Combining Enumerative and Symbolic Techniques for Diagnosis of Discrete-Event Systems

Abderraouf Boussif, Mohamed Ghazel, Kais Klai

Combining Enumerative and Symbolic Techniques for Diagnosis of Discrete-Event Systems

Abderraouf Boussif
Univ. Lille Nord de France,
F-59000 Lille, France
IFSTTAR, Cosys/Estas,
F-59650 Villenveuve d'Ascq, France
FR
abderraouf.boussif@ifsttar.fr

Mohamed Ghazel
Univ. Lille Nord de France,
F-59000 Lille, France
IFSTTAR, Cosys/Estas,
F-59650 Villenveuve d'Ascq, France
FR
mohamed.ghazel@ifsttar.fr

Kais Klai
LIPN, CNRS UMR 7030,
Univ. Paris 13, Sorbonne Paris Cité,
FR
kais.klai@lipn.univ-paris13.fr

In this paper, an efficient approach to verify diagnosability of discrete-event systems is proposed. The approach consists in constructing a hybrid diagnoser based on the symbolic observation graph (SOG), which is a technique that combines symbolic and enumerative representations in order to build a deterministic observer from a partially observed model. The construction of the diagnoser as well as the verification of diagnosability are performed simultaneously on-the-fly, which can considerably reduce the generated state space of the diagnoser and thus the overall running time. Furthermore, the proposed approach provides a heuristic strategy in order to converge fast into the necessary part, of the diagnoser, for analysing diagnosability.

Discrete-Event Systems, Diagnosability Analysis, Symbolic Observer Graph, On-the-Fly Verification.

1. INTRODUCTION

In automated monitoring and fault diagnosis of complex dynamic systems, one of the central tasks is to detect and identify the occurrence of failures as early as possible. This task has become an active research area in recent years (Zaytoon and Lafortune 2013). From the theoretical point of view and at a high level of abstraction, Discrete-Event Systems (DESs) are more suitable for performing diagnosis analysis on complex systems (Cassandras and Lafortune 2007).

One of the main issues in diagnosis activity that must be addressed is diagnosability investigation. Analysing diagnosability of a system intends to determine accurately whether any predetermined failure or class of failures can be detected and identified within a finite delay following its occurrence (Sampath et al. 1995).

Diagnosability verification has received considerable attention since the seminal paper by (Sampath et al. 1995), which provides a basic concept and a formal definition of diagnosability analysis and fault diagnosis of DESs that were adopted and further developed later. In this paper, (Sampath et al. 1995), the original definition of diagnosability was introduced in the language context. A systematic method to check diagnosability based on a dedicated deterministic version of the model derived from the original system, a so-called *diagnoser*, was also provided. It consists of a specific observer of the system associated with a labelling function that attributes to each state (or macro-state), in this observer, a label indicating whether the state is reached by a faulty execution or not, i.e. an execution where some particular unobservable events, called *faults*, have occurred or not.

Other automata-based approaches (Jiang et al. 2001; Yoo and Lafortune 2002), aiming to reduce computational complexity have been then proposed. In (Yoo and Lafortune 2002), a polynomial-time algorithm for checking diagnosability based on a structure called *verifier* is adopted. In (Jiang et al. 2001), an algorithm based on the twin plant structure (a parallel composition of the investigated automaton with itself) is proposed. Reformulations of these works in model-checking framework were first proposed in (Cimatti et al. 2003) and extended in (Boussif and Ghazel 2015). The goal is to check

diagnosability property in the same way as to check any safety property.

Furthermore, some works on diagnosability of DESs turned to Petri nets (PNs) formalism, benefiting from the mathematical and graphical representations capability and the well-developed theory underlying PNs (Peterson 1981). (Ushio et al. 1998) extended Sampath's study to systems modelled by PNs with the assumption that some places are observables whereas all of the transitions are unobservable. A diagnoser is constructed from the reachability graph. In (Wen and Li 2005), the authors proposed a sufficient condition for testing diagnosability by checking the structure of T-invariants of a PN. In (Cabasino et al. 2009) the modified basis reachability graph (MBRG) and basis reachability diagnoser (BRD), which provide a compact representation of the reachability graph, were developed. In (Basile et al. 2012), an approach for checking diagnosability by quantifying the finite delay of diagnosability (the so-called K-diagnosability) was proposed by using the integer linear programming (ILP) technique. A structure called verifier net (VN) was introduced in (Cabasino et al. 2014) to deal with diagnosability for both bounded and unbounded PNs. Recently, (Liu et al. 2014a) has proposed an on-the-fly and incremental diagnosis technique to construct a diagnoser from a bounded PN in order to verify diagnosability and K-diagnosability properties.

To get a general overview on the literature pertaining to diagnosis of DESs, the reader can refer to the recent survey in (Zaytoon and Lafortune 2013), where theoretical and practical issues, tools and other issues in relation with diagnosis are discussed.

The challenge of analysing diagnosability is the combinatorial explosion problem that appears during the building of the intermediate models (diagnoser, verifier, twin plant, MBRG, etc.). This is due to the high complexity of these constructed models and to the generation of the whole state-space which may have considerable time and memory cost.

To partially overcome this problem, we propose, in this paper, an efficient approach to construct a hybrid diagnoser on-the-fly, in the sense of combining enumerative and symbolic techniques. The contributions of this paper are twofold:

1. We provide a behavioural diagnoser based on the Symbolic Observation Graph (Haddad et al. 2004) which is an efficient binary decision diagram (BDD) based abstraction of the model state space. Thus, macro-states of the diagnoser will be compacted using BDDs while transitions between macro-states are represented by enumerate observable events.
2. We design an appropriate algorithm, for simultaneously constructing the diagnoser and checking diagnosability on-the-fly. Actually, the verification process is stopped (only a part of the diagnoser is built) as soon as the diagnosability is proven to be unsatisfied, which can considerably reduce the generated state space of the diagnoser. Furthermore, the proposed algorithm is endowed with a heuristic strategy in order to converge fast into the necessary part of the diagnoser for diagnosability analysis.

The proposed work is related to the CARONTE FP7 project that focussing on the security of land transportation systems. In particular, the technique discussed here can be used to tackle, especially, cyber attacks which may target railway operation management systems.

The paper is structured as follows. In Section 2, we introduce the basic background needed to deal with diagnosability and to develop our approach. In Section 3, we recall the notion of diagnosability as well as the original diagnoser approach. Section 4 is devoted to discuss the Symbolic Observation Graph adapted to the context of this paper. In Section 5, the verification approach is sketched out then an on-the-fly algorithm based on the SOG is presented. Section 6 discusses the pertinent existing work in relation with the present work. Finally, conclusion remarks and future research directions are given in Section 7.

2. PRELIMINARIES

We first recall some standard notations that will be used in the sequel. Let Σ be a finite alphabet of events (actions). A string is a finite sequence of events in Σ. ϵ denotes the empty string. Given a string s, the length of s is denoted by $|s|$. The set of all strings formed by events in Σ is denoted by Σ^*. Any subset of Σ^* is called a *language*. Given a string $s \in L$, $L/s \triangleq \{t \in \Sigma^* | s.t \in L\}$ is called the *post-language* of L after s and defined as L/s. L is said to be *extension-closed* when $L.\Sigma^* = L$.

The approach introduced in this paper applies to discrete-events systems modelled by Labelled Transitions Systems (LTSs for short). The formal definition of LTS is as follows.

Definition 1 *(LTS): An LTS over Σ is defined by a 4-tuple $\langle Q, \Sigma, \rightarrow, q_0 \rangle$, where:*

- *Q is a finite set of states;*
- *Σ is a finite set of events;*
- *$\rightarrow \subseteq Q \times \Sigma \times Q$ is the transition relation;*
- *$q_0 \in Q$ is the initial state.*

In the remainder of this section, we consider a given LTS $G = \langle Q, \Sigma, \rightarrow, q_0 \rangle$. For $q, q' \in Q$ and $\sigma \in \Sigma$, we denote $q \xrightarrow{\sigma} q' \triangleq (q, \sigma, q') \in \rightarrow$. $q \rightarrow$ means that $\exists q' \in Q : q \xrightarrow{\sigma} q'$. If $s = \sigma_1, \sigma_2, \ldots, \sigma_n$ is a string (sequence of events), $\bar{s}$ denotes the set of actions occurring in s. Moreover, $q \xrightarrow{s} q'$ denotes that $\exists q_1, q_2, \ldots, q_{n-1} \in Q$ such that, $q \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \ldots q_{n-1} \xrightarrow{\sigma_n} q'$. $q \xrightarrow{*} q'$ denotes that q' is reachable from q (i.e. $q \xrightarrow{s} q'$ for some $s \in \Sigma^*$), and $q \xrightarrow{*}_E q'$ holds if $\bar{s} \subseteq E$.

We denote by $Enable(q)$ the set of events σ s.t. $q \xrightarrow{\sigma}$, for a set of states $Q' \subseteq Q$, $Enable_E(Q')$ denote the set of enabled events from the set of states Q', i.e. $Enable(Q')$ denotes $\bigcup_{q \in Q'} Enable(q)$.

An execution from the initial state q_0 of an LTS G is a finite sequence of transitions $\pi = q_0 \xrightarrow{\sigma_1} q_1 \ldots \xrightarrow{\sigma_n} q_n$. The event-trace of π, denoted by $Tr(\pi)$, is the sequence of events $\sigma_1, \ldots, \sigma_n$, $\pi[i]$ stands for the prefix of π truncated at state q_i, i.e., $\pi[i] = q_0 \xrightarrow{\sigma_1} q_1 \ldots \xrightarrow{\sigma_i} q_i$ and $last(\pi)$ represents the last state of π. The set of finite executions of LTS G from the initial state q_0 is denoted by $Runs(G)$. The behaviour of G is described by its language $L(G) = \{s \in \Sigma^* | q_0 \xrightarrow{s}\}$.

As we are interested in diagnosis issues, partial observation plays a central role. In this regard, some events in Σ are observable, i.e. their occurrence can be observed, while the rest are unobservable. Thus, the event set Σ can be partitioned as $\Sigma = \Sigma_o \biguplus \Sigma_u$, where Σ_o denotes the set of observable events and Σ_u the set of unobservable events. To reflect the limitation on observation, we define the projection function $P : \Sigma^* \rightarrow \Sigma_o^*$. In the usual manner, $P(\sigma) = \sigma$ for $\sigma \in \Sigma_o$; $P(\sigma) = \epsilon$ for $\sigma \in \Sigma_u$, and $P(s\sigma) = P(s)P(\sigma)$, where $s \in \Sigma^*$, $\sigma \in \Sigma$. Thus, P simply erases the unobservable events in any event-trace. The inverse projection operation P_L^{-1} is defined by $P_L^{-1}(y) = \{s \in L(G) : P(s) = y\}$. Any two executions π and π' are called *indistinguishable* with respect to the projection function P if they can generate the same observed event-trace. With a slight abuse of notation, we write $P(\pi) = P(\pi')$ if π and π' are *indistinguishable*.

In the context of fault diagnosis, let $\Sigma_f \subseteq \Sigma_u$ denote the set of failure events. They are usually represented using unobservable events, since their detection and diagnosis would be trivial if they were observable. We partition the set of failure events into disjoint failure classes $\Sigma_f = \Sigma_{f_1} \biguplus \Sigma_{f_2} \biguplus \cdots \biguplus \Sigma_{f_m}$, with Σ_{f_i} denotes the failure class f_i.

3. DIAGNOSABILITY ANALYSIS

In this work, only diagnosability analysis of permanent faults is considered. Once a fault has occurred, the system remains irreparably faulty. We assume that the LTS G under consideration satisfies the following two assumptions:

1. The language generated by G is live, i.e. there is an executable transition from any state of the system.
2. The LTS G is finite, in term of the state space, and does not contain cycles formed only by unobservable events.

3.1. Definition of diagnosability

Diagnosability is an important property in the monitoring and fault diagnosis activities. In simple terms, it refers to the ability to infer accurately, from a partially observed execution, about the faulty behaviour within a finite delay after possible occurrences of faults. The original definition of diagnosability, introduced in the seminal work of (Sampath et al. 1995) is recalled in the following.

Definition 2 *(diagnosability (Sampath et al. 1995)) A prefix-closed and live language L is said to be diagnosable with respect to the projection function P and with respect to a failure class of faults Σ_f if the following holds*

$$(\exists n_i \in \mathbb{N})\ [\forall s \in \Psi(\Sigma_f)]\ (\forall t \in L/s)\ [|t| \geq n_i \Rightarrow D]$$

where the diagnosability condition D is

$$\omega \in P^{-1}[P(s.t)] \Rightarrow \Sigma_f \in \omega.$$

with $\Psi(\Sigma_f)$ is the set of finite sequences that terminate with a faulty event from Σ_f.

The above definition means the following. Let s be any sequence generated by the LTS G that ends with a failure event from Σ_f, and let t be any sufficiently long continuation of s. Condition D then requires that every sequence ω belonging to the language that produces the same observable trace as sequence $s.t$ ($P(\omega) = P(s.t)$) must hold a failure event from Σ_f. In other terms, diagnosability requires that every failure event leads to observations distinct enough to identify the failure type within a finite delay.

3.2. Verification of diagnosability

In order to analyse diagnosability, (Sampath et al. 1995) has proposed a systematic approach that consists in building a specific model called *diagnoser*. It is a deterministic automaton whose transitions correspond to the observable events of the system and whose states are estimation system state associated with labels to indicate if a state is reached by an observable trace containing a faulty event or not.

Definition 3 *(Diagnoser (Sampath et al. 1995))*

Let $G = \langle Q, \Sigma, \rightarrow, q_0 \rangle$ be an LTS to be diagnosed. A diagnoser of G is a deterministic LTS $G_d = \langle \mathcal{X}, \Sigma_o, \rightarrow_d, \mathcal{X}_0 \rangle$ associated with a tagging function $Diag : \mathcal{X} \rightarrow 2^{\triangle}$, with $\triangle = \{N, F\}$ (for only one class of failures). with N means normal and F means faulty.

Each diagnoser state x has the form $x = \{(q_1, l_1), \ldots, (q_n, l_n)\}$, with $q_i \in Q$ and $l_i \in \triangle$. If $\forall i = 1, \ldots, n$, we have $l_i = N$ (resp. $l_i = F$), the diagnoser state x is said to be $N\text{-}certain$ (resp. $F\text{-}certain$), otherwise $F\text{-}uncertain$ state.

For more details about the formal framework and algorithmic procedure of constructing the diagnoser, we refer the reader to the original paper (Sampath et al. 1995).

We define an f-indeterminate cycle in a diagnoser to be a cycle composed exclusively of $F\text{-}uncertain$ diagnoser states and corresponding to the presence of two cycling traces, in the system, that sharing the same observable events, such that the faulty event f from the class Σ_f occurs in the 1^{st} trace but not in the 2^{nd}. The notion of f-indeterminate cycle is very important, since it helps to give a necessary and sufficient condition for diagnosability analysis.

Theorem 1 *((Sampath et al. 1995))*

A system modelled by an LTS G is diagnosable if and only if there are no f-indeterminate cycles in its diagnoser G_d for any class of faults Σ_f.

3.3. Example

Let us consider the LTS G in Figure 1 (adapted from (Sampath et al. 1995)). The set of observable events is $\Sigma_o = \{a, b, d, t\}$ and the set of unobservable events is $\Sigma_u = \{u, f\}$ with f a faulty event in Σ_f.

The diagnoser G_d corresponding to the LTS G is depicted in Figure 2. There exists a cycle of f-uncertain states composed of $\{3F, 7N\}$ and $\{5F, 10F, 12N\}$ w.r.t. the observable sequence $(bd)^*$.

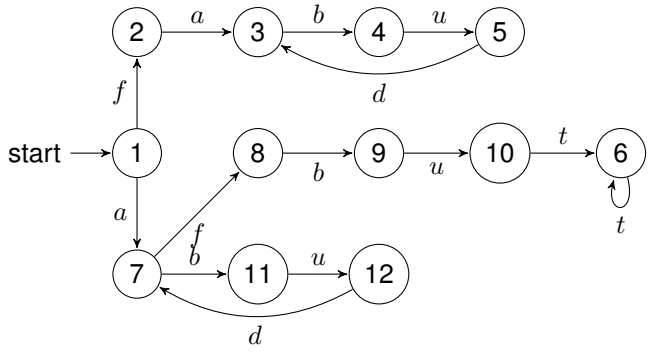

Figure 1: The LTS G

This cycle corresponds to two cycles in the LTS G. The 1^{st} one, which is composed of states $\{3\}, \{4\}, \{5\}$ w.r.t. the sequence $(bud)^*$, which is reachable from the faulty sequence fa. The 2^{nd} cycle, which is composed of states $\{7\}, \{11\}, \{12\}$ and reached by the fault-free sequence a. Thus we can infer, according to Theorem 1, that there exists an f-indeterminate cycle in the diagnoser and consequently the LTS G is not diagnosable w.r.t. to faulty class Σ_f and the projection function P.

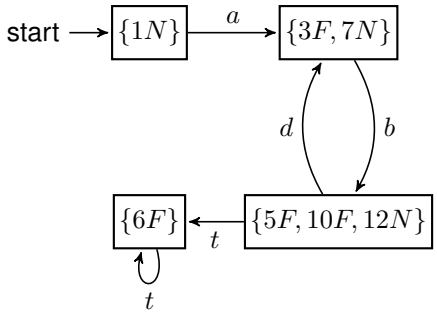

Figure 2: Diagnoser G_d of the LTS G

It is worth noticing that the diagnoser can be used either off-line to check diagnosability or on-the-fly by connecting it to the system in order to provide on-line diagnosis upon the occurrence of observable events.

4. SYMBOLIC OBSERVATION GRAPH (SOG)

In this section, we present the so-called *symbolic observation graph* (Haddad et al. 2004) and we show how it is used to abstract LTS behaviour. In (Haddad et al. 2004), the authors have introduced the SOG as an abstraction of the reachability graph of concurrent systems and showed that the verification of an event-based formula of LTL/X on the SOG is equivalent to the verification on the original reachability graph. The construction of the SOG is guided by the set of observable events.

The SOG is defined as a graph where each node is a set of states linked by unobserved events and each arc is labelled with an observable event. The SOG nodes are called *aggregates* and may be represented and handled efficiently using decision diagram techniques (BDDs, see for instance (Bryant 1992)). The SOG is said to be hybrid since it is both an explicit and a symbolic structure: the graph is represented explicitly while the nodes are sets of states encoded and managed symbolically. Despite the exponential theoretical complexity of the size of a SOG, it has a very moderate size in practice (see (Haddad et al. 2004; Klai and Petrucci 2008); (Klai and Poitrenaud 2008) for experimental results).

In the following, we first define what an aggregate is formally, before providing a formal definition of a SOG associated with an LTS.

Definition 4 *(aggregate)*

Consider the LTS $G = \langle Q, \Sigma, \rightarrow, q_0 \rangle$ *with* $\Sigma = \Sigma_o \biguplus \Sigma_u$. *An aggregate* a *is a non empty set of states satisfying:* $q \in a \Leftrightarrow Saturate_{\Sigma_u}(q) \subseteq a$*; where* $Saturate_{\Sigma_u}(q) = \{q' \in Q | q \stackrel{*}{\rightarrow}_{\Sigma_u} q'\}$.

In the following, $Saturate_{\Sigma_u}$ is extended to sets of states as follows: $Saturate_{\Sigma_u}(Q') = \bigcup_{q \in Q'} Saturate_{\Sigma_u}(q)$.

Definition 5 *(symbolic observation graph)*

The deterministic symbolic observation graph SOG(G*) associated with an LTS* G *is an LTS* $\langle \mathcal{A}, \Sigma_o, \rightarrow_{\Sigma_o}, a_0 \rangle$ *where:*

1. $\mathcal{A}$ *is a finite set of aggregates such that:*
   a) *There is an aggregate* $a_0 \in \mathcal{A}$ *s.t.* $a_0 = Saturate_{\Sigma_u}(q_0)$*;*
   b) *For each* $a \in \mathcal{A}$ *and for each* $\sigma \in \Sigma_o$*, if* $\exists q \in a, q' \notin a$*:* $q \stackrel{\sigma}{\rightarrow} q'$ *then* $Saturate(\{q' \notin a | \exists q \in a, q \stackrel{\sigma}{\rightarrow} q'\})$ *equals* a' *for some aggregate* a' *and* $(a, \sigma, a') \in \rightarrow_{\Sigma_o}$*;*
2. $\rightarrow_{\Sigma_o} \subseteq \mathcal{A} \times \Sigma_o \times \mathcal{A}$ *is the transition relation, obtained by applying* $1.b)$.

The SOG can be constructed by starting with the initial aggregate a_0 and iteratively adding new aggregates as long as the condition of $1.b)$ holds (see (Haddad et al. 2004) for a construction algorithm).

5. USING SOGS TO ANALYSE DIAGNOSABILITY

In this section, we discuss how the SOG is used in order to build a hybrid diagnoser and we provide an on-the-fly algorithm to construct the hybrid diagnoser and to verify diagnosability simultaneously.

The underlying idea behind using SOG for diagnosability analysis is basically to tackle the state explosion phenomenon raised when building the diagnoser. It is worth recalling here that the Sampath's diagnoser is computed with an exponential complexity regarding the state space of the model (Sampath et al. 1995). Obviously, this represents a serious limit of the diagnoser based approach when large models are handled.

Using the results of Section 4, we would use the SOG to perform a hybrid diagnoser construction. In order to capture the feature of analysing diagnosability which is tracking the ambiguous behaviour of the system, i.e. normal and faulty executions which share the same observable events, we modify the structure of the aggregate, introduced in Definition 4, by splitting the set of states (managed using BDDs) into tow sets of states in the same aggregate: one set contains normal states, i.e. states reachable by fault-free sequences, and the other contains faulty states, i.e. states reachable by sequences containing one (or more) faulty events. Both of sets are represented and managed using BDDs.

Figure 3 depicts the general form of a diagnoser aggregate where two BDDs represent two sets of states; BDD_n represents the set of normal states, while BDD_f represents the set of faulty ones. The set of faulty states may be reached from the set of normal states by the occurrence of a faulty event, and thus a faulty transition form BDD_n to BDD_f may exist in any diagnoser aggregate. Depending on the executed behaviour, i.e. the executed sequence, the diagnoser aggregate may contain the two sets of states (BDDs) or only one set. If a diagnoser aggregate contains only BDD_n (resp. BDD_f), it is called an $N\text{-}certain$ (resp. $F\text{-}certain$) diagnoser aggregate, else it is an $F\text{-}uncertain$ diagnoser aggregate in the same way as in the classic diagnoser (Definition 3).

The dashed arrows show the different possibilities that a transition from a diagnoser aggregate can produce. For instance, observable event b enabled by the diagnoser aggregate can be enabled from both normal and faulty sets or from only one set. This feature will be considered during the on-the-fly construction of our hybrid diagnoser, since we do not need to construct the diagnoser aggregates reached through an observable event from only the faulty set of an aggregate. Moreover, this information will be used to establish some heuristics that prioritizing the branches to be followed while building the hybrid diagnoser.

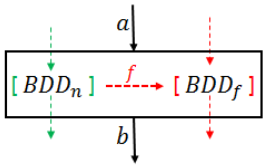

Figure 3: A diagnoser aggregate

Definition 6 *(diagnoser aggregate)*

Consider an LTS $G = \langle Q, \Sigma, \rightarrow, q_0 \rangle$ *with* $\Sigma = \Sigma_o \biguplus \Sigma_u$ *and* $\Sigma_f \subset \Sigma_u$*. A diagnoser aggregate* $a = \langle Q_n, Q_f \rangle$ *is a non empty set of states satisfying:*

1. $\forall q \in Q$ *s.t.* $q_0 \xrightarrow{*f*} q$ *(i.e.* q *is reachable by a faulty sequence):* $q \in a \Leftrightarrow Saturate_{\Sigma_u}(q) \subseteq a.Q_f$*;*

2. $\forall q \in Q$ *s.t.* $q_0 \xrightarrow{*}_{\Sigma \setminus \Sigma_f} q$ *(i.e.* q *is reachable by a fault-free sequence):* $q \in a \Leftrightarrow Q' = Saturate_{\Sigma_u \setminus \Sigma_f}(q) \subseteq a.Q_n \wedge Saturate_{\Sigma_u}(Img(Q', f) \subseteq a.Q_f$*.*

3. $\forall q, q' \in a, \exists s, s' \in \Sigma^*$*, s.t.* $q_0 \xrightarrow{s} q$*,* $q_0 \xrightarrow{s'} q'$*, and* $P(s) = P(s')$

with $Saturate_{\Sigma_u \setminus \Sigma_f}(q) = \{q' \in Q | q \rightarrow_{\Sigma_u \setminus \Sigma_f} q'\}$*, and* $Img(Q', f) = \{q' | q \in Q' : q \xrightarrow{f} q'\}$*, i.e. it returns the set of immediate successors of states in* Q' *through the occurrence of event* f*.*

To simplify the notation, we denote by $a.Q_n$ *(resp.* $a.Q_f$*) the set of normal (resp. faulty) states in an aggregate* a*.*

5.1. Constructing the hybrid diagnoser

We now introduce the hybrid diagnoser which is a modified SOG built from the LTS model G.

Definition 7 *(hybrid diagnoser)*

The hybrid diagnoser $D_{SOG}(G)$ *associated with an LTS* G *is a modified SOG* $\langle \Gamma, \Sigma_o, \rightarrow_{SOG}, \Gamma_0 \rangle$*.*

1. Γ *is a finite set of diagnoser aggregates.*

2. Γ_0 *is the initial diagnoser aggregate with;*
   a) $\Gamma_0.Q_n = Saturate_{\Sigma_u \setminus \Sigma_f}(q_0)$*;*
   b) $\Gamma_0.Q_f = Saturate_{\Sigma_u}(Img(\Gamma_0.Q_n, f))$*.*

3. $\rightarrow_{SOG} \subseteq \Gamma \times \Sigma_o \times \Gamma$ *is the transition relation, defined as below,*
   $\forall a, a' \in \Gamma, \sigma \in \Sigma_o$ *s.t.* $\sigma \in Enable(a.Q_n \cup a.Q_f)$*:*
   $a \xrightarrow{\sigma}_{SOG} a' \Leftrightarrow a'.Q_n = Saturate_{\Sigma_u \setminus \Sigma_f}(Img(a.Q_n, \sigma))$
   $\wedge \quad a'.Q_f = Saturate_{\Sigma_u}(Img(a'.Q_n, f) \quad \cup$
   $Img(a.Qf, \sigma))$

To summarize, the hybrid diagnoser $D_{SOG}(G)$ is constructed as follows: let the current aggregate of the diagnoser be a, and the next observed event be σ. The target diagnoser aggregate a' of the hybrid diagnoser is computed following these rules:

1. If $\sigma \in Enable(a.Q_n) \cap Enable(a.Q_f)$ then:
   a. $a'.Q_n = Saturate_{\Sigma_u \setminus \Sigma_f}(Img(a.Q_n, \sigma))$.
   b. $a'.Q_f = Saturate_{\Sigma_u}(Img(a'.Q_n, f) \quad \cup$
   $Img(a.Qf, \sigma))$.

2. If $\sigma \in Enable(a.Q_n) \setminus Enable(a.Q_f)$ then:
   a. $a'.Q_n = Saturate_{\Sigma_u \setminus \Sigma_f}(Img(a.Q_n, \sigma))$.
   b. $a'.Q_f = Saturate_{\Sigma_u}(Img(a'.Q_n, f))$.

3. If $\sigma \in Enable(a.Q_f) \setminus Enable(a.Q_n)$ then:
   a. $a'.Q_n = \emptyset$.
   b. $a'.Q_f = Saturate_{\Sigma_u}(Img(a.Q_f, \sigma))$.

These rules preserve a specific fault propagation. From an $F\text{-}uncertain$ diagnoser aggregate, we can reach either another $F\text{-}uncertain$, an $F\text{-}normal$ or an $F\text{-}certain$ diagnoser aggregate, from an $N\text{-}certain$ diagnoser aggregate, we can reach either another $N\text{-}certain$ diagnoser aggregate or an $F\text{-}uncertain$ one, and finally from an $F\text{-}certian$ diagnoser aggregate, we can reach only another $F\text{-}certain$ diagnoser aggregate, which depicts exactly the hypothesis of permanent failures. Figure 4 illustrates these points.

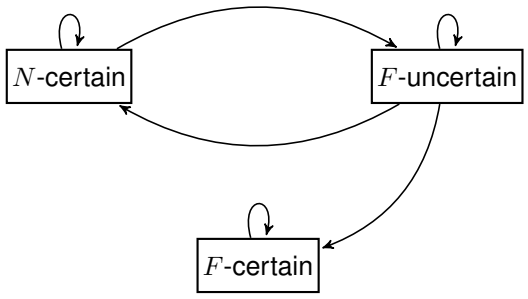

Figure 4: Fault propagation on the hybrid diagnoser

Example 1 *Figure 5 represents the hybrid diagnoser associated with the LTS of Example 1, depicted in Figure 1. As it is intended to construct the hybrid diagnoser on-the-fly, it is more convenient to represent the hybrid diagnoser as a tree-like structure.*

The initial aggregate composed of the initial state of the LTS and the state 2 reachable from state 1 by the occurrence of faulty event f*. Both of the diagnoser aggregates (2) and (3) contain two sets*

of states (each one is represented by a BDD). After the occurrence of event d we reach diagnoser aggregate (4), which is the same as diagnoser aggregate (2) thus, there exists a cycle on the hybrid diagnoser composed of aggregates (2) and (3) by executing the observable event sequence $(bd)^*$. Diagnoser aggregate (5) is reached after the occurrence of event t and it contains only the set of faulty states thus, it is an F-$certain$ diagnoser aggregate. As F-$certain$ diagnoser aggregates are not necessary to analyse diagnosability, in on-the-fly constructing of the hybrid diagnoser, we do not construct them. Indeed, one knows that all the subsequent aggregates will be F-$certain$ as well. Besides, computing such aggregates is not necessary for online diagnosis.

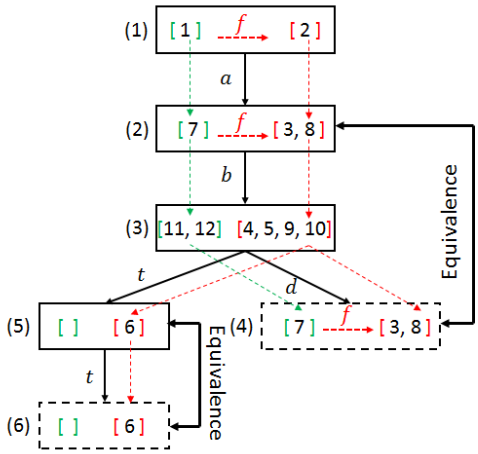

Figure 5: Hybrid diagnoser of the LTS in Figure 1.

We recall that our goal is to avoid the state-explosion problem, not only by providing this compact form (SOGs) to build the hybrid diagnoser but also by constructing the hybrid diagnoser on-the-fly and verifying diagnosability simultaneously. Constructing the hybrid diagnoser on-the-fly serves to avoid generating the whole state space of the diagnoser even if the system is diagnosable, i.e. as we deal with permanent faults, we do not need to construct the part, of the hybrid diagnoser, containing only F-certain diagnoser aggregates since such a part is not necessary for analysing diagnosability (see (Liu et al. 2014a) for more details).

Hereafter, we provide the SOG-based algorithm needed for on-the-fly construction of the hybrid diagnoser. The following function and data structures are used:

- $Img(S, t)$, as described previously, returns the set of immediate successors of the states of a set S through the occurrence of event t.
- OBDDs (Ordered Binary Decision Diagram) are used to represent the sets of states belonging to an aggregate, i.e. the set of normal states and the set of faulty states in an aggregate. This task is performed by the function Reduce().
- The hybrid diagnoser is represented by a standard graph representation with a set of vertices, namely V, and a set of edges, namely E, connecting these aggregates and labelled with observable events.
- $EnableObs(S)$ returns the set of observed events that are enabled by at least one of the states in set S.
- $Saturate_{\Sigma_i}()$, as defined before, computes the various states reached through events from set Σ_i.
- $Stack$ is an ordered set of 5-uplet, which contains two sets of states (S_n, S_f) and three sets of events (Evt_f, Evt_n, Evt) with $Evt = (Evt_n \cup Evt_f) \backslash (Evt_n \cap Evt_f)$.
- $IsUncertain()$ is a function that returns Boolean value ($true$ if the encountered cycle is composed of only f-$uncertain$ diagnoser aggregates and $false$ otherwise).
- $IsIndeterminate()$: is a function that returns Boolean value ($true$ if the existing cycle is an f-indeterminate cycle and $faulse$ otherwise. This function will be discussed later.
- $RemoveLast(S)$ is an operation that removes, then returns, the last event of a set S.
- For the sake of simplicity, we consider that Σ_f contains only one faulty event, generalization to a set of faulty events is straightforward.

The initialization step (lines 1-11) serves to compute the initial diagnoser aggregate, handle it efficiently using OBDD (function Reduce()), and push it, associated with its enabled observable events, into the stack. The construction of the hybrid diagnoser is performed using a depth first exploration: As long as the stack in not empty, a new observable event t, enabled by the diagnoser aggregate a at the top of the stack, is removed from the set of enabled events (Evt) and then processed. If such an event does not exist, the corresponding aggregate is poped from the stack (lines 13-15). Otherwise, if the set Evt_n of events enabled by the set of normal states S_n of the diagnoser aggregate a, is empty then the aggregate a is poped from the stack (lines 16 and 18). This step serves to avoid the construction of the subsequent F-$certain$ diagnoser aggregates, i.e. since this part

of the hybrid diagnoser is not necessary to analyse diagnosability.

The computation of the new aggregate a', reachable through an observable event from aggregate a, is completed by saturation on the unobservable events (lines 20-26). If a' has already been encountered (i.e. existence of a cycle) then the hybrid diagnoser is updated by adding a new edge (lines 27-28) and if the cycle is $uncertain$ (i.e., contains only f-$uncertain$ diagnoser aggregates), the function $IsIndeterminate()$ is launched in order to detect whether there exists an f-indeterminate cycle or not (line 29-32). If the cycle is an f-indeterminate one then we output that the model is undiagnosable and we stop the diagnoser construction. Otherwise, construction is continued, a' with its enabled observable events are pushed into the stack, and so on. When the stack is empty, then the necessary part of the diagnoser, for analysing diagnosability, is completely built, we output that the model is diagnosable.

As mentioned in Section 3, diagnosability analysis is performed by searching two infinite executions that share the same observed event-sequences such that one sequence contains a faulty event and not the other one. That means to search an f-indeterminate cycle in the diagnoser (Sampath et al. 1995). The same procedure is used in our case, i.e. searching f-indeterminate cycles in the hybrid diagnoser. Two steps are needed to check the existence of f-indeterminate cycles when a cycle of F-$uncertain$ diagnoser aggregate is found in the hybrid diagnoser:

1. Extract the observed event-sequence that leads to this cycle (of F-$uncertain$ diagnoser aggregates).

2. Check if this observed event-sequence corresponds to two cycle in the LTS model. One cycle is reached by a fault-free event-sequence and the other one is reached by a faulty event-sequence.

This task is performed by function $IsIndeterminate()$ in the Algorithm 1 (line 28) which calls a specific function ($path\_exists()$) from the $digraph$ library (Rushton 2012). ($digraph$ is a library dedicated for searching cycles from the system model).

We emphasize that verification of the existence of f-indeterminate cycles is performed on-the-fly in parallel to the process of constructing the hybrid diagnoser, i.e. the process of constructing the hybrid diagnoser runs and when a cycle of F-$uncertain$ diagnoser aggregates is found, we check whether

Algorithm 1 On-the-fly algorithm to construct the hybrid diagnoser

DiagSOG $(LTS, \Sigma_o, \Sigma_u, f)$;
Diagnoser aggregate a, a';
Set of vertices V;
Set of edges E;
Set of Events Evt_n, Evt_f, Evt;
Set of states S_n, S_f, S'_n, S'_f;
Stack $st = \emptyset$;

1: S_n = Saturate$_{\Sigma_u \setminus f}(q_0)$;
2: S_f= Img(S_n, f);
3: **if** $(S_f \neq \emptyset)$ **then**
4: S_f = Saturate$_{\Sigma_u}(S_f)$;
5: **end if**
6: $a = \langle S_n, S_f \rangle$;
7: Reduce (a, Σ_u);
8: $V = a$; $E = \emptyset$;
9: $Evt_n = EnableObs(S_n)$;
10: $Evt_f = EnableObs(S_f)$;
11: st.Push($\langle S_n, S_f, Evt_n, Evt_f, Evt \rangle$;
12: **while** $(st \neq \emptyset)$ **do**
13: $\langle S_n, S_f, Evt_n, Evt_f, Evt \rangle = st$.Top();
14: **if** $Evt = \emptyset$ **then**
15: $\langle S'_n, S'_f, Evt_n, Evt_f, Evt \rangle = st$.Pop();
16: **else**
17: **if** $(Evt_n = \emptyset$ **then**
18: $\langle S'_n, S'_f, Evt_n, Evt_f, Evt \rangle = st$.Pop();
19: **else**
20: $t = RemoveLast(Evt)$;
21: $S'_n = Img(S_n, t)$;
22: S'_n = Saturate$_{\Sigma_u \setminus f}(S'_n)$;
23: $S'_f = Img(S_f, t) \cup Img(S'_n, f)$;
24: S'_n = Saturate$_{\Sigma_u}(S'_f)$;
25: $a' = \langle S'_n, S'_f \rangle$;
26: Reduce (a', Σ_u);
27: **if** $(\exists w \in V \mid w = a')$; **then**
28: $E = E \cup a \xrightarrow{t}_{SOG} w$;
29: **if** $(IsUncertain())$ **then**
30: **if** $(IsIndeterminate())$ **then**
31: return UNDIAGNOSABLE;
32: **end if**
33: **end if**
34: **else**
35: $V = V \cup \{a'\}$;
36: $E = E \cup a \xrightarrow{t}_{SOG} a'$;
37: $Evt_n = EnableObs(S'_n)$;
38: $Evt_f = EnableObs(S'_f)$;
39: st.Push$\langle S'_n, S'_f, Evt_n, Evt_f, Evt \rangle$;
40: **end if**
41: **end if**
42: **end if**
43: **end while**
44: return DIAGNOSABLE;

this cycle corresponds to an f-indeterminate cycle or not. If it is the case (i.e. the cycle is an f-indeterminate cycle), then the whole process is stopped and a negative verdict is emitted regarding diagnosability, else the building process is continued.

Example 2 *Let us take again LTS G of Figure 1 and its hybrid diagnoser (Figure 5). We have a cycle, in the hybrid diagnoser, composed of only F-uncertain diagnoser aggregates $(2) \rightleftharpoons (3)$. Once the algorithm of construction arrives at this cycle, we check if this cycle is an f-indeterminate one or not, before continuing the construction process. The cycle of f-uncertain diagnoser aggregates $(2) \rightleftharpoons (3)$ is reached by executing the observed event-sequence $a(bd)^*$.*

In LTS G, the observed event-sequence $a(bd)^*$ corresponds to two event-sequences:

1. The faulty sequence $fa(bud)^*$ that leads to a cycle composed of states 3, 4, and 5.
2. The fault-free sequence $a(bud)^*$ that leads to a cycle composed of states 7, 11, 12.

Thus, we can infer that the cycle, in the hybrid diagnoser, composed of diagnoser aggregates (2) and (3) is an f-indeterminate cycle. Thus, we stop constructing the hybrid diagnoser and we output that LTS G is non diagnosable with respect the fault f.

5.2. A heuristic strategy to improve the building algorithm

Our algorithm for constructing the hybrid diagnoser is based on a depth-first search (DFS) to investigate the state space (diagnoser aggregate in the developed tree-like structures) execution by execution. Generally, no rules are defined to select the execution to be investigated first, i.e. the order of execution exploring is arbitrary. However, as we deal with diagnosability analysis, in our case, the diagnoser aggregate structure provides some information that can be exploited to direct the search in such a way as to increase the chances of quickly obtaining a diagnosability verdict by exploring the most promising executions at first.

When we deal with diagnosability analysis, the interesting executions of the system are those which share the same observed event-sequence such that some of them contain a faulty event and the others are fault-free. This is reduced to track the observed event-sequences, in the hybrid diagnoser, leading to F-uncertain aggregates. Generally, there exists three types of enabled transitions from any aggregate, as depicted in Figure 6.

1. Transitions enabled only by states from the faulty set (Figure 6.(a)). As said before, this type of branches will not be explored.
2. Transitions enabled only by states from the normal set (Figure 6.(b)). In this case, we need to continue the construction since other faults may occur in the future.
3. Transitions enabled from both normal and faulty sets (Figure 6.(c)). In this case, the reached diagnoser aggregate will be certainly F-uncertain.

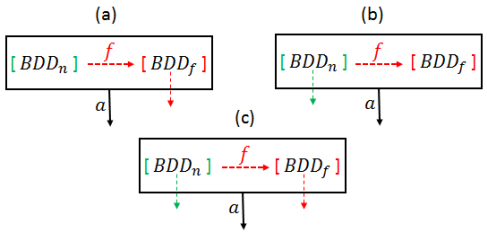

Figure 6: Types of enabled transitions from an aggregate

This last type of transitions is the most-promising to find an f-indeterminate cycle since we know, a priori, that the new diagnoser aggregate will be certainly an f-uncertain aggregate, contrary to the other above cases. Thus, it will be the first to be explored in order to direct the construction of the hybrid diagnoser and to potentially speed up the verification process. We note that in the actual version of the algorithm, this heuristic strategy is not implemented.

6. RELATED WORKS

In the literature, there are several diagnoser-based approaches for analysing diagnosability inspired from the seminal work of (Sampath et al. 1995). In (Zad et al. 2003) a state-based approach for on-line passive fault diagnosis was introduced. In the state-based approach, it is assumed that the set of states of the system model can be partitioned according to the faulty or normal condition of the system. In this work, a specific diagnoser is constructed as a finite state machine that takes information from the system (i.e. sequences of inputs/outputs) and generates an estimate of the condition of the system (i.e., faulty or normal). Establishing of this diagnoser has exponential time complexity. However, a model reduction scheme with polynomial time complexity is proposed to reduce the computational complexity of the procedure.

(Schumann et al. 2004) propose a symbolic framework based on binary decision diagrams

for the diagnosis of DESs. A symbolic version of Sampath's diagnoser was proposed, while requiring considerably lower space and time than the enumerative approach of (Sampath et al. 1995). Recently, (Liu et al. 2014a) propose an on-the-fly algorithm for constructing and checking diagnosability of discrete-event systems modelled by LPNs using an enumerative approach. The goal is to avoid the construction of the whole state-space of the diagnoser especially when the system is not diagnosable. The approach was experimented over a Petri net benchmark and the obtained results were promising compared to those of Sampath's approach. A tool, called OF-PENDA (Liu et al. 2014b), was developed based on this approach to analyse diagnosability, K-diagnosability and K_{min}-diagnosability of systems, modelled by Labelled Petri Nets.

The approach proposed in this paper, takes advantage from these two last approaches (i.e., (Schumann et al. 2004) and (Liu et al. 2014a)) by combining the symbolic representation of diagnoser states and on-the-fly techniques for constructing the hybrid diagnoser and verifying diagnosability. We believe that this approach will improve efficiently analysis of diagnosability in terms of runtime and memory resources. We still need to apply the approach on several benchmarks in order to assess its efficiency. Indeed, determining analytical complexity while considering the worst case is not appropriate for such an on-the-fly approach.

7. CONCLUSION

In this work, we have developed an on-the-fly approach for diagnosability analysis, based on a hybrid diagnoser. The approach is based on the symbolic observation graph (SOG), which is a paradigm that combines symbolic and enumerative representations in order to build a deterministic observer from a partially observed model. The approach aims to improve the efficiently in terms of runtime and memory resources when analysing diagnosability.

Several future directions are considered. First, we wish to make experimentations over case-studies in order to assess the efficiency and the scalability of our approach and also to compare the obtained results with those provided by other existing approaches. Then, we will investigate some other practical versions of diagnosability, namely K-diagnosability and K_{min}-diagnosability. Finally, we intend to extend the proposed approach for analysing diagnosability based on the verifier approach by means of a non deterministic version of the symbolic observation graph.

ACKNOWLEDGEMENT

This research project has been supported by the European Commission under the 7^{th} Framework Programme through the CARONTE project - Contract N: FP7-SEC-2013-606967

REFERENCES

J. Zaytoon and S. Lafortune. Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, pages 308–320, 2013.

C. G. Cassandras and S. Lafortune. Introduction to discrete event systems. *Second Edition, Springer*, 2007.

M. Sampath, R. Sengupta, and S. Lafortune. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, pages 1555–1575, 40(9), 1995.

S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, pages 46(8), 1318–1321, 2001.

T. S. Yoo and S. Lafortune. Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Transactions on Automatic Control*, pages 47(9), 1491–1495, 2002.

A. Cimatti, C. Pecheur, and R. Cavada. Formal verification of diagnosability via symbolic model checking. *Int. Joint Conference on Artificial Intelligence*, 2003.

A. Boussif and M. Ghazel. Diagnosability analysis of input/output discrete event system using model checking. *The 5^{th} International Workshop on Dependable Control of Discrete Systems*, 2015.

J. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall PTR, 1981.

T. Ushio, I. Onishi, and K. Okuda. Fault detection based on Petri net models with faulty behaviors. *Systems, Man, and Cybernetics*, pages 113–118, 1998.

Y. Wen and C. Li. A polynomial algorithm for checking diagnosability of Petri nets. *IEEE International Conference on Systems, Man and Cybernetics*, 3:2542–2547, 2005.

M. P. Cabasino, A. Giua, S. Lafortune, and C. Seatzu. Diagnosability analysis of bounded Petri nets. *Proceedings of the 48^{th} IEEE Conference on Decision and Control (CDC) held jointly with 28th Chinese Control Conference*, pages 1254–1260, 2009.

F. Basile, P. Chiacchio, and G. De Tommasi. On k-diagnosability of Petri nets via integer linear programming. *Automatica*, 48(9):2047–2058, 2012.

M. P. Cabasino, A. Giua, and C. Seatzu. Diagnosability of discrete-event systems using labeled Petri nets. *IEEE Transactions on Automation Science and Engineering*, 11(1):144–153, 2014.

B. Liu, M. Ghazel, and A. Toguyéni. Toward an efficient approach for diagnosability analysis of des modeled by labeled petri nets. *Proceeding of the* 13^{th} *European Control Conference*, 2014a.

S. Haddad, J.-M. Ilié, K. Klai, and F. Wang. Design and Evaluation of a Symbolic and Abstraction-based Model Checker. 2^{nd} *International Symposium on Automated Technology for Verification and Analysis (ATVA'04)*, pages 198–210, 2004.

R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, 24(3):293–318, 1992.

K. Klai and L. Petrucci. Modular construction of the symbolic observation graph. *The* 8^{th} *International Conference on Application of concurrency to System Design*, pages 23–27, 2008.

K. Klai and D. Poitrenaud. MC-SOG: An LTL model checker based on symbolic observation graphs. *Proceedings of the* 29^{th} *International Conference on Application and Theory of Petri Nets*, pages 23–27, 2008.

A. Rushton. A directed graph conainer. *http:// www.andyrhshton.co.un/programming/stlplus-library-collection*, 2012.

S. H. Zad, R .H. Kwong, and W. M. Wonham. Fault diagnosis in discrete-event systems: Framework and model reduction. *IEEE Transactions on Automatic Control*, 48(7):1199–1212, 2003.

A. Schumann, Y. Pencole, and S. Thiebaux. Diagnosis of discrete event systems using ordered binary decision diagrams. 15^{th} *International Workshop on Principles of Diagnosis*, 2004.

B. Liu, M. Ghazel, and A. Toguyéni. OF-PENDA: A software tool for fault diagnosis of discrete event systems modeled by labeled petri nets. *International Worshop Petri Nets for Adaptive Discrete-Event Control Systems*, 2014b.