ECAI 2014
T. Schaub et al. (Eds.)

© 2014 The Authors and IOS Press.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License.
doi:10.3233/978-1-61499-419-0-861

Verifying the Precision of Diagnostic Algorithms

Xingyu Su[1] and **Alban Grastien**[1]

Abstract. Diagnosis of discrete event systems requires to decide whether the system model allows for certain types of executions to take place. Because this problem is hard, incomplete yet faster algorithms may be needed. This however can lead to a loss of precision.

This paper presents a method to decide whether precision is maintained by such incomplete algorithms. To this end we define the Simulation, which is a modification of the model that simulates how the algorithm works. We then use the twin plant method to decide whether diagnosability is maintained despite the imprecision of the diagnostic algorithm. We illustrate the benefits of this approach on two diagnostic algorithms, namely Independent-Windows Algorithms and Chronicle-based Diagnosis.

1 INTRODUCTION

Model-based diagnosis of discrete event systems (DES) aims at deciding whether a system is running normally or is experiencing faulty behaviors. The diagnoser uses a model of the system — a finite state machine in the case of DES — to search for system executions consistent with the observations [15, 1, 12].

Because diagnosis of DES is a hard problem the use of imprecise yet faster algorithms is sometimes inevitable. Such algorithms include the Independent-Windows Algorithm [19], which does not carry all the historical information about the system execution, and the Chronicle-based Diagnostic Algorithm [5], which uses pattern recognition techniques. These algorithms however are less precise than the perfect model-based algorithm.

Faults can be very harmful to a system and expensive to recover from if not correctly identified. Diagnosability—the property that every fault will be diagnosed—is therefore often imposed, for instance by adding expensive sensors. Imprecise algorithms on diagnosable systems may fail to identify faults. The goal is this article is to provide a general method for deciding whether an imprecise algorithm will preserve the diagnosability of a system.

Diagnosability of DES is a well-studied problem. It has been shown [9] that proving non-diagnosability amounts to finding a critical witness, a pair of (infinite) executions on the model that are indistinguishable (they produce the same observations), one of which is faulty and the other one nominal; and diagnosability can be proved by showing that there is no such witness.

We demonstrate in this paper that a similar approach can be used in our situation. We define the concept of "Simulation", a finite state machine that represents how the diagnostic algorithm works. Diagnosability is preserved iff no critical witness can be found on the synchronization of the system and the Simulation. We demonstrate the correctness of our approach and we illustrate the construction of the Simulation for the two diagnostic algorithms presented before.

This paper is organized as follows. Section 2 provides the definitions of DES model, diagnosis and diagnosability of DES. Section 3 defines the generic notion of a diagnostic algorithm and the issue of proving that an algorithm is precise. Section 4 reviews the Independent-Window Algorithms and demonstrates how to test the precision of such algorithms by building a Simulation. Section 5 reviews Chronicle-based Diagnosis and illustrates the precision testing by building a Simulation. Section 6 demonstrates our implementation of building a Simulation for Independent-Window Algorithms and for Chronicle-based Diagnosis. Section 7 concludes this study and outlines the future work.

2 DIAGNOSIS and PRELIMINARIES

We provide definitions for diagnosis and diagnosability of DES [2]. We also discuss how to verify the diagnosability of a DES model.

2.1 Diagnosis of DES

We use automata to represent DES, although we also give definitions at the language level. An *Automaton* is a tuple $\langle Q, \Sigma, T, I, L\rangle$ 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*, and $L : Q \to \{N, F\}$ is a *mode label function* where N stands for *nominal* and F for *faulty* such that $(\langle q, e, q'\rangle \in T \wedge L(q) = F) \Rightarrow L(q') = F$.

Faults can be defined at the event level or the state level. Traditionally faults are defined as specific events. Jéron et al. [8] defined faults as pattern of events. Both formalisms are equally expressive and we consider faults at the state level for simplicity. Because we consider diagnosable systems each fault can be diagnosed separately; we therefore make a single-fault assumption.

A system behavior is represented by a trajectory on the automaton $q_0 \xrightarrow{e_1} \dots \xrightarrow{e_k} q_k$ where $q_0 \in I$. We also view the model as a pair of languages $\mathcal{L} = \mathcal{L}_N \cup \mathcal{L}_F$ such that a sequence of events $e_1, \dots, e_k \in \mathcal{L}_l$ ($l \in \{N, F\}$) iff there exists a trajectory $q_0 \xrightarrow{e_1} \dots \xrightarrow{e_k} q_k$ where $L(q_k) = l$.

An automaton has a set of observable events $\Sigma_o \subseteq \Sigma$ and a set of unobservable events $\Sigma \setminus \Sigma_o$. The observation $obs(\sigma)$ of $\sigma \in \Sigma^*$ is defined as the restriction of σ to the set of observable events such that $obs(\sigma)$ leads to unobservable events in σ being removed.

The system to diagnose is modeled as an automaton. Figure 1 shows a graphical representation for a DES model. It visualizes the states, initial state, transitions, events and the results of the mode label function L, i.e. the state F is faulty and the other states are nominal. Finally, a, b, c, d are observable events and u, v are unobservable events.

[1] Optimisation Research Group, NICTA, Australia and Artificial Intelligence Group, Australian National University, Australia, email: u4383016@anu.edu.au, alban.grastien@nicta.com.au

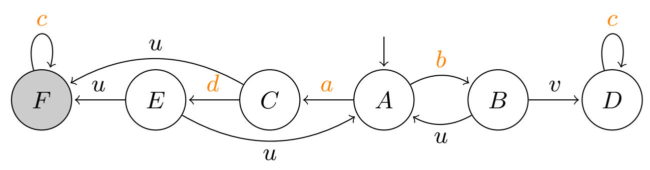

Figure 1. DES model

The *Diagnosis* $\Delta(M, o)$ of observation o using model M is defined by:

- N if $\exists \sigma \in \mathcal{L}_N.\ obs(\sigma) = o$ and
- F otherwise.

In the literature, diagnosis of DES is performed by computing on the complete model the paths that generate the observations received on the system, or equivalently the belief state of the system. This belief state can be computed on-line, by iteratively computing the set of states that can be reached from the current belief state through transitions that would produce exactly the next observation. If this is done explicitly as by Baroni et al. [1], the number of these states makes the approach inapplicable for many real-world problems. Schumann et al. [16] proposed symbolic approaches where the belief states are represented in propositional logic, e.g., BDD. However, symbolic representation is also subject to exponential blow-up in space. Finally, distributed, decentralized and hierarchical diagnosis aims to compute relevant components of a system and then merge to a final diagnosis because the computation of a global model is not practical for a large-scale system [12, 17, 18].

Sampath et al. [15] proposed to build a deterministic finite automaton that associates each sequence of observations with the diagnosis. Rintanen [14] proved however that this approach has an exponential size in the number of states, which makes it impractical for large systems.

2.2 Diagnosability of DES

Diagnosability is the question whether a fault will always be diagnosed [15]. This property is often a strong requirement that the system designer wants to enforce.

Diagnosability can be checked by searching for faulty trajectories 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 this reduction of the diagnosability problem, the model is used twice [7]: it is used i) to find the faulty trajectory on the system and ii) to (unsuccessfully) diagnose the trajectory. However we may have to use different models for these two usages, one model for the system and the other model for the diagnosis. If the system model is abstracted before being used for diagnosis, the abstracted model should not be used to generate the faulty trajectory (that trajectory may not be a possible system behavior); on the other hand, whether the trajectory can be diagnosed should be tested with the diagnosis model.

A diagnosis model M is *diagnosable* w.r.t. a system model M' if the following holds:

$$\exists n \in \mathbf{N}.\ \forall s \in \mathcal{L}'_F.\ \forall t \in \Sigma^*.$$

$$\left(st \in \mathcal{L}' \wedge |t| \geq n \Rightarrow \Delta(M, obs(st)) = F\right).$$

Given a faulty behavior s and additional n events (represented by t), the diagnosis (using the model M) of the observation $obs(st)$ produced by st should be "faulty".

Using automata, Jiang et al. [9] have shown that diagnosability can be checked in polynomial time with respect to the number of states. To this end they build a structure called a *twin plant*, which is the classical automaton synchronization 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 $L(q) = N$ and $L(q') = F$. A similar approach was also proposed by Yoo and Lafortune [20].

3 PRECISION OF DIAGNOSTIC ALGORITHMS

The diagnostic algorithm Δ is often impractical for large, real-world systems. We give here a formal definition of (potentially imprecise) diagnostic algorithms. We define the precision criterion that a diagnostic algorithm should satisfy and show how this criterion can be tested with the notion of a Simulation. We end the section with a comparison with existing, ad-hoc methods used in the literature to verify the precision of diagnostic algorithms.

3.1 Diagnostic algorithms and precision

We are interested in considering algorithms that may return results that are less precise than Δ, but that may run faster. We also want however to be able to give guarantees about the output of the algorithm, e.g., precision.

Definition 1 (Diagnostic Algorithm) *A* diagnostic algorithm *is a function* A : MODELS $\times\ \Sigma_o{}^* \rightarrow \{N, F\}$ *where* MODELS *is the set of possible models and the following holds:*

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

The first condition ensures that the diagnosis is monotonic [11], 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 (the converse may not hold).

Definition 2 (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 very similar to that of diagnosability (Section 2.2) except that we require the algorithm A (as opposed to Δ) to detect the fault.

3.2 Simulation

Diagnosability of a model can be tested using the twin plant approach. We here show how to decide the precision of a diagnostic algorithm. To this end we define a new model $si(M, A)$ that "simulates" the behavior of the diagnostic algorithm.

Definition 3 (Simulation) *Given a diagnostic algorithm A and a model M, the* Simulation *of A and M is an Automaton $si(M, A)$ s.t. $\forall o \in \Sigma_o{}^*.\ A(M, o) = \Delta(si(M, A), o)$.*

Examples of Simulation for the system model M' will appear in Section 4 and 5. Using the Simulation, we can use the following theorem to prove diagnosability.

Theorem 1 *Algorithm A is precise for a diagnosis model M 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 Looking at Definition 2, $A(M, obs(st))$ equals $\Delta(si(M, A), o)$ by Definition 3, which means that the formula of Definition 2 transforms into the formula of model diagnosability where M is replaced by $si(M, A)$ and $o = obs(st)$. □

Theorem 1 gives us a procedure of verifying the precision of an algorithm. Given a Simulation of the diagnostic algorithm one can compute the twin plant by synchronizing the model with the Simulation and search for ambiguous cycles. Such cycles are witnesses of system behaviors that are not diagnosable.

3.3 Related work

We examine three cases of testing the precision of diagnosis. They are distributed diagnosis, diagnosis using an Abstract Model of DES and diagnosis using static and dynamic sensors.

First, Kan John et al. [10] 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 test the precision of distributed diagnosis by constructing a Simulation for the distributed diagnosis process.

Second, Grastien et al. [7] 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 tested the diagnosability of an Abstract Model, which ensures the precision of the diagnosis using an Abstract Model. By Theorem 1, if the diagnosis using an Abstract Model is Simulated, then the precision of this diagnosis approach can also be verified.

Third, Cassez and Tripakis [3] studied the sensor minimization problems for both static and dynamic observers. For static observers, the goal is to minimize 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 Simulation is an alternative approach to test the diagnosability.

To sum up, it is feasible to test the precision of the above three approaches of diagnosis by constructing a Simulation for each approach as defined in Section 3.2.

4 INDEPENDENT-WINDOWS ALGORITHMS

We recently proposed [19] the class of Independent-Windows Algorithms (IWAs) for DES. IWAs slice the flow of observations into windows that are diagnosed independently. IWAs aim at improving flexibility (because windows are diagnosed separately) and complexity (because the size of the windows is bounded). However because the links between the observations are lost the precision of the IWAs may be reduced; it may happen for instance that a fault can be diagnosed only by observing two specific events that appear on different windows.

4.1 Principle of IWAs

A window w is a sub-sequence of the actual observations $obs(\sigma)$. The diagnosis of a window w consists in determining whether there exists a nominal system behavior that generates this sub-sequence of observations. In the example of Figure 1 for instance, a window $[a, d, c]$ is symptomatic of a fault.

IWAs create a number of windows that are each diagnosed separately. The diagnosis is the conjunction of each window, i.e., the fault is diagnosed for $obs(\sigma)$ iff the fault is diagnosed for at least one window.

IWAs include several variants that differ in how they build their windows. Al_1 for instance assumes that the observations are sliced into consecutive windows of identical length; for instance, the first window contains the first k observations, the second window contains the observations from number $k + 1$ to $2 \times k$, etc. Al_2 on the other hand makes sure that the windows overlap, so that any sufficiently small group of consecutive observations are guaranteed to appear together in at least one window.

4.2 Simulation of IWAs

We call k-Simulation of M the automaton built as follows (see the illustration for the system of Figure 1 on Figure 2): The set Q of states[2] is copied $k + 1$ times with the initial state being the original initial state; each copy indicates how many observations have been made in the current window. The unobservable transitions are kept within each copy, while the observable transitions move the state from the current copy to the next (hence recording the number of observations). When the last copy has been reached, the state is reset i.e., an unobservable transition is added from any state of the last copy to any state of the first copy; the only restriction is that the fault label is maintained. There is, for instance, a transition from D_4 to A_0 as D and A share the same label, but none from D_4 to F_0 as D and F do not.

Theorem 2 *The k-simulation of a diagnosis model M is the Simulation of Al_1 for window size k and model M as defined in Definition 3.*

Proof (Sketch) Assume that Al_1 diagnoses that no fault occurred. This implies that, for each window, Al_1 was able to find a nominal sequence of events authorized by the model that produces the observations of the window. It is possible to concatenate the parts of these sequences that match the windows' observations with ε transitions, thus producing a sequence of transitions authorized by the k-simulation. Therefore the application of algorithm Δ with the k-simulation as input model will return the same diagnosis (no fault).

On the other hand, assume that the k-simulation is used together with algorithm Δ and returns that no fault occurred. This implies that the k-simulation allows for a sequence of events that contains no fault. That sequence can be split into a sub-sequence for any window, and since each initial state of each sub-sequence is reachable, then a

[2] It is assumed here that all states in Q are reachable from the initial state.

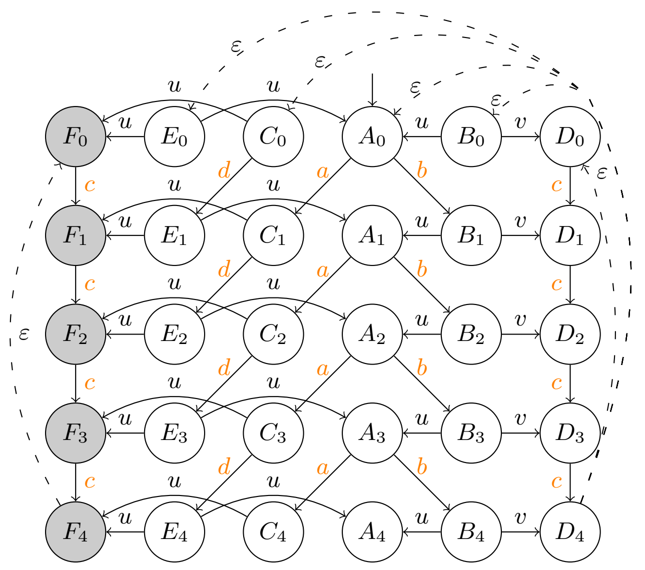

Figure 2. Part of Al_1 simulation for the DES model in Fig. 1. Dotted lines also need to link E_4 to E_0, C_0, A_0, B_0, D_0. Same applies to C_4, A_4, B_4.

prefix can be found in the original model which, when added to the sub-sequence, produces a sequence allowed by the model. Therefore Al_1 will return the same diagnosis (no fault). □

The Simulation of other algorithms in the IWAs family can be built from the k-Simulation. For instance, in the case Al_2, two k-Simulation can be run in parallel one to diagnose the same windows as in Al_1 and the other one to diagnose the windows that overlap with the windows of Al_1.

The k-Simulation is k times bigger than the original model, which means that verifying diagnosability still remains polynomial. Remember however that the k-Simulation is used by the diagnoser and only constructed off-line to test precision. Secondly given a symbolic representation of the model, e.g., as a BDD, then the symbolic representation of the k-Simulation is equally easy to build.

5 CHRONICLE-BASED DIAGNOSIS

Chronicles are collections of events connected by temporal constraints on their occurrence time. In Chronicle-based Diagnosis, Chronicles represent symptoms of failure [5]. A diagnoser is simply a system that recognizes Chronicles in the flow of observations. Chronicles can be either written by hand or generated automatically from a model.

Diagnosability with Chronicles has been studied by Pencolé and Subias [13]. They computed the language associated with each Chronicle and checked whether these languages are exclusive. Diagnosability testing was carried out as exclusiveness tests on the reachability graphs of Time Petri Nets.

We demonstrate that our approach can be used to verify the precision of a set of Chronicles. We show how a Simulation can be built that represents how the Chronicle-based Diagnostic Algorithm works. We build the Chronicle Simulation for a single Chronicle, but the extension to a set of Chronicles can be done by computing the synchronization of these Simulations.

5.1 Chronicle

We define the following fundamental concepts of Time Interval, three operations on Time Intervals, and Chronicle. We also provide the definition for a sequence of events being recognized by a Chronicle.

A Time Interval represents a set of dates when an event is supposed to take place. A Time Interval is written $I = [beginning, end]$, where $beginning \in \mathbf{Z}^\star$ and $end \in \mathbf{Z}^\star$ (i.e., both $+\infty$ and $-\infty$ are also possible values). $\mathbf{I}^+$ is the set of strictly positive time intervals and $\mathbf{I}^-$ is the set of strictly negative time intervals. $\mathbf{I}$ represents a set of time intervals. We also consider a special case of time interval $I_\emptyset$: if $beginning > end$ in a time interval I, then I becomes $I_\emptyset$ meaning that no occurrence time for the associated event will satisfy the constraints on time occurrence. We then define three operations on time intervals.

- Time interval intersection: If $I = [b, e]$ and $I' = [b', e']$, then $I \cap I' = [max(b, b'), min(e, e')]$.
- Time reduction operation: If $I = [b, e]$, then $I - t = [b - t, e - t]$.
- Time disable operation: $I \setminus [b', e']$ means that $[b', e']$ is disabled in the time interval I. This operation could result in disjunctive intervals but it will however be applied in situations where this does not happen.

Dechter et al. [4] 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. STP can be solved by a *Directed Edge-Weighted Graph*, or a *Distance Graph* [4]. Each node represents an event and each edge represents the distance between the time points of two events. In order to construct a minimal distance graph, we apply *Floyd-Warshall*'s all-pairs-shortest-paths-algorithm to a distance graph. The complexity is $O(n^3)$ and n is the number of nodes [4].

A *Chronicle* is a tuple $\langle N, EL, B\rangle$ where N is a finite set of *nodes* in a STP model, $EL : N \to \Sigma$ is an *event label function*, $B : N \times N \to \mathbf{Z} \times \mathbf{Z}$ is a *binary constraint function* for a pair of nodes.

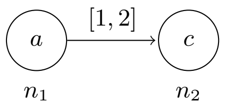

Figure 3. Chronicle 1 represented by a time constraint graph: if there is a c event within one or two time steps after an a, then the system is faulty.

Figure 3 shows Chronicle 1 ($Ch1$) represented by a time constraint graph [4]. $Ch1$ is a Chronicle for the DES model in Figure 1 that can diagnose the fault. Notice that, as opposed to the original work on Chronicles, we define the time constraints in terms of number of observations between two observed events rather than the actual time between the event occurrences.

Given a sequence of observations $o_1, o_2, \ldots, o_k$ and a Chronicle $Ch = \langle N, EL, B\rangle$, the Chronicle is recognized in the sequence of observations iff $\exists f : N \to \{1, 2, \ldots, k\}$ such that $\forall n \in N.\ \forall n' \in N.\ EL(n) = o_{f(n)} \wedge f(n') - f(n) \in B(\langle n, n'\rangle) \wedge n \neq n' \Rightarrow f(n) \neq f(n')$. Therefore, Ch is recognized in the sequence of observations iff there exists a function f that index each node of the chronicle with an event of the flow, and these events satisfy the constraints. Chronicle-based Diagnosis checks whether the chronicle is recognized in the observations, and return faulty iff it is.

5.2 Chronicle automaton and chronicle simulation

We first build a non-deterministic finite state machine called a Chronicle Automaton; the Simulation will be the determinisation of this Chronicle Automaton. Each state of the Chronicle Automaton stores the list of observable events of the Chronicle that have been recognized so far, and when the next events are expected to occur. To this end, we define *Partially Recognized Chronicle* ($PRCh$) and *Fully Recognized Chronicle* ($FRCh$).

A *Partially Recognized Chronicle* ($PRCh$) is a tuple $\langle Ch, U\rangle$ where Ch is a Chronicle $\langle N, EL, B\rangle$ and $U : N \to \{I_R, I_\emptyset\} \cup \mathbf{I}^+$ is a *unary constraint function* for a node. U defines a time interval that specifies when a node is expected to be recognized.

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 the other event occurrence has been stored in U); consequently, $U(n)$ is then replaced by $I_R = (-\infty, 0]$.

A *Fully Recognized Chronicle* ($FRCh$) is a Partially Recognized Chronicle $\langle Ch, U\rangle$ such that $\forall n \in N.\, U(n) = I_R$.

A *Chronicle Automaton* (CA) is an Automaton, which is a tuple $\langle Q, \Sigma, T, I, L\rangle$ where $Q = \{PRCh\}$ is the set of PRCh, Σ is a finite set of *events*, $T \subseteq Q \times \Sigma \times Q$ is the set of *transitions* defined below, $I = \{A\}$ defined below, $L(q) = F$ iff $q = FRCh$.

In the initial state A, 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 final states are $FRCh$ such that $\forall n \in N.\, U(n) = I_R$.

The transition function is presented in Algorithm 1. The Chronicle Automaton is non deterministic, i.e., when an observable event of the Chronicle is observed, it is unknown whether it is part of the Chronicle recognition or not. To simplify the definition of the transition, we assume however that the events of the Chronicle are replaced with the node during the construction of the automaton; in the example of Figure 3, the set of events of the Chronicle Automaton would be $\{a, b, c, d, n_1, n_2\}$, where the first four events correspond to the observations being not part of the recognition of the Chronicle, and n_1, n_2 are the nodes in the Chronicle. For instance, observing 'a' in the observations '$abbbac$' may not mean observing the event of the node n_1. The event of n_1 is recognized at the fifth step and the event of n_2 is recognized at the sixth.

Algorithm 1 first acknowledges that a new observation has been made by reducing all intervals by one. Then the node corresponding to the observation made is set to zero if such a node exists and if this is possible. The Floyd-Warshall's algorithm ($update\_constraint$) is then applied. Finally the details of when the observations were made are ignored. 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.

The Chronicle Automaton is built by iteratively computing the successors of all the PRCh found, starting with the initial state. Figure 4 shows the Chronicle Automaton of $Ch1$ where the events have been renamed.

The Chronicle Automaton is not the Simulation of the Chronicle. Indeed their semantics are different: in the Chronicle Automaton, the fault is diagnosed if there exists a path consistent with the observations that leads to a faulty state; in the Simulation, all such paths should lead to a faulty state. Therefore the Simulation is the determinization of the Chronicle Automaton.

Algorithm 1: build_one_transition

```
   Input: $PRCh$ and $e$
   Output: $PRCh'$
1  $U_1 := U - 1$
2  $U_2 : N \to \mathbf{I}$ such that $\forall n \in N$, if $n = e$ then
3      $U_2(n) := U_1(n) \cap [0, 0]$
4  else
5      $U_2(n) := U_1(n) \setminus [0, 0]$
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      $U_4(n) := I_R$
9  else
10     $U_4(n) := U_3(n)$
11 return $\langle Ch, U_4\rangle$
```

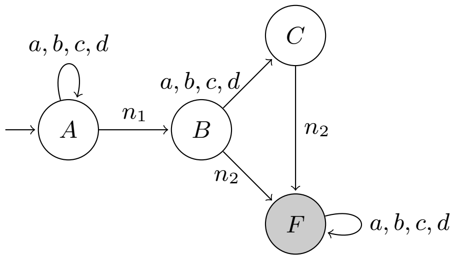

Figure 4. Chronicle Automaton for the Chronicle 1 in Figure 3: n_1 can be replaced with a and n_2 with c because only n_1 and n_2 are the nodes in the Chronicle and the other events are not part of the recognition of the Chronicle.

Theorem 3 *The determinization of the Chronicle Automaton of a Chronicle is the Simulation for Chronicle-based Diagnosis.*

Proof (Sketch) Assume that the Chronicle is recognized. Then one can use the f recognition function defined before to label the observations. When following the single path on the Chronicle Automaton labeled by this new sequence, we reach by construction a state that is labeled faulty. Hence the corresponding path in the determinized automaton leads to a faulty state.

Conversely, any path on the Chronicle Automaton that leads to a faulty state defines the f recognition function.

□

Finally, we examine the drawback of generating Chronicle Automaton and a Simulation for Chronicle-based Diagnosis. In the worst case, the number of states in a Chronicle Automaton is $O(|e|^r)$ where $|e|$ is the number of events, $[beginning, end]$ is the widest binary constraint in a given chronicle and $r = end - beginning + 2$. Nevertheless, after a transition is computed, any invalid PRCh will be removed immediately due to the time inconsistency of $I_\emptyset$. Therefore, this time consistency checking reduces the complexity compared to the worst case. Also, it should be noted that for a given chronicle, the construction for a Chronicle Automaton and a Simulation is off-line and one-off computation.

6 IMPLEMENTATION

In order to build the Simulation for Al_1 and Al_2 of IWAs (Section 4.2), we use BDD as the data structure for DES. We implement BDD using JDD, a Java library for creation and operations on BDD variables[3]. First, the input DES model is in des_comp format according to the Dia-Des project of Yannick Pencolé[4]. Second, we use BDD variables for every state, event, and then build transitions. If there are multiple automatons, we synchronize them on the shared events. Next, we make a copy of the automaton, unfold that copy to build the Simulation, and build a twin plant using the automaton and the unfolded copy. Finally, we implement the Forward Algorithm of symbolic model checking to test diagnosability [6].

The experiments show that IWA Al_1 with any k is not precise for the system in Figure 1; on the other hand, IWA Al_2 is precise for any $k \geq 2$. For instance, if $k = 4$ and we want to diagnose this system given a sequence of observations b, b, a, d, c, c, c, c, IWA Al_2 is able to precisely diagnose, but IWA Al_1 is not.

For Chronicle-based Diagnosis, we implement Algorithm 1 (Section 5.2) in Java to build a Chronicle Automaton. After that, a Simulation is a deterministic Chronicle Automaton. One way to compute a deterministic automaton is to use the AT&T Finite-State Machine Library[5]. Our algorithms for instance show that a Chronicle-based diagnoser with $Ch1$ is precise.

7 CONCLUSION

Computing the precision of a diagnostic algorithm was an unsolved problem. This paper presents a method to decide whether precision is maintained if an incomplete diagnostic algorithm is used for DES diagnosis. To this end, we define the Simulation, which is a modification of the model that simulates the process of a diagnostic algorithm generating a diagnostic result. We then use the twin plant method to decide whether diagnosability is maintained despite the imprecision of the diagnostic algorithm.

We illustrate the benefits of our approach on IWAs and Chronicle-based Diagnosis. IWAs [19] do not carry all the historical information about the system execution. Chronicle-based Diagnosis [5] uses pattern recognition techniques. These algorithms however are less precise than the perfect model-based algorithm. We test the precision and demonstrate the correctness of our approach in Theorem 2 and Theorem 3.

In conclusion, we would like to see the approach presented in this paper to be adopted by other researchers for testing the precision of their approach. We believe that this theoretical framework will also allow researchers to consider more aggressive approaches to reduce complexity, as they can now assess the precision of these approaches. Finally we would like to extend this theory to the case where the system is not diagnosable: how can we prove that an imprecise algorithm does not reduce the precision of diagnosis.

ACKNOWLEDGEMENTS

NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

The authors also want to thank the reviewers for their constructive comments.

[3] javaddlib.sourceforge.net/jdd/
[4] homepages.laas.fr/ypencole/diades/html/index.html
[5] http://www2.research.att.com/∼fsmtools/fsm/

REFERENCES

[1] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella, 'Diagnosis of large active systems', *Artificial Intelligence (AIJ)*, **110**(1), 135–183, (1999).
[2] C. Cassandras and S. Lafortune, *Introduction to discrete event systems (2nd ed.)*, Kluwer Academic Publishers, 2008.
[3] F. Cassez and S. Tripakis, 'Fault diagnosis with static and dynamic observers', *Fundamenta Informaticae*, **88**(4), 497–540, (2008).
[4] R. Dechter, I. Meiri, and J. Pearl, 'Temporal constraint networks', *Artificial Intelligence (AIJ), Special Vol. on Knowledge Representation*, **49**(1–3), 61–95, (1991).
[5] C. Dousson, 'Alarm driven supervision for telecommunication networks: II on-line chronicle recognition', *Annals of Telecommunications (AOT)*, **51**(9–10), 501–508, (1996).
[6] A. Grastien, 'Symbolic testing of diagnosability', in *20th International Workshop on Principles of Diagnosis (DX-09)*, pp. 131–138, (2009).
[7] A. Grastien and G. Torta, 'A theory of abstraction for diagnosis of discrete-event systems', in *9th Symposium on Abstraction, Reformulation and Approximation (SARA-11)*, pp. 50–57, (2011).
[8] T. Jéron, H. Marchand, S. Pinchinat, and M.-O. Cordier, 'Supervision patterns in discrete event systems diagnosis', in *8th International Workshop on Discrete Event Systems (WODES-06)*, pp. 262–268, (2006).
[9] S. Jiang, Z. Huang, V. Chandra, and R. Kumar, 'A polynomial algorithm for diagnosability of discrete-event systems', *IEEE Transactions on Automatic Control (TAC)*, **46**(8), 1318–1321, (2001).
[10] P. Kan John, A. Grastien, and Y. Pencolé, 'Synthesis of a distributed and accurate diagnoser', in *21st International Workshop on Principles of Diagnosis (DX-10)*, pp. 209–216, (2010).
[11] G. Lamperti and M. Zanella, 'On monotonic monitoring of discrete-event systems', in *18th International Workshop on Principles of Diagnosis (DX-07)*, pp. 130–137, (2007).
[12] Y. Pencolé and M.-O. Cordier, 'A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks', *Artificial Intelligence (AIJ)*, **164**(1–2), 121–170, (2005).
[13] Y. Pencolé and A. Subias, 'A chronicle-based diagnosability approach for discrete timed-event systems: application to web-services', *Journal of Universal Computer Science (JUCS)*, **15**(17), 3246–3272, (2009).
[14] J. Rintanen, 'Diagnosers and diagnosability of succinct transition systems', in *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pp. 538–544, (2007).
[15] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, 'Diagnosability of discrete-event systems', *IEEE Transactions on Automatic Control (TAC)*, **40**(9), 1555–1575, (1995).
[16] A. Schumann, Y. Pencolé, and S. Thiébaux, 'A spectrum of symbolic on-line diagnosis approaches', in *22nd Conference on Artificial Intelligence (AAAI-07)*, pp. 335–340, (2007).
[17] R. Su and W. Wonham, 'Global and local consistencies in distributed fault diagnosis for discrete-event systems', *IEEE Transactions on Automatic Control (TAC)*, **50**(12), 1923–1935, (2005).
[18] R. Su and W. Wonham, 'Hierarchical fault diagnosis for discrete-event systems under global consistency', *Journal of Discrete Event Dynamic Systems (JDEDS)*, **16**(1), 39–70, (2006).
[19] X. Su and A. Grastien, 'Diagnosis of discrete event systems by independent windows', in *24th International Workshop on Principles of Diagnosis (DX-13)*, pp. 148–153, (2013).
[20] T. Yoo and S. Lafortune, 'Polynomial-time verification of diagnosability of partially observed discrete-event systems', *IEEE Transactions on Automatic Control (TAC)*, **47**(9), 1491–1495, (2002).