. Al_6 ($k = 60$) is precise to diagnose "robot" model 3. Notice that Section 6.5.3 shows that Al_5 ($k = 60$) is not always precise to diagnose "robot" model 3 using the same abstract states. In contrast, Al_6 is precise to diagnose "robot" model 3 without requiring more detailed abstract states. Therefore, this set of experiments shows that the benefits of TWA Al_6 are consistent with the theory presented in Chapter 5.

6.6 Summary

This chapter evaluates the performance of window-based diagnostic algorithms through experiments. This chapter presents the implementations, the experimental models, and how to generate the scenarios for diagnosis. The performance of three IWAs and two TWAs is measured by the precision of diagnosis, computational time, maximum memory use, average memory use, and diagnostic distance. Diagnostic distance is defined as the number of observations between a fault occurrence and the fault diagnosis of a diagnostic algorithm.

This chapter also reports the experimental settings, the data gather from the experiments, and the evaluations on Al_1, Al_2, Al_3, Al_5, and Al_6. It then compares the results with those of the exact diagnostic algorithm Al_0.

- For a "robot" model with 14 glasses and 9 customers, Al_1 ($k = 250$, $i = 0$) precisely diagnoses the fault in each of the 500 scenarios. However, Al_1 ($k = 150,200$, $i = 0$) are unable to precisely diagnose some of the 500 scenarios. This set of experiments provides a useful indication on how small the value of the time window size (k) can be while being able to precisely diagnose most scenarios of the "robot" model.

- For a "robot" model with 14 glasses and 9 customers, Al_2 ($k = 250$, $i = 0$) precisely diagnoses the fault in each of the 500 scenarios. However, Al_2 ($k = 150,200$, $i = 0$) are unable to precisely diagnose some of the 500 scenarios. Notice that the time windows of Al_2 ($k = 250$, $i = 0$) *cover* those of Al_1 ($k = 250$, $i = 0$). This set of experiments provides a useful indication on how small the

value of the time window size (k) can be while being able to precisely diagnose most scenarios of the "robot" model.

- For a "robot" model with 14 glasses and 9 customers, Al_3 ($k = 350, i = \frac{\lfloor \frac{n}{k} \rfloor \times k}{2}, d = 350$) is able to precisely diagnose the fault in each of the 500 scenarios. This set of experiments demonstrates that the benefits of Al_3 are consistent with the theory presented in Chapter 4, i.e. Al_3 skips a specified number of observations in a scenario, runs faster than the exact diagnostic algorithm Al_0, and has lower average memory use. On the other hand, the maximum memory use is higher than that of Al_0 due to the resetting of system states and diagnosing observed events in a new time window.

- For "robot" model 2 with 14 glasses and 9 customers, Al_5 ($k = 25, 50$) precisely diagnose the fault in each of the 500 scenarios. This set of experiments demonstrates that the benefits of Al_5 are consistent with the theory presented in Chapter 5, i.e. the maximum and average memory use is reduced compared to using Al_0. The trade-off is that the computational time is longer than that of Al_0 due to the operations on abstract belief states performed between the time windows.

- For "robot" model 3 with 14 glasses and 9 customers, Al_5 ($k = 30, 60$) are unable to precisely diagnose the fault in some of the 500 scenarios while Al_6 ($k = 30, 60$) precisely diagnose the fault in each of the 500 scenarios. This set of experiments demonstrates that Al_6 is more precise than Al_5 without requiring more detailed abstract states. It also demonstrates that the benefits of Al_6 are consistent with the theory presented in Chapter 5, i.e. both maximum and average memory use are reduced compared to using Al_0. The trade-off is that the computational time is longer than that of Al_0 due to the operations on abstract belief states performed between the time windows.

In conclusion, the experimental results of IWAs show that Al_3 has promising performance in comparison to the exact diagnostic algorithm Al_0. In particular, Al_3 ($k = 350, i = \frac{\lfloor \frac{n}{k} \rfloor \times k}{2}, d = 350$) precisely diagnoses all of the 500 scenarios. Also, the run time and the average memory use are consistently reduced compared to using Al_0. The results of Al_3 ($k = 350$) indicate that a greater value of k, i.e. fewer time windows, leads to shorter computational time, as well as lower peak and average memory use than using a smaller value of k. To diagnose a component-based DES model, IWA Al_3 has good diagnostic performance since Al_3 can skip a specified number of observations and start diagnosis as an IWA.

The experimental results of TWAs demonstrate that both Al_5 and Al_6 reduce the maximum and average memory use compared to using Al_0. In comparison to Al_5, Al_6 improves the precision without requiring more detailed abstract states. However, the trade-off of using TWAs is that the computational time is longer than that of using Al_0 due to the operations on abstract belief states performed between the time windows. Given a DES model, the simulations for Al_5 and Al_6 presented in Section 5.4 of Chapter 5 can be used to verify the precision of a TWA w.r.t. the given DES model.

Chapter 7

Conclusion

7.1 Summary

This Thesis addresses the problem of on-line diagnosis of a DES, which was initially proposed by Sampath et al. [1995]. Given a flow of observable events generated by the underlying system, the problem consists in determining whether the DES is operating normally or not, based on a behavioral model of it. On-line diagnosis means diagnosing a system on the fly and in real time such that constraints on the computational time and memory space are imposed. This work identifies that the main challenge of on-line diagnosis is to deal with the complexity of a diagnostic algorithm in order to monitor the observable flow on the fly, and generate a succession of belief states that are consistent with the flow. In fact, the difficulty is that the number of the belief states has been proved to be exponential w.r.t. the number of system states [Rintanen, 2007]. The existing diagnostic algorithms attempt to compute at any time a belief state that is consistent with the observable flow from the time when the system starts operating to the current time. The main drawback of such a conservative strategy is the inability to *follow* the observable flow for a large system due to the exponential size of the generated belief states. Also, the temporal complexity to handle all of the belief states remains a challenge.

Because diagnosis of DES is a hard problem, the use of faster diagnostic algorithms is inevitable. Such algorithms include the IWAs [Su and Grastien, 2013], which do not carry all of the historical information about the system execution, and chronicle-recognition diagnostic algorithms [Dousson, 1996], which uses pattern recognition techniques for diagnosis. However, these algorithms may be imprecise to diagnose a diagnosable system than using an exact model-based diagnostic algorithm, e.g. Sampath et al. diagnosis [Sampath et al., 1995]. Faults are very harmful to a system and expensive to recover from if not correctly diagnosed. Hence, it is essential to examine how to measure the quality of using a potentially imprecise diagnostic algorithm w.r.t. a diagnosable DES model.

In the literature, diagnosability of DES is an important property to measure the capability of a diagnostic system to identify faults and the quality of diagnosis. Diagnosability is well-known criterion of DES, which is initially proposed by Sampath et al. [1995]. Diagnosability of DES holds if using the model, a fault can always be diagnosed after it occurs. Chapter 3 extends the diagnosability of a DES model, and studies the precision of an imprecise diagnostic algorithm w.r.t. a DES model. A diagnostic algorithm is defined as *precise* w.r.t. a DES model if it always diagnoses the fault after it occurs. Precision can be verified using known methods such as the twin plant method by Jiang et al. [2001], on the condition that a simulation is built, which is a modified model that *simulates* how a diagnostic algorithm runs on a given DES model. The precision holds iff there is no critical witness in the synchronisation of the DES model and the simulation. This work also illustrates how to construct the simulation for chronicle-recognition diagnosis in Chapter 3.

As one is able to verify the precision of a diagnostic algorithm w.r.t. a DES model by building a simulation, Chapter 4 proposes a new class of diagnostic algorithms called Independent-Window Algorithms (IWAs). IWAs are window-based diagnostic algorithms, which slice a sequence of observations into time windows, and diagnose them independently. This approach is different to the conservative strategy. It proposes diagnostic algorithms that are only applied on the very last events of the observable flow, and *forget* about the past. IWAs slice an observation sequence into time windows so that each time window is diagnosed independently. IWAs diagnose a certain number of observations for one time window, and move to another time window without keeping any information. On the other hand, IWAs may cause imprecise diagnosis. Since IWAs diagnose time windows independently, imprecision happens when both current and past observations are necessary to understand the system behaviour. Chapter 4 demonstrates the construction of a simulation in order to verify the precision an IWA w.r.t. a DES model.

Chapter 5 further proposes Time-Window Algorithms (TWAs). TWAs are inspired by IWAs. Chapter 5 formally presents two TWAs, namely, Al_5 and Al_6. The difference is that TWAs remember a certain knowledge between the time windows so that the precision of diagnosis is preserved. The strategy of TWAs is a compromise between the two extreme strategies of exact and imprecise diagnosis, e.g. a compromise between Sampath et al. diagnosis [Sampath et al., 1995] and IWAs [Su and Grastien, 2013]. Such a compromise is achieved by looking for the minimum piece of information to *remember* from the past, called *abstracted belief state*, so that a window-based algorithm will certainly ensure the same precision as using an exact diagnostic algorithm. Chapter 5 also demonstrates how to verify the precision of each TWA w.r.t. a DES model by constructing a simulation.

Chapter 6 evaluates the performance of window-based diagnostic algorithms through experiments. It evaluates Al_1, Al_2, Al_3, Al_5, and Al_6 by comparing their results with those of the exact diagnostic algorithm Al_0. The performance is measured by the precision of diagnosis, computational time, peak memory use, average memory use, and diagnostic distance. Diagnostic distance is defined as the number of observations between a fault occurrence and the fault diagnosis of a diagnostic algorithm. The experimental results of Al_3 ($k = 350, i = \frac{\lfloor \frac{n}{k} \rfloor \times k}{2}, d = 350$) indicate that a greater value of k, i.e. fewer time windows, leads to shorter computational time, as well as lower peak and average memory use than using a smaller value of k. To diagnose a component-based DES model, IWA Al_3 has good diagnostic performance since Al_3 can skip a specified number of observations and start diagnosis as an IWA. The experimental results of TWAs demonstrate that both Al_5 and Al_6 reduce the maximum and average memory use compared to using Al_0. In comparison to Al_5, Al_6 improves the precision without requiring more detailed abstract states. However, the trade-off of using a TWA is that the computational time is longer than that of using Al_0 due to the operations on abstract belief states performed between the time windows. Finally, the precision of a TWA w.r.t. a DES model varies on the settings of abstract states. Given a DES model, the simulations for Al_5 and Al_6 presented in Section 5.4 of Chapter 5 can be used to verify the precision of a TWA w.r.t. the given DES model.

7.2 Future Work

The future work consists of three aspects. First, Chapter 3 proposes to verify the precision of a diagnostic algorithm w.r.t. a DES model by constructing a simulation. This is a general method that is useful to verify the precision of a diagnostic approach. This novel approach also allows researchers to consider more aggressive approaches to reduce the complexity of diagnosis, as they can now assess the precision of their diagnostic approaches. Therefore, it is worthwhile to extend this theory to the case where the system is not diagnosable. The research question will focus on the impact of using an imprecise diagnostic algorithm on a non-diagnosable system. One approach is to categorise a scenario of a non-diagnosable system into nominal, faulty, or ambiguous. Given a scenario, the performance of using an imprecise diagnostic algorithm on the scenario needs to be compared with the result of using an exact diagnostic algorithm. Such a comparison should also consider the category of a scenario, i.e. whether it is nominal, faulty, or ambiguous.

Second, this work proposes window-based diagnostic algorithms in Chapter 4 and Chapter 5. The extension is *backbone diagnosis*, which aims to identify what is

known for sure during a diagnostic process. Notice that both TWAs and backbone diagnosis remember the information from the past time windows of diagnosis. The main difference between TWAs and backbone diagnosis is that TWAs keep track of the abstract belief state of a system while backbone diagnosis maintains the known information, e.g. a certain variable holds or not. Furthermore, backbone diagnosis will be useful to investigate the root cause of ambiguity if a diagnoser is unable to decide whether the system is in the nominal mode or the faulty mode. For example, the system status is precisely known up to which observation in a scenario, and then the ambiguity appears. In summary, backbone diagnosis is inspired by TWAs. Backbone diagnosis is useful to investigate the root cause of ambiguity so that the sensors of the system model may be modified in order to resolve the ambiguity of diagnosis.

Third, AI planning and restoration is a subject that is closely related to this work on diagnosis and diagnosability. It is an important research question to combine the planning technique developed by Ciré and Botea [2008] where the goal is defined as a specification in Linear Temporal Logic (LTL), with the symbolic diagnosability testing developed by Grastien [2009] where diagnosability testing is specified in terms of LTL formula. Therefore, the future work will focus on a model that captures both aspects so that diagnosis leads to accurate alarm processing while planning leads to robust configuration and restoration. Self-healability was initially studied by Cordier et al. [2007]. Self-healing is a problem in which a system should be able to diagnose and repair itself [Cordier et al., 2007; Grastien, 2015]. Grastien [2015] further proposed to combine consistency checks and conformant planning problems. Grastien [2015] firstly defined diagnosis as a problem to find the optimal repair plan, and then presented a method to combine diagnosis and repair using DES modelling. A planner searches a plan based on a sample of the system belief state. After that, a diagnoser verifies the applicability of a plan. If the plan is not valid, the diagnoser will return a state of the belief state that needs to be added to the sample. This process is repeated until a valid repair plan is found. Notice that the self-healing process does not involve the computation of the exact belief state [Grastien, 2015]. In the context of Smart Grid, it is an open research topic to develop an approach that joins AI planning and diagnosis to improve the efficiency of event and failure handling, minimise the impact of power loss, as well as recover and restore to a robust state withholding the diagnosability property.

Appendix A

Figures for the "Robot" Model

The appendix shows the figures for the components in the "robot" model of Chapter 6 with one instance of `Order Monitor`, one instance of `Fault Monitor`, two instances of `Customer`, and two instances of `Glass`.

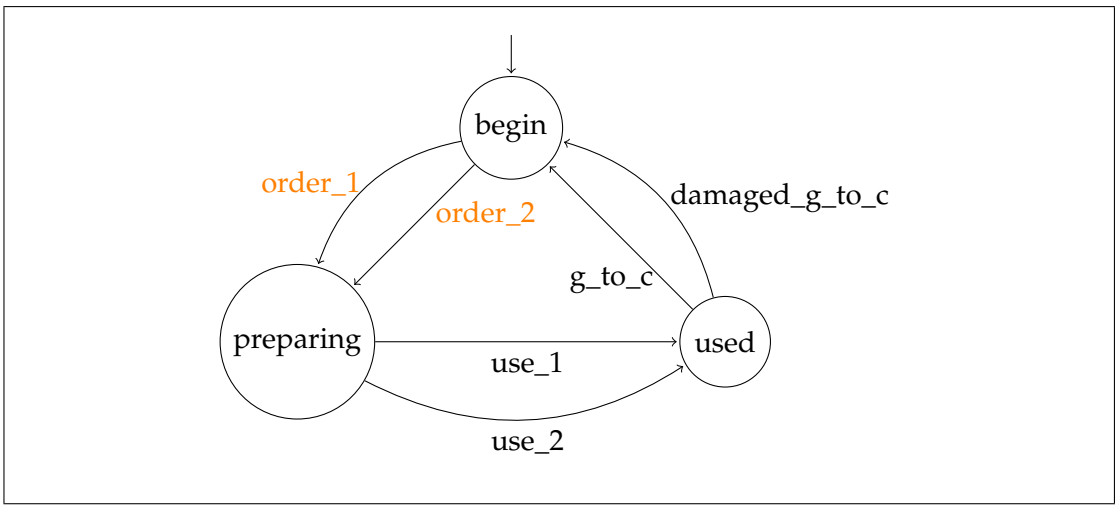

Figure A.1: An *Order Monitor* for two customers and two glasses: "g_to_c" means giving a normal glass to a customer, and "damaged_g_to_c" means giving a damaged glass to a customer; "order_1" and "order_2" are the observable events while the other events are unobservable.

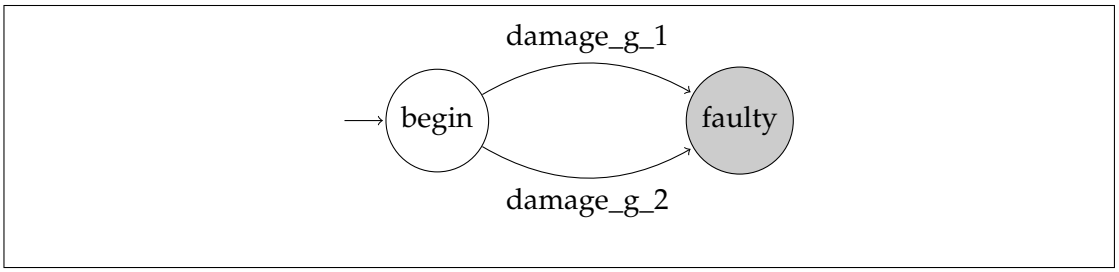

Figure A.2: A *Fault Monitor* for two glasses: "damage_g_1" means glass 1 becomes damaged, and "damage_g_2" means glass 2 becomes damaged. Both events are unobservable.

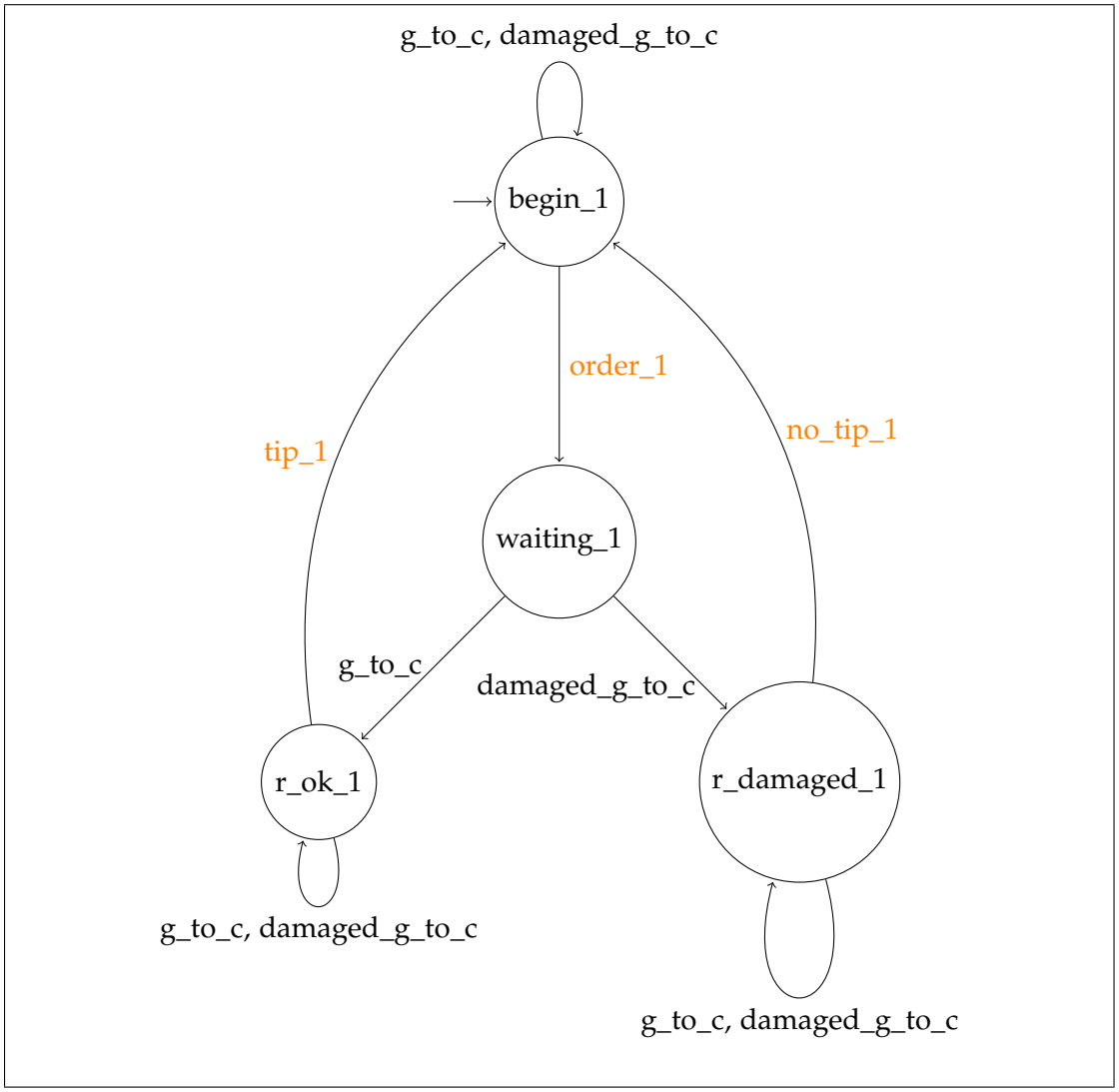

Figure A.3: *Customer 1*: the state "r_ok_1" means that customer 1 has received a normal glass, and the state "r_damaged_1" means that customer 1 has received a damaged glass; the event "g_to_c" means giving a normal glass to a customer, and the event "damaged_g_to_c" means giving a damaged glass to a customer; "order_1", "tip_1", and "no_tip_1" are the observable events while the other events are unobservable; the loops with "g_to_c" and "damaged_g_to_c" are used to model the sequencing of multiple orders and glasses, i.e. one glass is given to one customer.

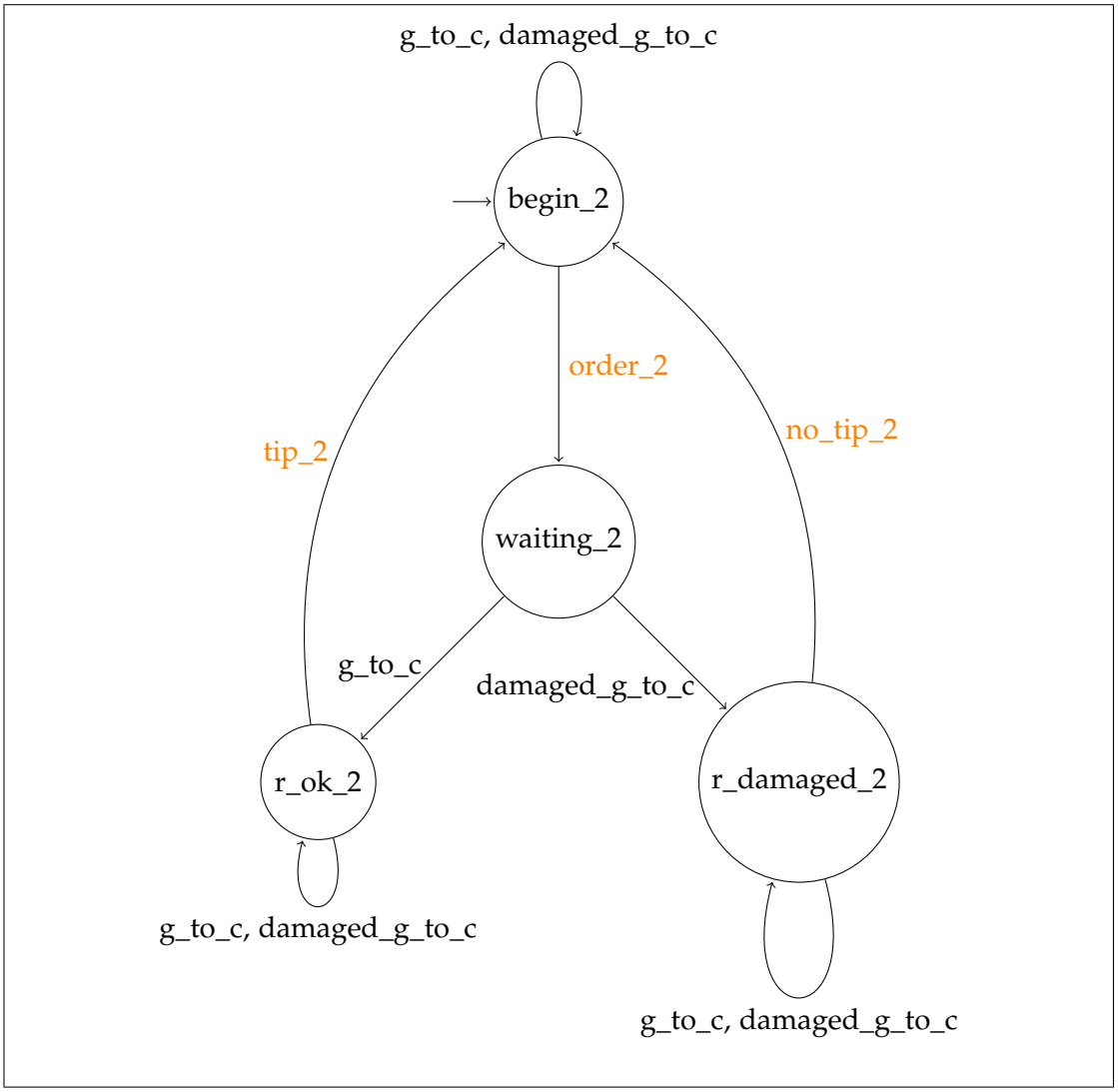

Figure A.4: *Customer 2*: the state "r_ok_2" means that customer 2 has received a normal glass, and the state "r_damaged_2" means that customer 2 has received a damaged glass; the event "g_to_c" means giving a normal glass to a customer, and the event "damaged_g_to_c" means giving a damaged glass to a customer; "order_2", "tip_2", and "no_tip_2" are the observable events while the other events are unobservable; the loops with "g_to_c" and "damaged_g_to_c" are used to model the sequencing of multiple orders and glasses, i.e. one glass is given to one customer.

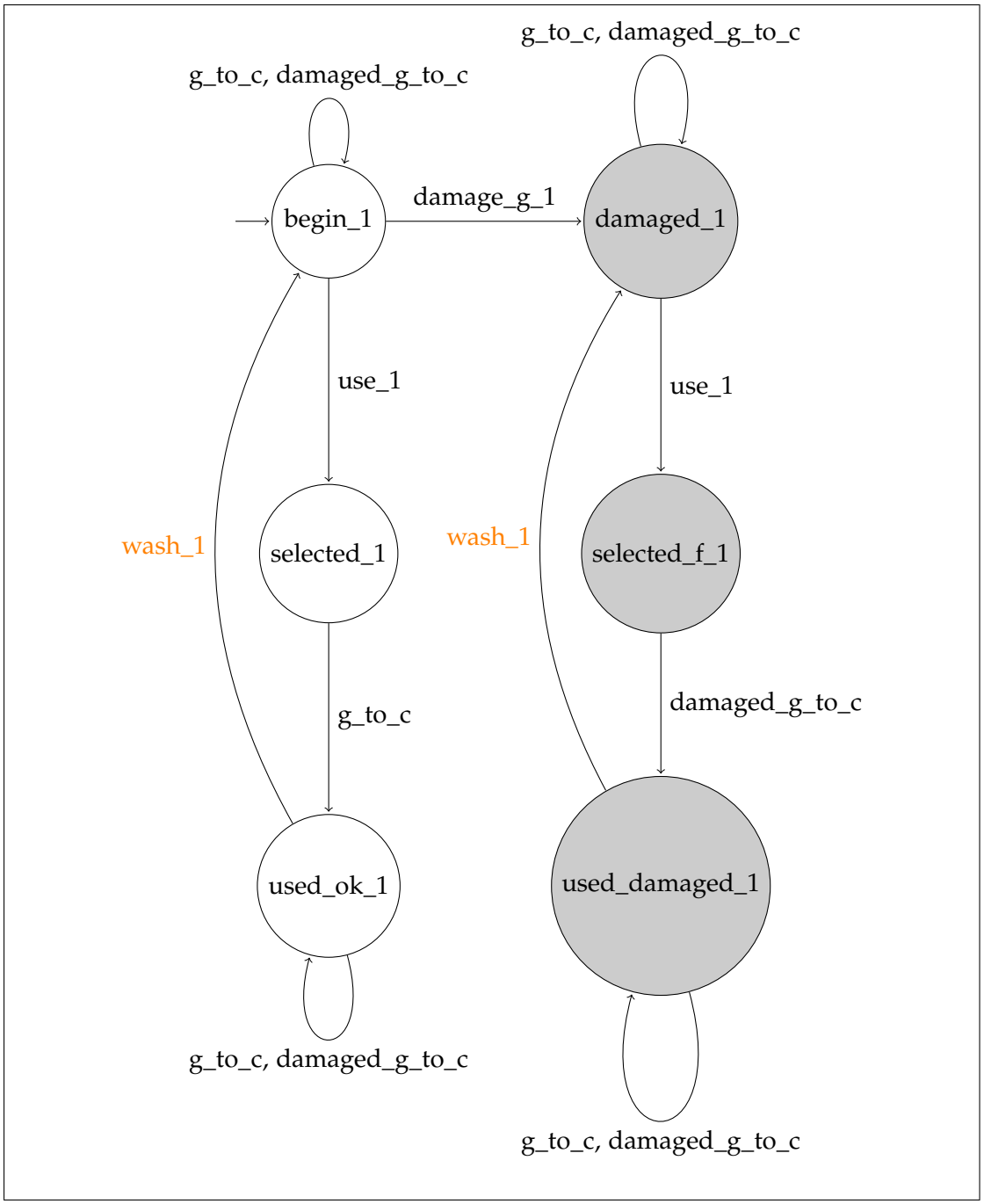

Figure A.5: *Glass 1*: the state "selected_f_1" means that the damaged glass 1 is selected for a customer order; the event "g_to_c" means that glass 1 is given to a customer; the event "damaged_g_to_c" means that a damaged glass is given to a customer; the event "damage_g_1" means that glass 1 is damaged; "wash_1" is the only observable event while the other events are unobservable; the four loops are used to model the sequencing of multiple orders and glasses.

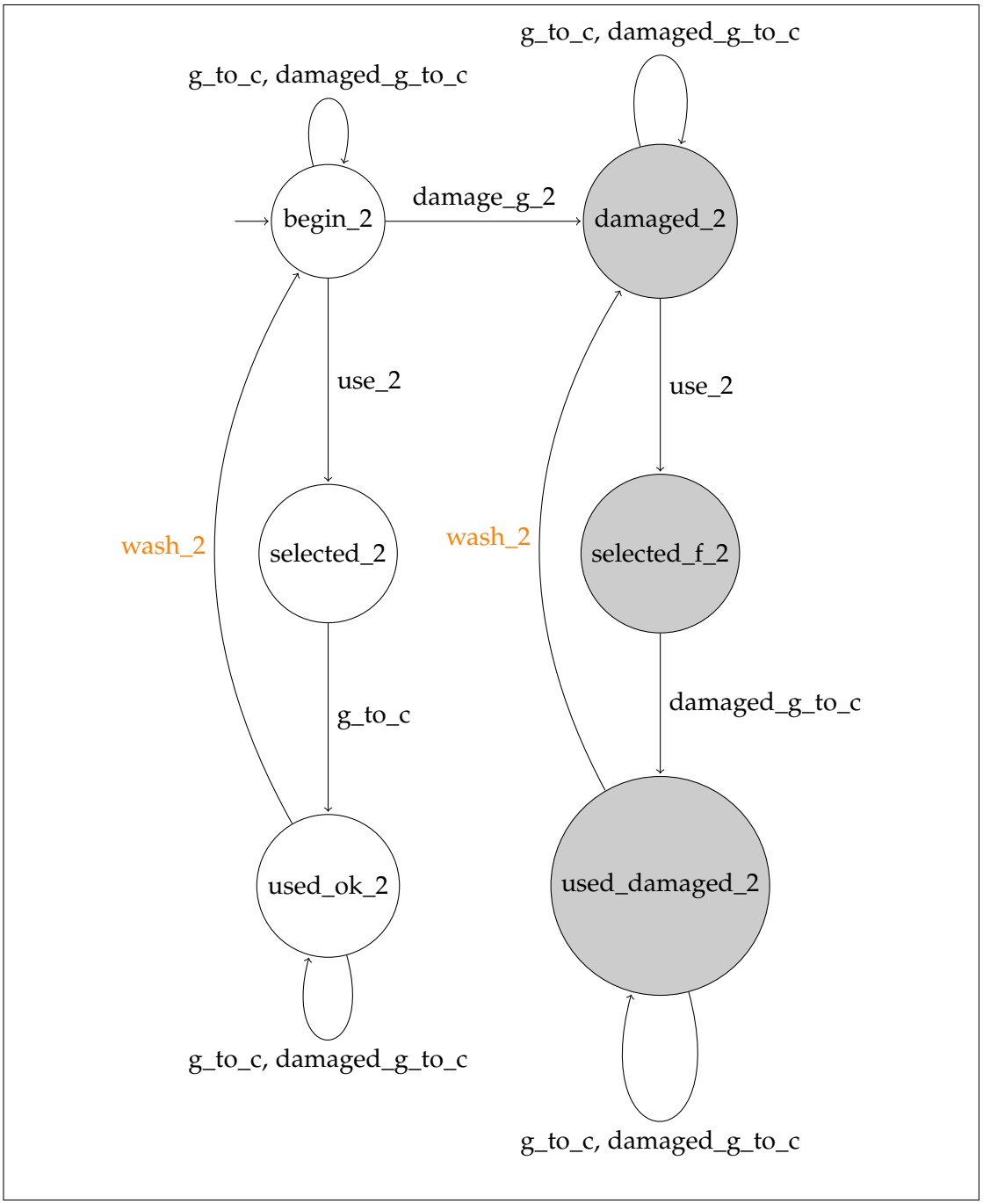

Figure A.6: *Glass 2*: the state "selected_f_2" means that the damaged glass 2 is selected for a customer order; the event "g_to_c" means that glass 2 is given to a customer; the event "damaged_g_to_c" means that a damaged glass is given to a customer; the event "damage_g_2" means that glass 2 is damaged; "wash_2" is the only observable event while the other events are unobservable; the four loops are used to model the sequencing of multiple orders and glasses.

Bibliography

Baroni, P.; Lamperti, G.; Pogliano, P.; and Zanella, M., 1999. Diagnosis of large active systems. *Artificial Intelligence (AIJ)*, 110, 1 (1999), 135–183. (cited on pages 2, 9, 10, 37, 63, and 65)

Bauer, A.; Botea, A.; Grastien, A.; Haslum, P.; and Rintanen, J., 2011. Alarm processing with model-based diagnosis of discrete event systems. In *22nd International Workshop on Principles of Diagnosis (DX-11)*, 52–59. (cited on page 90)

Bayoudh, M.; Travé-Massuyès, L.; and Olive, X., 2008. Coupling continuous and discrete event system techniques for hybrid system diagnosability analysis. In *18th European Conference on Artificial Intelligence (ECAI-08)*, 219–223. (cited on page 7)

Brandán-Briones, L.; Lazovik, A.; and Dague, P., 2008. Optimal observability for diagnosability. In *19th International Workshop on Principles of Diagnosis (DX-08)*, 31–38. (cited on pages 25, 29, and 116)

Cassandras, C. and Lafortune, S., 2008. *Introduction to Discrete Event Systems (2nd ed.)*. Springer, New York, N.Y. (cited on pages 1, 7, and 39)

Cassez, F., 2010. Dynamic observers for fault diagnosis of timed systems. In *49th IEEE Conference on Decision and Control (CDC-10)*, 4359–4364. (cited on pages 29, 33, and 35)

Cassez, F. and Tripakis, S., 2008. Fault diagnosis with static and dynamic observers. *Fundamenta Informaticae*, 88, 4 (2008), 497–540. (cited on page 49)

Cassez, F.; Tripakis, S.; and Altisen, K., 2007. Sensor minimization problems with static or dynamic observers for fault diagnosis. In *7th International Conference on Application of Concurrency to System Design, IEEE Computer Society (ACSD-07)*, 90–99. (cited on page 33)

Cimatti, A.; Pecheur, C.; and Cavada, R., 2003. Formal verification of diagnosability via symbolic model checking. In *18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, 363–369. (cited on page 26)

Ciré, A. and Botea, A., 2008. Learning in planning with temporally extended goals and uncontrollable events. In *18th European Conference on Artificial Intelligence (ECAI-08)*, 578–582. (cited on page 166)

Clarke, E.; Grumberg, O.; and Peled, D., 1999. *Model Checking*. MIT Press, Cambridge, MA. (cited on page 26)

Cordier, M.-O.; Pencolé, Y.; Travé-Massuyès, L.; and Vidal, T., 2007. Self-healability = diagnosability + repairability. In *18th International Workshop on Principles of Diagnosis (DX-07)*, 251–258. (cited on page 166)

de Kleer, J. and Williams, B., 1987. Diagnosing multiple faults. *Artificial Intelligence (AIJ)*, 32, 1 (1987), 97–130. (cited on page 7)

Dechter, R.; Meiri, I.; and Pearl, J., 1991. Temporal constraint networks. *Artificial Intelligence (AIJ), Special Vol. on Knowledge Representation*, 49, 1–3 (1991), 61–95. (cited on pages 51 and 54)

Dousson, C., 1996. Alarm driven supervision for telecommunication networks: II on-line chronicle recognition. *Annals of Telecommunications (AOT)*, 51, 9–10 (1996), 501–508. (cited on pages 2, 34, 38, 49, 51, 61, and 163)

Epp, S., 2004. *Discrete Mathematics with Applications*. Brooks/Cole Publishing Company, Boston, MA. (cited on page 57)

Grastien, A., 2009. Symbolic testing of diagnosability. In *20th International Workshop on Principles of Diagnosis (DX-09)*, 131–138. (cited on pages 24, 25, 27, and 166)

Grastien, A., 2015. Self-healing as a combination of consistency checks and conformant planning problems. In *26th International Workshop on Principles of Diagnosis (DX-15)*. (cited on page 166)

Grastien, A. and Anbulagan, 2009. Incremental diagnosis of des with a non-exhaustive diagnosis engine. In *20th International Workshop on Principles of Diagnosis (DX-09)*, 345–352. (cited on pages 19, 20, and 35)

Grastien, A.; Anbulagan; Rintanen, J.; and Kelareva, E., 2007. Diagnosis of discrete-event systems using satisfiability algorithms. In *22nd Conference on Artificial Intelligence (AAAI-07)*, 305–310. (cited on pages 2, 14, 15, 16, 37, 63, 93, and 112)

Grastien, A. and Torta, G., 2011. A theory of abstraction for diagnosis of dicrete-event systems. In *9th Symposium on Abstraction, Reformulation and Approximation (SARA-11)*, 50–57. (cited on pages 44 and 49)

Hashtrudi Zad, S.; Kwong, R.; and Wonham, W., 2003. Fault diagnosis in discrete-event systems: framework and model reduction. *IEEE Transactions on Automatic Control (TAC)*, 48, 7 (2003), 1199–1212. (cited on page 23)

Jéron, T.; Marchand, H.; Pinchinat, S.; and Cordier, M.-O., 2006. Supervision patterns in discrete event systems diagnosis. In *8th International Workshop on Discrete Event Systems (WODES-06)*, 262–268. (cited on pages 7, 11, 21, 33, and 40)

Jiang, S.; Huang, Z.; Chandra, V.; and Kumar, R., 2001. A polynomial algorithm for diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control (TAC)*, 46, 8 (2001), 1318–1321. (cited on pages 3, 24, 28, 34, 38, 45, 46, and 164)

Kan John, P.; Grastien, A.; and Pencolé, Y., 2010. Synthesis of a distributed and accurate diagnoser. In *21st International Workshop on Principles of Diagnosis (DX-10)*, 209–216. (cited on page 49)

Kautz, H. and Selman, B., 1992. Planning as satisfiability. In *10th European Conference on Artificial Intelligence (ECAI-92)*, 359–363. (cited on page 14)

Lamperti, G. and Zanella, M., 2007. On monotonic monitoring of discrete-event systems. In *18th International Workshop on Principles of Diagnosis (DX-07)*, 130–137. (cited on page 46)

Pencolé, Y. and Cordier, M.-O., 2005. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence (AIJ)*, 164, 1–2 (2005), 121–170. (cited on pages 2, 3, 17, 18, 37, 63, and 65)

Pencolé, Y. and Subias, A., 2009. A chronicle-based diagnosability approach for discrete timed-event systems: application to web-services. *Journal of Universal Computer Science (JUCS)*, 15, 17 (2009), 3246–3272. (cited on page 50)

Reiter, R., 1987. A theory of diagnosis from first principles. *Artificial Intelligence (AIJ)*, 32, 1 (1987), 57–95. (cited on page 7)

Rintanen, J., 2007. Diagnosers and diagnosability of succinct transition systems. In *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 538–544. (cited on pages 2, 3, 4, 11, 33, 37, 63, 65, 66, 92, 117, and 163)

Rintanen, J., 2009. Planning and SAT. In *Handbook of Satisfiability* (Eds. A. Biere; M. Heule; H. van Maaren; and T. Walsh), vol. 185 of *Frontiers in Artificial Intelligence and Applications*, 483–504. IOS Press. (cited on page 14)

Rintanen, J. and Grastien, A., 2007. Diagnosability testing with satisfiability algorithms. In *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 532–537. (cited on page 27)

Rozé, L. and Cordier, M.-O., 2002a. Diagnosing discrete-event systems : extending the "diagnoser approach" to deal with telecommunication networks. *Journal of Discrete Event Dynamical Systems (JDEDS)*, 12, 1 (2002), 43–81. (cited on pages 2, 37, and 63)

Rozé, L. and Cordier, M.-O., 2002b. Diagnosing discrete-event systems: extending the diagnoser approach to deal with telecommunication networks. *Journal on Discrete-Event Dynamic Systems: Theory and Applications (JDEDS)*, 12, 1 (2002), 43–81. (cited on page 91)

Russell, S. and Norvig, P., 2010. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, N.J. (cited on page 7)

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D., 1995. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control (TAC)*, 40, 9 (1995), 1555–1575. (cited on pages 1, 2, 3, 5, 10, 13, 23, 24, 33, 34, 35, 37, 38, 44, 45, 63, 64, 65, 91, 112, 163, and 164)

Schumann, A., 2007. *Towards Efficiently Diagnosing Large Scale Discrete-Event Systems*. Doctor of Philosophy Thesis, The Australian National University, Canberra, Australia. (cited on pages 5, 119, 120, and 122)

Schumann, A.; Pencolé, Y.; and Thiébaux, S., 2010. A decentralised symbolic diagnosis approach. In *19th European Conference on Artificial Intelligence (ECAI-10)*, 99–104. (cited on pages 2, 12, 37, 63, 65, 78, and 112)

Su, R. and Wonham, W., 2005. Global and local consistencies in distributed fault diagnosis for discrete-event systems. *IEEE Transactions on Automatic Control (TAC)*, 50, 12 (2005), 1923–1935. (cited on pages 2, 37, and 63)

Su, X. and Grastien, A., 2013. Diagnosis of discrete event systems by independent windows. In *24th International Workshop on Principles of Diagnosis (DX-13)*, 148–153. (cited on pages 2, 5, 6, 34, 35, 38, 163, and 164)

Su, X. and Grastien, A., 2014a. Verifying the precision of diagnostic algorithms. In *21st European Conference on Artificial Intelligence (ECAI-14)*, 861–866. (cited on page 6)

Su, X. and Grastien, A., 2014b. Window-based diagnostic algorithms for discrete event systems: what information to remember. In *25th International Workshop on Principles of Diagnosis (DX-14)*. (cited on page 6)

Yoo, T. and Lafortune, S., 2002. Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Transactions on Automatic Control (TAC)*, 47, 9 (2002), 1491–1495. (cited on pages 3, 24, 26, 34, 38, 45, and 90)