Diagnosis of Discrete Event Systems Using Satisfiability Algorithms: A theoretical and empirical study

Alban Grastien, Anbu Anbulagan

***Abstract*—We propose a novel algorithm for the diagnosis of systems modelled as discrete event systems. Instead of computing all paths of the model that are consistent with the observations, we use a two-level approach: at the first level *diagnostic questions* are generated in the form *does there exist a path from a given subset that is consistent with the observations?*, whilst at the second level a SAT solver is used to answer the questions. Our experiments show that this approach, implemented in SAT, can solve problems that we could not solve with other techniques.**

***Index Terms*—Diagnosis, Discrete Event Systems, Propositional Satisfiability**

I. Introduction

Because of imperfect design, bad conception, improper use or simply natural ageing or uncontrollability of the environment, any system is prone to malfunction. These malfunctions or *faults* may reduce the quality of service of the system and be harmful for the system (through the cascading effect) or to the users. Monitoring the observations generated by sensors on the system or by the system itself is necessary to determine the occurrence of faults (*detection*), to determine which faults took place (*identification*), and to determine where faults took place (*isolation*). This is the task of *diagnosis* [1].

We are interested here in model-based diagnosis (MBD) where the diagnostic task is performed using only a description of the system's behaviour (the *model*); furthermore, we assume the model is a *finite-state discrete event system* (DES, [2]), i.e., a model for dynamic systems where the state evolution is discrete and its domain is finite. The traditional approach to diagnosis of DES is to compute all model paths consistent with the observations, and to extract the diagnosis information (such as, which faulty events occurred) from these paths. This approach is very expensive for large systems as the size of the model is exponential in the number of components; most techniques developed in the last 15 years aim at reducing this complexity but the class of systems they can diagnose is still very limited. In general however, we are not interested in all possible system behaviours but in more specific information.

Diagnosis of DES can be seen as finding specific paths on the system model. Similar path-finding problems include *classical AI planning*, *model-checking* and *diagnosability*. A successful approach that has been developed for these three problems is the use of propositional satisfiability (SAT) algorithms [3]–[5]. Given a propositional formula defined on a set of Boolean variables, SAT is the problem of finding an assignment to the variables that makes the formula logically *true*, or determining that no such assignment exists. The path-finding problem is transformed into a propositional formula that is satisfiable if and only if a path exists; a SAT solver is then run to find a solution to this problem, if any; finally the path is trivially reconstructed from the satisfying assignment.

We propose a similar approach for the diagnosis of DES. Our approach does not search for all the paths consistent with the observations; instead we use a two-level approach that looks for specific paths:

1) At the first level the diagnostic problem is transformed into simple problems that we call *diagnostic questions*. Each question is a decision problem of the following form: *Does the* DES *allow for a path generating the observations and satisfying a certain property?*
2) Each diagnostic question is answered at the second level. If the answer is *yes*, a path is returned that supports the positive answer. We implemented this level with SAT.

To illustrate the feasability of this method, we solve a simplified diagnostic problem whose purpose is to find an explanation that minimizes the number of fault occurrences; the relevance of such a problem is discussed in the next section, and the more general case is discussed in the Extension section. This allows us to concentrate on the second level of this approach, i.e., the translation of the diagnostic question into a SAT problem and its resolution.

This two-level approach allows us to use efficient SAT algorithms to solve hard diagnostic problems. Using this approach we can generate the miniminal-cardinality diagnosis and detect the occurrence of specific events. We make the following contributions:

1) We define a formal framework for a two-level approach to diagnosis of DES based on diagnostic questions.
2) We show how diagnostic questions can be translated into SAT problems (illustrated for the class of questions required for our simplified diagnostic problems).
3) We provide experimental results showing that this approach is able to compute the preferred diagnosis for a class of hard diagnostic problems.
4) We discuss the resolution of more general diagnostic problems and SAT-related improvements.

Alban Grastien is with the Optimisation Research Group of NICTA, 7 London Circuit, Canberra ACT 2601, Australia, and the AI Group of the Australian National University (email: alban.grastien@nicta.com.au).

Anbu Anbulagan is with the Australian National University, Canberra ACT 2600, Australia (email: a.anbulagan@anu.edu.au).

Related Work

DES diagnosis has been very well-studied since the seminal work from Sampath et al. [6]. The main issue addressed in the last 15 years is the complexity of the task, which is in general exponential in the number of components. The general approach is to compute a representation of all the system behaviours compatible with the observations, from which the diagnostic information can be extracted. Sampath et al. proposed to compile the model in a diagnoser that returns diagnosis in linear time; however, the compiled structure is double exponential in the number of components [7]. Decentralized [8], distributed [9]–[11], and symbolic [12] methods have been used to contain the exponential blow-up, with real yet unsufficient impact.

Petri net modelling was proposed to address the complexity issue [13]–[15]. Petri nets are often more compact than automata and represent concurrency more elegantly. The class of systems they can represent is different from that of automata: unbounded Petri nets can have an infinite state space; on the other hand, work on Petri nets often assumes no unobservable cycles. However the unfolding of Petri nets is subject to branching explosion which makes their use difficult. Recently a new approach was proposed [16], [17] that reduces the generation of diagnosis explanations to a sequence of Mixed-Integer Programming problems; while this reduction is very similar to our reduction to SAT, the existing approaches generate all explanations which seems inapplicable in practice. Furthermore, how to diagnose fault patterns [18] in this framework is still open.

More similar to our article Pencolé et al. proposed to diagnose each fault separately [19]. Grastien and Torta [20] studied how different diagnostic questions can be reformulated in a series of questions.

Outline: The next section presents the diagnosis of DES. Our two-level approach is discussed in Section III. The use of SAT for solving diagnostic questions is discussed in Section IV. We present a translation from a diagnostic question to SAT in Section V. An experimental validation is given in Section VI. Possible extensions are discussed in Section VII.

II. DIAGNOSIS OF DISCRETE EVENT SYSTEMS

Different approaches can be used for diagnosis, but the most robust are *model-based techniques* [1]. Model-based diagnosis (MDB) uses a description of the system – the *model* – and compares the possible system evolutions predicted by the model to the actual observations on the system to determine which evolutions may have taken place.

A. Discrete Event Systems

In this article, we are interested in DES, a class of models for dynamic systems. The state of a DES is usually the composition of states from each of its *components*. Because the components behave independently (to some extent), the state space is usually exponential in the number of components in the system. It is therefore inefficient or even impossible to build a DES with explicit enumeration of all system states. Two methods can be used to alleviate this problem:

- The decentralised approach: The DES can be implicitly defined as the composition of *component (local) models*. All components being relatively simple, their models can be constructed easily and reused for different instances of the same component type. The synchronisation between the local models must also be defined.
- The symbolic approach: The state of the system can be defined as the assignment of *state variables* to finite (small) domains. As opposed to the decentralised approach, the transition rules from one state to the other potentially involve not only the local state but any state variable.

The symbolic approach is very appealing for the SAT implementation we provide in this article because it makes the translation natural. The decentralised model, however, is generally the easiest to construct. Therefore in this article we define a new model that combines both types of model and take advantage of both approaches; the expressiveness of this model is however similar to most DES formalisms.[1]

In our framework a DES is defined as a set of interconnected component models and each component is modelled in a symbolic fashion.

1) Component:

Definition 1 (Component Model): A component c_i is *modelled* by a tuple $\langle V_i, E_i, R_i, I_i\rangle$ where

- V_i is a set of *state variables*; each variable $v \in V_i$ is defined on a finite domain $d(v)$;
- E_i is a set of *events* partitionned in *exogeneous events* E_i^{exo}, *input events* E_i^{in}, *observable events* E_i^{obs}, and *output events* E_i^{out};
- R_i is a set of *rules* (see after);
- I_i is the *initial assignment* of V_i.

A rule $r \in R_i$ is a tuple $\langle pre_r, e_r^I, E_r^O, \mathit{eff}_r\rangle$ where

- pre_r is the *precondition* of the rule defined as a propositional formula over V_i,
- $e_r^I \in E_i^{exo} \cup E_i^{in}$ is the *triggering event* of rule r,
- $E_r^O \subseteq E_i^{obs} \cup E_i^{out}$ is the (possibly empty) set of *events generated* by the rule trigger, and
- eff_r is the (possibly null) *effects* of the rule defined as a partial assignment of V_i.

A state q^i (or simply q) of component c_i is defined as the total assignment of each state variable $v \in V_i$ on its domain $d(v)$: $q \in \prod_{v\in V_i} d(v)$. The value of variable v in state q is denoted $v(q)$. Exogeneous events correspond to spontaneous events; input events to events triggered by other components; output events to events generated as a reaction of a rule trigger (some of which might trigger an input event on another component); observable events are similar to output events except that an external observer will notice the occurrence of such events. In this framework, a triggering event cannot be observable, although it is possible to define two events to circumvent this limitation. For instance, later in Example 1,

[1] As opposed to finite-state DES, Petri nets (PN) for instance allow for infinite state spaces, but most works on PN are restricted to the so-called safe sub-class of PN where the expressiveness is the same [13]; similarly, process algebra (PA) are potentially more powerful but, again, all existing works restrict themselves to a class of PA with finite state space [21].

a component may reboot and this reboot is observable; this behaviour is captured by two separate events: $\texttt{asy}_i$ is an event that models the triggering aspect of the reboot while $\texttt{Reb}_i$ is an event that models the alarm emitted. The evolution of component c_i is modelled through the trigger of a sequence of rules, each of which has to satisfy the current state and may affect this state.

Definition 2 (Path on a Component): A path on component c_i modelled by $\langle V_i, E_i, R_i, I_i\rangle$ is a sequence $q_0 \xrightarrow{e_1^I|E_1^O} q_1 \xrightarrow{e_2^I|E_2^O} \dots \xrightarrow{e_n^I|E_n^O} q_n$ s.t. there exists a sequence of rules $r_1, \dots, r_n$ where

- $\forall j \in \{1, \dots, n\}$, we define r_j as $\langle pre_{r_j}, e^I_{r_j}, E^O_{r_j}, \mathit{eff}_{r_j}\rangle$,
- $\forall j \in \{1, \dots, n\}$, $q_{j-1} \models pre_{r_j}$,
- $\forall j \in \{1, \dots, n\}$, $e_j^I = e^I_{r_j}$ and $E_j^O = E^O_{r_j}$, and
- $\forall j \in \{1, \dots, n\}$, $\forall v \in V_i$, $\begin{cases} v(q_j) = v(\mathit{eff}_{r_j}) & \text{if } v(\mathit{eff}_{r_j}) \text{ is defined,} \\ v(q_j) = v(q_{j-1}) & \text{otherwise.} \end{cases}$

The component model hence implicitly represents an automaton where each node is a state of the component model and each transition links state q to state q' if there exists a path $q \xrightarrow{e_r^I|E_r^O} q'$ of length one between the two states. When only the paths from the initial state are requested, the initial state of the automaton is the initial state I_i.

Example 1: There is no benchmark in diagnosis of DES and this article formally presents a model we used in previous work.[2] Figures 1a and 1b show one instance of the components in the model. This component models a fictional server in a network. The nominal state is O. When a fault takes place (event $\texttt{f}_i$, transition from O to F), the component c_i emits the message $\texttt{reb}_{i\to j}$ to each of its neighbours c_j (there is therefore a single transition with as many output events as this component has neighbours), asking them to reboot. The event $\texttt{asy}_i$ models the asynchronous effect of an event. The two observable events are represented with upper-case letters: $\texttt{Reb}_i$ corresponds to the alarm saying the component c_i is rebooting; $\texttt{Back}_i$ corresponds to the alarm saying the component c_i has finished rebooting. Finally the event $\texttt{reb}_{i\leftarrow j}$ corresponds to the reception of a rebooting message for a neighbouring component c_j; there are hence as many transitions between O and W as there are neighbours to c_i.

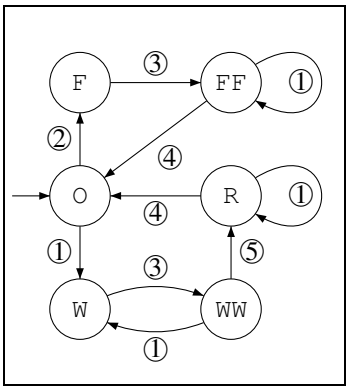
Each component has exactly four neighbours. The component state is modelled by four Boolean variables: f indicates whether a fault just took place on the component; w indicates whether a neighbour forced this component to reboot; a models the asynchronous behaviour before and after the component reboots; r indicates the component finished rebooting. All states assignments are given in Figure 1c. To illustrate, the transitions leading to state O are modelled by the following rule: $\langle r = \top, \texttt{asy}_i, \{\texttt{Back}_i\}, \{r \to \bot, f \to \bot\}\rangle$.

2) System: A system is a set of interconnected components. All components evolve separately but the connections restrict their evolutions and force them to synchronise.

Definition 3 (System Model): A *system model* is a tuple $SD = \langle \mathit{COMPS}, M, S\rangle$ where

- *COMPS* is a set of m components $\{c_1, \dots, c_m\}$,

[2]The model is available at www.grastien.net/ban/data/tac11/.



a) Automaton representing how the component state may change

①	$\texttt{reb}_{i\leftarrow j} \mid \emptyset$
②	$\texttt{f}_i \mid \{\texttt{reb}_{i\to j}\}_j$
③	$\texttt{asy}_i \mid \{\texttt{Reb}_i\}$
④	$\texttt{asy}_i \mid \{\texttt{Back}_i\}$
⑤	$\texttt{asy}_i \mid \emptyset$

b) Transitions label for Automaton a)

State	f	a	r	w
O	$\bot$	$\bot$	$\bot$	$\bot$
F	$\top$	$\top$	$\bot$	$\bot$
FF	$\top$	$\bot$	$\top$	$\bot$
R	$\bot$	$\bot$	$\top$	$\bot$
W	$\bot$	$\top$	$\bot$	$\top$
WW	$\bot$	$\bot$	$\bot$	$\top$

c) Symbolic encoding of the component states

Fig. 1. The model of a single component.

- M is a mapping that associates each component $c_i \in \mathit{COMPS}$ with its model $\langle V_i, E_i, R_i, I_i\rangle$ s.t. $i \neq j \Rightarrow E_i \cap E_j = \emptyset$,
- $S \subseteq \bigcup_{c_i \in \mathit{COMPS}} E_i^{out} \times \bigcup_{c_j \in \mathit{COMPS}} E_j^{in}$ is a set of pairs $\langle e_i^O, e_j^I\rangle$ where e_i^O is an output event of component c_i and e_j^I is an input event of component $c_j \neq c_i$ s.t. $\{\langle e_1, e_2\rangle, \langle e_3, e_4\rangle\} \subseteq S \Rightarrow \big(e_1 = e_3 \Leftrightarrow e_2 = e_4\big)$.

We define $E^{exo} = \bigcup_i E_i^{exo}$, $E^{in} = \bigcup_i E_i^{in}$, $E^{obs} = \bigcup_i E_i^{obs}$, and $E^{out} = \bigcup_i E_i^{out}$. A synchronisation $\langle e_i^O, e_j^I\rangle \in S$ means that the output event e_i^O must take place at the same time as the input event e_j^I. These two events model the same physical reality, e_i^O from the point of view of component c_i, e_j^I from the point of view of component c_j.

The system state is defined as a tuple of states for each component: $q = \langle q^1, \dots, q^m\rangle$ where $\forall i \in \{1, \dots, m\},\ q^i \in Q_i$. The system behaviour is modelled by a path that corresponds to the synchronised execution of m paths on each component. The component path definition needs to be extended to allow empty transitions where nothing happens on the component.

Definition 4 (Extended Component Path): The *extended set of rules* of component c_i modelled by $\langle V_i, E_i, R_i, I_i\rangle$ is defined by $R_i^* = R_i \cup \{\langle \mathit{true}, \varepsilon_i, \emptyset, \{\}\rangle\}$ where $\varepsilon_i \notin E_i$.

An *extended path* ρ of component c_i modelled by $\langle V_i, E_i, R_i, I_i\rangle$ is a path on the fictional component c_i^* modelled by $\langle V_i, E_i \cup \{\varepsilon_i\}, R_i^*, I_i\rangle$.

Definition 5 (Synchronised Transitions): Let $SD = \langle \mathit{COMPS}, M, S\rangle$ be a system and let $t = \{t_1, \dots, t_m\}$ be a set of component transitions s.t. $\forall i \in \{1, \dots, m\},\ t_i = q_i \xrightarrow{e_i^I|E_i^O} q_i'$ is an extended path of component c_i. The set of transitions t is *synchronised* if there exists $i \in \{1, \dots, m\}$ s.t.:

1) $e_i^I \in E_i^{exo}$,

2) $\forall j \neq i,\ e_j^I \neq \varepsilon_j \Rightarrow e_j^I \in E_j^{in} \wedge \exists k \in \{1,\ldots,m\} : \exists e \in E_k^O : \langle e, e_j^I \rangle \in S$, and
3) $\forall j \in \{1,\ldots,m\},\ \forall e \in E_j^O,\ \forall e',\ \langle e, e' \rangle \in S,\ \forall k,\ e' \in E_k^{in} \Rightarrow e' = e_k^I$.

The *synchronisation* of t is the path $q \xrightarrow{E^I|E^O} q'$ defined by:

- $q = \langle q_1, \ldots, q_m \rangle$,
- $q' = \langle q'_1, \ldots, q'_m \rangle$,
- $E^I = \bigcup_{e_i^I \neq \varepsilon_i} \{e_i^I\}$, and
- $E^O = \bigcup_{i \in \{1,\ldots,m\}} E_i^O$.

In Definition 5 the first property ensures an exogeneous event triggers the transition (on component c_i); the second property ensures that if a transition takes place on another component c_j, then it is triggered by the occurrence of an input event synchronised with an output event of another component c_k; finally the third property ensures any output event that is part of a synchronisation is indeed synchronised. In the following, the subset of non-empty transitions of t (i.e., those transitions t_i s.t. $e_j^I \neq \varepsilon_j$) is called a *macro-transition*.

The definition of a transition label and, consequently, the definition of synchronisation are generally simpler in the literature. A transition is usually associated with a single event and the synchronisation simply makes sure that events that are shared by different components occur at the same time on each of these components. The expressiveness of our model is very similar to the traditional models. One advantage is that the model of a single component is less dependant from the other components in our approach. For instance, in an electrical circuit, a switch being closed may lead to a circuit-breaker to trip, depending on the current state of the circuit; this can be modeled by complex events such as e.g. `switch_closed_tripping_breaker` but an approach based on sets of events is more appealing.

Observe also that a transition label can be used to symbolically represent the set of transitions where this label appears; this is particularly useful if the label has real semantics and does not correspond to an arbitrary set of transitions. By associating several labels with each transition, we increase the number of sets of transitions that can be represented easily, which is very useful for SAT solvers as they naturally manipulate transitions symbolically.

Definition 6 (System Path): A *path* on the DES $SD = \langle COMPS, M, S \rangle$ is a sequence $q_0 \xrightarrow{E_1^I|E_1^O} \ldots \xrightarrow{E_n^I|E_n^O} q_n$ s.t. $\forall i \in \{1,\ldots,m\}$, $q_0^i \xrightarrow{e_1^{iI}|E_1^{iO}} \ldots \xrightarrow{e_n^{iI}|E_n^{iO}} q_n^i$ is an extended path of component c_i and each path $q_{j-1} \xrightarrow{E_j^I|E_j^O} q_j$ is the synchronisation of the set of paths $\{q_{j-1}^i \xrightarrow{e_j^{iI}|E_j^{iO}} q_j^i\}$.

Example 2: The system in the benchmark we introduce here contains 20 components. The synchronisations are defined as follows: $\langle \mathtt{reb}_{i\to j}, \mathtt{reb}_{j\leftarrow i} \rangle \in S$ for all pair $\langle c_i, c_j \rangle$ of neighbouring components.

The system topology is represented in Figure 2 where components are represented by rectangles and connections by lines between the rectangles. To fully illustrate the connectivity one component (in coordinate $\langle 0, 1 \rangle$) and its four neighbours are filled in grey. Formally the topology forms a torus: the position of each component is defined by two integers $x \in \{0,\ldots,3\}$ and $y \in \{0,\ldots,4\}$; two components $\langle x_1, y_1 \rangle$ and $\langle x_2, y_2 \rangle$ are neighbours if their values x and y vary in total by one unit:

$$\begin{array}{c} (x_1 = x_2) \wedge (y_1 \pm 1 = y_2 \mod 5) \\ \vee \\ (y_1 = y_2) \wedge (x_1 \pm 1 = x_2 \mod 4) \end{array}$$

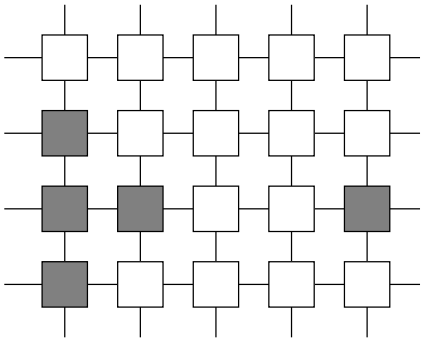


Fig. 2. The system topology. Each box corresponds to a component whose model is given in Figure 1. The grey boxes represent one component and its four neighbours.

B. The Diagnostic Problem

The system dynamically evolves from external and internal stimuli. We assume that each possible evolution of the system is represented by a path on the DES. When an evolution takes place, the system generates a sequence of observations which is exactly the sequence of observable events that takes place in the corresponding path.

Definition 7 (Observations): The *observations* of path $\rho = q_0 \xrightarrow{E_1^I|E_1^O} \ldots \xrightarrow{E_n^I|E_n^O} q_n$ are defined as the sequence $obs(\rho) = (E_1^O \cap E^{obs}, \ldots, E_n^O \cap E^{obs})$.

Notice that we consider every transition of the path is observable; a transition that corresponds to no observable event is still observed and generates an empty observation. This assumption is relaxed in Section V. Some frameworks consider state-based observations; such a framework can be easily recast in our framework.

Observations on the system allow us to monitor its behaviour, although it may not be sufficient to determine precisely the system behaviour. The system takes an unknown evolution modelled by an unknown path ρ on DES SD; what we do know about path ρ is the sequence of observations $obs(\rho)$ which is exactly the sequence generated by the system. The *diagnoser*'s task is to reason on the model to determine what happened on the system. In general, there may be many explanations for the system observations; a diagnosis is therefore generally a collection of *diagnosis candidates*, each diagnosis candidate being associated with one or more explanations (or even zero if the diagnoser is imprecise). The expected output of diagnosis is often which faults occurred, i.e., particular unexpected events. In the seminal work from Sampath et al. [6], a diagnosis candidate is the set of faults that occurred on the system, i.e., a diagnosis is a collection of sets of faults. For intermittent faults [22], the number of occurrences becomes relevant [23]. Of interest is also the order in which the failure took place, as certain sequences can be more

harmful than others; we started investigating such problems, including using methods developped in this paper [24], [25]. Several approaches prefer to compute a structure (for instance an automaton) that represents all system behaviours consistent with the observations, from which structure it is assumed any relevant information can be easily retrieved [8], [9].

For this paper however, we consider a simplified diagnostic problem, which allows us to concentrate on the SAT part of the diagnostic algorithm. Notice however that all these definitions of diagnosis can be handled by this approach. We propose to do a diagnosis-based alarm clustering. In a large system where many incidents can take place roughly at the same time, alarm clustering is the problem of grouping the observations (alarms) that pertain to the same incident to simplify the presentation of these alarms. To this end, we proposed [26] to compute a single explanation of the observations from which the relations between the observations can be retrieved. Using several explanations would not help here as the inferences from each explanation would be contradictory. In order to provide the best clustering possible, the single explanation returned by the diagnosis should be the most probable. Therefore, we compute the explanation that minimizes the number of (rare) "unexpected events" (to make it easier to relate to existing works, these events are later referred to as "faults" although they may not correspond to faults). Concentrating on the preferred explanation is by no means new, and has been used e.g. for Livingstone [27], a model-based diagnoser developed by NASA for self-reconfiguring autonomous spacecrafts, although Livingstone considered a probabilitic model to decide which explanation is most likely.

Definition 8 (Diagnostic Problem and Diagnosis): A *diagnostic problem* is a pair $\langle SD, obs, F\rangle$ where SD is a system model, obs represents the observations of some path on the model, and $F \subseteq E$ is a subset of system events.

The *diagnosis* of diagnostic problem $\langle SD, obs, F\rangle$ is a path on SD that generates obs and that minimizes the number of occurrences of events from set F.

III. Proposed Approach

This approach generalizes the work presented in [28]; it can be used to solve more general problems than the simplified diagnostic problem considered in this paper [24], [25], although this line of research is still on-going work.

Diagnosis aims at explaining what is going on in the system. As we have shown previously existing algorithms usually compute all possible evolutions that can lead to the observations, and then extract the relevant information. We propose an opposite approach where we check the existence of evolutions that support certain diagnostic results. To this end, we introduce the concept of *diagnostic question* and recast the diagnostic problem in a sequence of such problems.

Definition 9 (Diagnostic Question): A *diagnostic question* $\mathcal{Q}$ is a tuple $\langle SD, obs, \mathcal{C}\rangle$ where $SD = \langle \mathit{COMPS}, M, S\rangle$ is a system model, obs is a sequence of observations, and $\mathcal{C}$ is a set of paths.

The answer to the diagnostic question $\langle SD, obs, \mathcal{C}\rangle$ is

yes if there exists a path ρ on SD s.t. $obs(\rho) = obs$ and $\rho \in \mathcal{C}$, together with path ρ,

no otherwise.

For a given diagnostic problem, a diagnostic question is defined solely by the set of paths $\mathcal{C}$.

We solve the diagnostic problem by translating it into a sequence of diagnostic questions. Hence the two levels of the algorithm:

1) In the first level, the algorithm chooses the diagnostic questions. This choice can be made statically or based on the result of the previous diagnostic questions.
2) In the second level, the algorithm solves the diagnostic questions. In this article, the solver is based on a SAT procedure.

In our particular definition of diagnosis, the first level is implemented as follows. For the first diagnostic question, the set $\mathcal{C}_1$ is defined as the set of paths that contains no fault (nominal behaviour). If the answer to this question is positive, then we just found a path that minimizes the number of faults; otherwise, a new question is generated with the set $\mathcal{C}_2$ of paths that allow one fault occurrence. New questions with sets allowing an increasing number of faults are generated until a positive answer is obtained, at which stage a preferred diagnosis is found. More general diagnostic problems can be solved using a more sophisticated first level; however, this paper concentrates on the second level.

IV. SAT and Diagnostic Questions

In this section we present SAT and discuss how diagnostic questions can be solved using SAT. The precise translation to SAT is given in the next section.

A. The SAT Problem: Definition and Algorithms

Let $\mathcal{V}$ be a set of propositional variables and let Φ be a propositional formula over $\mathcal{V}$. The *Boolean satisfiability problem* (SAT) is the problem of determining whether there exists an assignment $\alpha : \mathcal{V} \to \{\top, \bot\}$ that makes the formula Φ logically true. SAT solvers usually return such an assignment. The formula is often expressed in conjunctive normal form (CNF), but this is not a strong restriction as any formula can be transformed to an equivalent CNF formula by introducing only a linear number of variables. A CNF is a conjunction of *clauses*, each clause being a disjunction of *literals*, each literal being a variable or its negation. An example of a clause is

$$a \vee b \vee \neg c \tag{1}$$

which specifies that either a should be set to $\top$, b to $\top$, or c to $\bot$ to satisfy the clause. Each clause can be seen as a constraint on the assignment α.

SAT was the first problem shown to be NP-Complete [29], which makes it a hard problem in general. However the last two decades have seen massive energy deployed by the SAT community to develop fast and efficient algorithms based on systematic methods and on stochastic local search (SLS). SLS algorithms are unable in general to demonstrate a formula Φ is unsatisfiable; we therefore ignore them in the following.

Current systematic algorithms are based on the DPLL procedure [30]. The algorithm picks a *branching variable*,

assigns it arbitrarily to $\top$ or $\bot$, propagates the information, and recursively picks another variable until all variables are assigned. The information propagation serves two purposes. First it allows to infer certain assignments; for instance, if a and b are both assigned to $\bot$ and the clause (1) belongs to the SAT problem, then c has to be assigned to $\bot$; this is known as *unit propagation* (UP). Second, it allows to detect inconsistencies when all literals of a clause are negated; in this case, the SAT algorithm backtracks to the last relevant branching decision to assign the variable differently.

An improvement that is intensively used is the *conflict-directed clause learning* (CDCL) mechanism. When an inconsistency (a *conflict*) is detected, the graph of UP is studied to extract a set of assignments that, together with the UP mechanism, led to the conflict. This set can be rewritten into a clause that is added to the SAT problem. This clause allows to avoid similar conflicts in the future: thanks to UP, this clause will allow to derive the non-conflicting assignment. Of importance is the choice of the branching variable; existing procedures favour the variables involved in the last conflicts [31]. Another improvement is the *one-step lookahead* [32] used as a heuristic for the next branching variable.

The results published by the *SAT competition*[3] show that solvers based on clause learning (such as zChaff [33], MINISAT [34], RSat [35] or glucose [36]) are very good on industrial problems; lookahead algorithms (such as Dew_Satz [37], Kcnfs (an improvement of cnfs [38]) and March_dl [39]) perform better on random problems. All these solvers can be accessed from the SAT competition website.

B. How to Express Diagnostic Questions in SAT

A diagnostic question is the problem of determining whether there exists a path on the DES that satisfies specific properties: it should be consistent with the observations and it should belong to a set of paths $\mathcal{C}$. Similar path-finding problems have already been successfully reduced to SAT, e.g. in planning [3], model-checking [4], and diagnosability [5].

Consider a path $\rho = q_0 \xrightarrow{E_1^I|E_1^O} \dots \xrightarrow{E_n^I|E_n^O} q_n$ of fixed length n.[4] The transitions define timesteps: the initial state is in timestep 0, the first transition takes place at timestep 1, etc. We now propose a set $\mathcal{V}$ of variables such that the path ρ can be modelled by an assignment $\alpha(\rho) : \mathcal{V} \to \{\top, \bot\}$, i.e., such that all paths are associated with a different assignment. Constraints will be defined to ensure that the paths which are solutions to the diagnostic question are exactly those associated with an assignment that satisfies these constraints. Solving the SAT problem will therefore exhibit a solution to the diagnostic question. The set $\mathcal{V}$ must represent all possible paths and is therefore defined as follows:

- For all component $c_i \in COMPS$, for all state variable $v \in V_i$ of component c_i, for all value ν in the domain $d(v)$ of variable v, for all timestep $j \in \{0, \dots, n\}$, $(v = \nu)@j \in \mathcal{V}$. Assignment $\alpha(\rho)$ assigns $(v = \nu)@j$ to $\top$ iff $v(q_j) = \nu$. These variables are used to record the system state at each timestep of the path.
- For all component $c_i \in$ *COMPS*, for all event $e \in E_i$ of component c_i, for all timestep $j \in \{1, \dots, n\}$, $e@j \in \mathcal{V}$. Assignment $\alpha(\rho)$ assigns $e@j$ to $\top$ iff $e \in E_j^I \cup E_j^O$. These variables are used to record which events took place at each timestep.

Additional variables, which are not strictly necessary, may also be introduced. Such auxilliary variables can make the SAT problem more compact and easier to solve; this will be illustrated in Section V.

By definition each path ρ is associated with exactly one assignment $\alpha(\rho)$. Given an assignment $\alpha(\rho)$, it is trivial to retrieve the corresponding path ρ.

Answering a diagnostic question $\mathcal{Q} = \langle SD, obs, \mathcal{C} \rangle$ consists in finding a path $\rho \in \mathcal{C}$ on SD generating observations obs. The corresponding SAT problem $\Phi(\mathcal{Q})$ is a collection of constraints that ensures that $\alpha(\rho)$ is a satisfying assignment to $\Phi(\mathcal{Q})$ iff ρ is a positive path of $\mathcal{Q}$. These constraints are partitioned in the following three sets:

- ρ should be a path on the model: constraints Mod.
- ρ should generate obs: constraints Obs.
- ρ should be a path of $\mathcal{C}$: constraints Que.

The SAT-based solving of a diagnostic question is implemented as follows: the SAT problem $\Phi(\mathcal{Q})$ is generated and passed to a SAT solver. If the problem is not satisfiable, then the answer to the diagnostic question is negative; otherwise, a satisfying assignment $\alpha(\rho)$ is produced from which a positive path ρ can be extracted. It should be clear that the SAT solver needs to be complete, i.e., it should be able to determine that a SAT problem is unsatisfiable.

C. Expected Performance of SAT-Based Diagnosis Algorithm

We now provide the intuitions that explain how SAT-based diagnosers can be expected to perform compared to more traditional approaches.

SAT has already been proposed to solve similar problems such as classical AI planning, model-checking, or diagnosability with some success. Let us turn to the latter problem which the readers may be more familiar with. A system is diagnosable if a fault can always be detected and identified by a diagnoser a bounded number of steps after it occurred. A system is not diagnosable if it exhibits a pair of paths that i) cycle and ii) produce the same observations, one of which being faulty and the other not [40]. Similarly to diagnosis, assuming an upper bound on the length of these paths, we look for a so-called critical path on the twin plant (the product of the system with itself) that satisfies certain properties, which can be recast as a SAT problem [5]. This approach allowed to prove non diagnosability of systems with $\sim 6^{200}$ states in seconds. (Diagnosability however cannot be proved with SAT as the length of the smallest critical path is quadratic in the number of states in the worst-case, which leads to an untractable SAT problem.)

By definition, each diagnostic question aims at finding a single path. The existing approaches explore all paths, and much of the effort is targetted at representing these paths efficiently

[3] Cf. `www.satcompetition.org/`.

[4] Choosing the appropriate value for n is a problem on its own; as this value depends on how many and what type of observations were received, this issue is discussed in the subsection V-B dedicated to translating the observations.

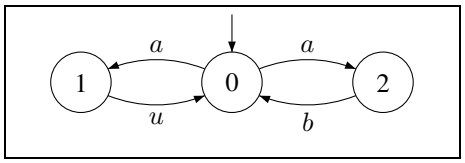


Fig. 3. Illustration of the unit propagation. a and b are observable, u is not. If a followed by b are observed, a SAT solver will trivially (i.e., without search effort) determine the system took transitions $0 \xrightarrow{a} 2 \xrightarrow{b} 0$.

and compactly (hence, the Sampath diagnoser states, the use of BDDs or Petri nets, etc.). Even if the diagnostic problem requires little search (i.e., not many paths are dead-ends), the effort of keeping all those paths is often prohibitive.

Considering more specifically SAT now, one very powerful aspect of the SAT algorithms is that they are very flexible in which variable they should instantiate first. On the contrary, existing algorithms generally develop a forward search starting from the initial state and trying to explain each observation incrementally. This aspect is clearly illustrated in the unit propagation, which accounts for most variable instantiations in SAT and which is illustrated on Figure 3. In this example, and starting from state 0, if observable event a is observed, a classical algorithm will have to consider both transitions $0 \xrightarrow{a} 1$ and $0 \xrightarrow{a} 2$. If the observation b is later made, the diagnoser will determine that the only first transition that allows the second observation is the one to state 2. On the other hand, a SAT-based diagnoser using a DPLL procedure, will be able to immediately infer that the second observation was generated from the transition $2 \xrightarrow{b} 0$ (because this is the only transition generating b); hence, the first transition $0 \xrightarrow{a} 2$ can be inferred without any search. This heuristic is very powerful, and it has been proved that SAT algorithms are stronger than the existing heuristics in classical AI planning [41].

SAT solvers have sophisticated procedures to find the best (non-unit propagation) variable to assign [31]. Furthermore, the problem can be more or less difficult depending which variable you start the search from; therefore, the SAT solver can stop the search if it considers it has been running for too long, and restart with a different set of variables [42].

These benefits are furthermore enhanced by the clause learning capability. If a partial assignment of the SAT variables, i.e., a partially defined path on the model, is consistent with the observations, then the variables involved in the inconsistency are identified and a clause is generated that will prevent a similar incorrect assignment in the future. Assume a diagnostic problem where the first observations can be explained by a very large number of paths involving a sensor being faulty; assume that the next observations prove that the sensor is not faulty. In the classical approach, all these paths will be explored and then discarded. SAT, on the other hand, will explore one such path, determine that the sensor is not faulty, and will ignore any other path that involves this fault.

Finally, the diagnosis question allows to focus on a small part of the state space. For instance, if we look for a path with $k = 4$ faults, then as soon as the path being built involves this number of faults, we can disregard any other faulty transition. We have shown [43] that a careful implementation of the set $\mathcal{C}$ of paths considered in the question can lead to powerful inferences that can dramatically affect the runtime.

We would like to stress a weakness of the SAT-based approach. In the traditional approach, all paths of the DES consistent with the observations are computed, from where the diagnosis is extracted. In our approach however, each diagnostic question is solved independently, the resolution of each question may lead to discovering over and over again the same (sub)results. We think two possible approaches can be used to alleviate this issue: i) as diagnostic questions on the same diagnostic problem share many clauses, it is possible to use incremental SAT solvers [44] which are specifically designed to reuse learnt information; ii) reasonably expensive pre-processing can be performed on the diagnostic problem before calling the SAT solver in order to include additional knowledge of the problem at hand (e.g., the assignment of state variable v at timestep j could be determined by local reasoning); the cost of pre-processing might pay off if used to accelerate the resolution of many SAT problems.

V. Translating Diagnostic Questions Into SAT

The previous section gave the intuition how SAT can be used to solve diagnostic questions. In this section we present precisely the translation of each set Mod, Obs, Que.

A. Translating the Model

This subsection introduces Mod, which is the set of constraints ensuring the returned path is indeed a path from the model SD. To simplify and reduce the CNF, we define the following set of variables:

- For all component $c_i \in \mathit{COMPS}$, for all rule $r \in R_i$, for all timestep $j \in \{1, \ldots, n\}$, $r@j \in \mathcal{V}$. Assignment $\alpha(\rho)$ assigns $r@j$ to $\top$ iff the rule r is triggered in the jth macro-transition of ρ.

With these additional variables, the translation is given in Figure 4. Each constraint has the following meaning:

(2): At each timestep, each state variable must be associated with at least one value.
(3): At each timestep, each state variable can be associated with at most one value (equivalent to a cardinality constraint [45]).
(4): A rule can be triggered only if its precondition is satisfied.
(5): If the rule is triggered, its effects apply.
(6): A state variable changes value only as an effect of a rule. (The set of rules assigning value ν to state variable v is represented by $R_{v,\nu}$.)
(7): A rule triggers only if its input event takes place.
(8): A rule triggering implies its output events taking place.
(9): An event occurring implies one of the rules it is associated with being triggered. (The set of rules associated with e is represented by R_e.)
(10): Synchronisation events have to take place together.
(11): At most one rule can trigger at a time on a component.
(12): The path ρ should start in the initial state.

$$\forall c_i \in \textit{COMPS},\ \forall v \in V_i,\ \forall j \in \{0,\dots,n\},\ \bigvee_{\nu \in d(v)} (v = \nu)@j \tag{2}$$

$$\forall c_i \in \textit{COMPS},\ \forall v \in V_i,\ \forall \{\nu, \nu'\} \subseteq d(v) \text{ s.t. } \nu \neq \nu',\ \forall j \in \{0,\dots,n\},\ \neg(v = \nu)@j \vee \neg(v = \nu')@j \tag{3}$$

$$\forall c_i \in \textit{COMPS},\ \forall r = \langle pre, e^I, E^O, \textit{eff}\rangle \in R_i,\ \forall j \in \{1,\dots,n\},\ r@j \rightarrow pre@(j-1) \tag{4}$$

$$\forall c_i \in \textit{COMPS},\ \forall r = \langle pre, e^I, E^O, \textit{eff}\rangle \in R_i,\ \forall v \in V_i \text{ s.t. } \textit{eff}(v) \text{ is defined},\ \forall j \in \{1,\dots,n\},\ r@j \rightarrow (v = \textit{eff}(v))@j \tag{5}$$

$$\forall c_i \in \textit{COMPS},\ \forall v \in V_i,\ \forall \nu \in d(v),\ \forall j \in \{1,\dots,n\},\ \neg(v = \nu)@(j-1) \rightarrow (v = \nu)@j \rightarrow \bigvee_{r \in R_{v,\nu}} r@j \tag{6}$$

$$\forall c_i \in \textit{COMPS},\ \forall r = \langle pre, e^I, E^O, \textit{eff}\rangle \in R_i,\ \forall j \in \{1,\dots,n\},\ r@j \rightarrow e^I@j \tag{7}$$

$$\forall c_i \in \textit{COMPS},\ \forall r = \langle pre, e^I, E^O, \textit{eff}\rangle \in R_i,\ \forall e \in E^O,\ \forall j \in \{1,\dots,n\},\ r@j \rightarrow e@j \tag{8}$$

$$\forall c_i \in \textit{COMPS},\ \forall e \in E_i,\ \forall j \in \{1,\dots,n\},\ e@j \rightarrow \bigvee_{r \in R_e} r@j \tag{9}$$

$$\forall \langle e_1, e_2\rangle \in S,\ \forall \{k, k'\} = \{1, 2\},\ \forall j \in \{1,\dots,n\},\ e_k@j \rightarrow e_{k'}@j \tag{10}$$

$$\forall c_i \in \textit{COMPS},\ \forall \{r_1, r_2\} \subseteq R_i \text{ s.t. } r_1 \neq r_2,\ \forall j \in \{1,\dots,n\},\ \neg r_1@j \vee \neg r_2@j \tag{11}$$

$$\forall c_i \in \textit{COMPS},\ \forall v \in V_i,\ (v = v(I_i))@0 \tag{12}$$

Fig. 4. Clauses enforcing ρ to be a path on the model.

Notice that the encoding presented above does not enforce the occurrence of a single exogeneous event per timestep. This is intentional and due to the number n of transitions in the path being unknown in general. The estimate of n depends on the type of observations on the system and is therefore discussed in Subsection V-B.

Several improvements can be introduced in the modelling:

1) Removing redundant variables: The goal is to reduce the number of variables in the SAT problem. This simplifies the work for the solver while maintaining correctness. When an input event e is associated with only one rule, or when two events always take place together (e.g., synchronisation events), their variables may be merged.
2) Factorizing: The purpose of this operation is to express a set of constraints as compactly as possible to reduce the SAT solver memory requirements [46]. A typical example is Constraint (11) which states that only one rule can take place at a time; for all pair of rules, these two rules cannot take place together. It is possible to factorize the constraints for the rules associated with different input events by forbidding the simultaneous occurrence of two different input events on any component; for each component, a set of rules is defined that forbids the simultaneous occurrence of different input events, and a set of rules is defined that forbids the simultaneous occurrence of different rules associated with the same input event. Consider j input events associated with k rules each; the first encoding requires $\frac{(j\times k)(j\times k-1)}{2} = O(j^2k^2)$ binary clauses while the second encoding requires only $\left(\frac{j(j-1)}{2} + j\frac{k(k-1)}{2}\right) = O(j^2 + jk^2)$ binary clauses.
3) Information: The purpose here is to include redundant information in the SAT problem in order to allow the solver to infer more information during the search. Consider the clauses Figure 5. The last clause can be inferred from the former ones; it is therefore unnecessary. Consider now that a is set to $\top$; the last clause allows to infer immediately that c should be set to $\top$ as well. This additional clause makes the problem simpler to solve. In our instance it is often the case that a single event e is associated with several rules r_i, all of which share an assignment $v := \nu$. The following clause can be added to the SAT problem: $e@j \rightarrow (v = \nu)@j$.
More techniques could be used to include addition information, such as *if event e takes place at timestep j, event e' cannot take place at timestep $j+1$*, etc. We did not implement these additional enhancements. There is a trade-off between adding useful information and increasing the size of the CNF and the SAT solver memory requirements.

$$\frac{\begin{array}{c} a \rightarrow b_1 \vee \dots \vee b_k \\ b_1 \rightarrow c \\ \dots \\ b_k \rightarrow c \end{array}}{a \rightarrow c}$$

Fig. 5. Hyper-resolution inference technique: the bottom clause can be deduced from the top clauses; contrary to the top clauses, the bottom clause allows to easy determine that c is true if a is true.

Notice that the improvements introduced by these techniques apply to all timesteps and are therefore worth investing some energy as they will be intensively used.

B. Translating the Observations

This subsection introduces Obs, which is the set of constraints ensuring the returned path generates the observations on the system. Depending on the system considered, various types of observations may be received which lead to different translations. We consider here three types of observations: timestamped observations, totally ordered observations and partially ordered observations. All these cases make the common assumption that no observation is lost.

1) Timestamped Observations: The simplest case is when the observations obs are timestamped, i.e., they are defined exactly as presented in Section II. The translation is given in Figure 6. Each constraint has the following meaning:

(13): If observable event e is observed at timestep j, this event is enforced at timestep j.

(14): If observable event e is not observed at timestep j, this event is disabled at timestep j.

$$\begin{gathered}\forall c_i \in \textit{COMPS},\ \forall e \in E_i^{obs},\\ \forall j \in \{0,\dots,n\},\ \text{if } e \in obs[j]\\ e@j\end{gathered} \tag{13}$$

$$\begin{gathered}\forall c_i \in \textit{COMPS},\ \forall e \in E_i^{obs},\\ \forall j \in \{0,\dots,n\},\ \text{if } e \notin obs[j]\\ \neg e@j\end{gathered} \tag{14}$$

Fig. 6. Constraints representing timestamped observations

One benefit of timestamped observations is that the number n of transitions is known exactly. This also implies that exactly one macro-transition takes place at each time step. This is enforced by the constraints Figure 7:

(15): No two exogeneous events take place at the same timestep. Note that using a cardinality constraint encoding as in [45] would probably be more efficient.

(16): At least one exogeneous event takes place at each timestep.

$$\begin{gathered}\forall c_{i_1}, c_{i_2} \in \textit{COMPS},\ \forall e_1 \in E_{i_1}^{exo},\ \forall e_2 \in E_{i_2}^{exo},\\ \forall j \in \{0,\dots,n\},\\ \neg e_1[t=j] \vee \neg e_2[t=j]\end{gathered} \tag{15}$$

$$\begin{gathered}\forall j \in \{0,\dots,n\},\\ \bigvee_{e \in E^{exo}} e[t=j]\end{gathered} \tag{16}$$

Fig. 7. Constraints restricting paths with one transition per timestep

2) Totally Ordered Observations: The most common hypothesis assumes a total order of observations. In this case, the observations emitted by the system is a sequence of non-empty sets of observations. Compared to the timestamped observations, the empty sets of observations are removed from obs. Each set of simultaneous observations is called a salve. For instance, if the observations are $[\{a,b\},\{a\}]$ (a was emitted at the same time as b, and a was emitted later again), then the first salve is $\{a,b\}$ and the second salve is $\{a\}$.

An immediate consequence is that the number n of transitions is unknown and the exact timestep when the observations were emitted is also unknown. For instance the timestamped observations $obs_1 = [\{o\}]$, $obs_2 = [\emptyset,\{o\}]$, and $obs_3 = [\{o\},\emptyset]$ are indistinguishable when translated as totally ordered: $obs' = [\{o\}]$. What value should be chosen for n to make sure that all paths on the DES are considered and that the SAT-based solving of a diagnostic question is sound? Before answering this question, we start with the following two remarks:

Remark one: We mentioned in subsection V-A that our translation Mod of the model in SAT does not enforce a transition to take place at each timestep (the corresponding constraint is the Constraint 15 defined for the timestamped observations). Thanks to this omission the standard translation of Subsection V-A not only represents paths of length n but also any path of smaller length. If any path generating o salves includes at most $n(o)$ transitions, then the value $n = n(o)$ can be safely used.

Some DESs contain silent cycles defined as paths $\rho = q_0 \xrightarrow{E_1^I|E_1^O} \dots \xrightarrow{E_k^I|E_k^O} q_k$ s.t.:

- $k \neq 0$,
- $\forall i \in \{1,\dots,k\},\ E_k^O \cap E^{obs} = \emptyset$, and
- $q_0 = q_k$.

In other words ρ is a non empty cycling path that generates no observations. The system may take an unbounded number of such cycles without generating observations, which means that $n(o)$ does not exist. Some approaches such as [47] forbid silent cycles. However, silent cycles are usually not a problem for diagnosis. Indeed silent cycles represent sequences of events that cannot be diagnosed since they generate no observation; furthermore shorter paths are usually more likely explanations than longer paths and they are considered as preferred explanation. Therefore, we may compute $n(o)$ as the maximal number of transitions that the system can take without taking silent cycle while generating o salves of observations.

Remark two: The translation does not enforce a maximum number of transitions per timestep (the corresponding constraint is Constraint (15) defined for the timestamped observations). Large systems exhibit many independent (concurrent) behaviours where some components interact for a moment without interfering with the rest of the network. Consider two events a and b associated with different components c_a and c_b. Consider two paths $\rho_1 = \langle q_0, q_0'\rangle \xrightarrow{\{a\}|\emptyset} \langle q_1, q_0'\rangle \xrightarrow{\{b\}|\emptyset} \langle q_1, q_1'\rangle$ and $\rho_2 = \langle q_0, q_0'\rangle \xrightarrow{\{b\}|\emptyset} \langle q_0, q_1'\rangle \xrightarrow{\{a\}|\emptyset} \langle q_1, q_1'\rangle$. These two paths are generally similar from the point of view of diagnosis. Furthermore the two transitions could be merged as follows: $\rho = \langle q_0, q_0'\rangle \xrightarrow{\{a,b\}|\emptyset} \langle q_1, q_1'\rangle$. This is formally not a path but the two paths ρ_1 and ρ_2 may be retrieved from ρ. In this example, the paths of length two will be considered by the SAT solver although $n = 1$. This technique allows to reduce considerably the value of n.

With these remarks in mind, several possible strategies can be used to determine n:

1) n can be computed globally, i.e., the maximum number of transitions for o salves is computed.
2) The maximum number m of unobservable transitions between two consecutive observations is computed and n is determined by $n = (m+1) \times o$ (as usual, we do not consider the events after the last observation).

Notice that in both cases the second remark above applies, i.e., independent macro-transitions may be reordered and may apply at the same timestep. The benefit of the first strategy over the second one is that it generates a smaller value for n. This is illustrated by Figure 8: we can easily show that o salves correspond to at most $n(o) = 2o+1$; however since up to two invisible transitions may occur between two salves, the second strategy will lead to $n = 3o$. Although the second strategy requires more timesteps, it allows to fix the timestep where the observations are generated. Therefore, the observations will be represented by variable assignments (unit clauses) from

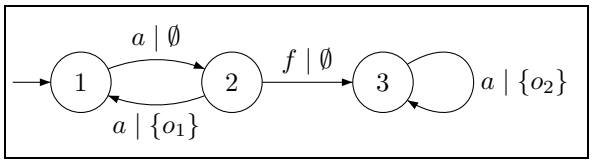


Fig. 8. A DES illustrating the difference between the two strategies for finding n. o_1 and o_2 being the only observable events, any two observable events are separated by at most two events, which means we can assume $3 \times o$ timesteps and the observations were generated every three timesteps. On the other hand, the upper-bound could be tighten to $(2 \times o) + 1$ but the timestep when each observation was emitted is uncertain.

which the unit propagation technique (the principal deduction technique used by SAT solvers) will be able to automatically deduce useful knowledge.

Having defined $n = (m+1) \times o$, it is possible to recast totally ordered observations in timestamped observations where

- the ith salve was produced at timestep $(m+1) \times i$,
- no observation was produced at timestep $j \mod (m+1) \neq 0$, and
- any number of macro-transitions may take place at the same time, i.e., the sets of constraints 14 and 15 do not apply.

3) Partially Ordered Observations: A difficulty that arises with large systems that are naturally decentralised is that the order in which the salves of observations were emitted is not obvious. A typical example is when the observations are alarms that warn of the existence of a fault somewhere in the system; because the fault and its side effects affect many components at the same time, all these components generate observations at roughly the same time. Many works in diagnosis of DES deal with partially ordered observations, e.g. [8], [9], [13], [48].

The observations obs are now defined as a set of salves equipped with a partial order relation $\preceq$. To model the observations in the SAT problem, we define a number of auxiliary variables that represent when each salve ξ took place.

- For all timestep $j \mod (m+1) = 0$ (as previously no observation is emitted at timestep $j \mod (m+1) \neq 0$), for all salve $\xi \in obs$, $\xi@j \in \mathcal{V}$. Assignment $\alpha(\rho)$ assigns $\xi@j$ to $\top$ iff the salve ξ is emitted at timestep j.
- For all timestep $j \mod (m+1) = 0$, for all salve $\xi \in obs$, $\overline{\xi@j} \in \mathcal{V}$. Assignment $\alpha(\rho)$ assigns $\overline{\xi@j}$ to $\top$ iff the salve ξ is emitted at timestep $j' \leq j$.

Many of those variables can actually be removed as their value is trivial. For instance consider two salves such that $\xi \prec \xi'$; then ξ' cannot be emitted at time $j = (m+1)$ because ξ has to be emitted before. The constraints on Figure 9 enforce the semantics of the new variables:

(17): A salve is emitted before or at timestep j iff it is emitted at timestep j or at the previous timestep $j' = j-(m+1)$.
(18): A salve ξ is emitted only once...
(19): ... but all salves are emitted.
(20): When a salve is emitted, the corresponding observable events take place.
(21): Observable events take place only when a corresponding salve is emitted.

$$\frac{\forall \xi \in obs,\ \forall k \in \{1,\dots,o\},\ \text{let } m' = m+1}{\overline{\xi@(km')} \leftrightarrow \overline{\xi@((k-1)\times m')} \vee \xi@(km')} \quad \text{where } \overline{\xi@0} \text{ is } \bot. \tag{17}$$

$$\frac{\forall \xi \in obs,\ \forall k \in \{1,\dots,o\},}{\overline{\xi@((k-1)\times(m+1))} \rightarrow \neg\xi@(k\times(m+1))} \quad \text{where } \overline{\xi@0} \text{ is } \bot. \tag{18}$$

$$\frac{\forall \xi \in obs,}{\overline{\xi@(o\times(m+1))}} \tag{19}$$

$$\frac{\forall \xi \in obs,\ \forall e \in \xi,\ \forall k \in \{1,\dots,o\},}{\xi@(k\times(m+1)) \rightarrow e@(k\times(m+1))} \tag{20}$$

$$\frac{\forall e \in E^{obs},\ \forall k \in \{1,\dots,o\},}{e@(k\times(m+1)) \rightarrow \bigvee_{\xi\in obs | e\in\xi} \xi@(k\times(m+1))} \tag{21}$$

Fig. 9. Constraints enforcing the values of ξ and $\overline{\xi}$.

Finally the partial order between the salves must be enforced. This is done by the constraints in Figure 10:

(22): The order between salves ξ and ξ' is enforced.
(23): Two salves $\xi \neq \xi'$ cannot be emitted at the same time.

$$\frac{\forall \xi,\xi' \in obs \text{ s.t. } \xi \prec \xi',\ \forall k \in \{1,\dots,o\},}{\xi'@(k\times(m+1)) \rightarrow \overline{\xi@((k-1)\times(m+1))}} \tag{22}$$

$$\frac{\forall \xi,\xi' \in obs \text{ s.t. } \xi \neq \xi',\ \forall k \in \{1,\dots,o\},}{\neg\xi@(k\times(m+1)) \vee \neg\xi'@(k\times(m+1))} \tag{23}$$

Fig. 10. Partial order on the salves

C. Translating the Type of Paths

This subsection introduces Que which is the set of constraints enforcing the path to be part of certain set $\mathcal{C}$. There is no generic translation to SAT as such a translation is not an explicit enumeration of all elements in set $\mathcal{C}$ but an implicit (symbolic) representation. We recently briefly presented some encodings [24].

As mentionned in Section III the sets $\mathcal{C}_i$ used for our diagnostic problem are defined as the set of paths with $i-1$ occurrences of faults. Let S be the set of SAT variables that register the occurrence of faults:

$$S = \{f@j \mid f \in F \ \wedge \ j \in \{1,\dots,n\}\}.$$

Then the set $\mathcal{C}_i$ can be encoded as the cardinality constraint that exactly $i-1$ variables in S are set to $\top$. We studied in [43] how carefully choosing the encoding can lead to dramatic performance improvement.

Traditionally, the diagnosis is the collection of sets of faults that label at least one path that explains the observations. Given a subset of faults $\delta \subseteq F$, the class $\mathcal{C}_\delta$ represents the set of paths that include an occurrence of $f \in F$ iff $f \in \delta$. The set δ belongs to the diagnosis iff the answer to diagnostic problem $\langle SD, obs, \mathcal{C}_\delta\rangle$ is positive.

For all $f \in F$, let $f_{\geq 1}$ be an auxiliary SAT variable that is true iff f occurs in the path:

$$f_{\geq 1} \longleftrightarrow (f@1 \vee \cdots \vee f@n)\,.$$

Let also v_δ be an auxiliary variable that is true iff δ is the set of faults that occur in the path:

$$v_\delta \longleftrightarrow \left(\bigwedge_{f \in \delta} f_{\geq 1} \wedge \bigwedge_{f \in F \setminus \delta} \neg f_{\geq 1} \right).$$

The set Que_δ corresponding to $\mathcal{C}_\delta$ can be encoded by the single unit clause v_δ.

The number of subsets of F is $2^{|F|}$, which can make the enumeration of all subsets of F impractical. If we assume that the number of diagnoses is small compared to $2^{|F|}$, a more clever approach can be adopted. Starting with the set $\mathcal{C}^0$ of all paths, we search for any path consistent with the observation. When a path is found, we extract the set δ_1 of faults that appear in this path and iteratively ask another question that forbid this set of faults: $\mathcal{C}^i = \mathcal{C}^{i-1} \setminus \mathcal{C}_{\delta_i}$. This is enforced by the set of constraints $Que^i = Que^{i-1} \cup \{\neg v_\delta\}$.

When the system is diagnosable [6], each fault can be diagnosed separately [20]. In this case, we can ask the diagnostic question with $\mathcal{C}_f$ (resp. $\mathcal{C}_{\neg f}$) that are defined as the sets of paths containing one or more (resp. zero) occurrence of f. The corresponding set of constraints can be defined by a unit clause over variable $f_{\geq 1}$.

VI. Validation

The following experiments were performed to evaluate the feasability of this paper's approach.

A. Experimental Set-Up

We use the benchmark presented in Section II; it is of interest to us because it shares many similarities with the type of system we want to diagnose [26]. The state of the system is modelled with only 80 Boolean variables (4 variables per component $\times 20$ components); the DES is therefore quite compact. However it is still very hard to use for diagnosis purposes. Indeed the model includes many concurrent behaviours together with synchronous behaviours. Furthermore the faults are not diagnosable in general [5], which means any model simplification or incomplete algorithm may introduce a loss of precision. All faults affect the output of the other faults and they share many symptoms.

We simulated a number of *scenarios* in the system, i.e., we generated paths on the DES that we then proposed to diagnose. We noticed that some parameters have a big influence on how hard the problem is to solve. In particular a question is whether the component should return quickly to its initial state or whether additional faults may take place while some of the components are rebooting. As it turns out, the latter case is easier to solve, possibly because it is easy to determine that a component is currently rebooting and to exonerate it from the following faults. We therefore defined ten levels of difficulty based on this property; level 1 for easy problems, level 10 for hard problems. We also introduced different number of faults, from one to twenty. Finally we considered three types of observations: timestamped observations, totally ordered observations, and partially ordered observations; in the latter case the order between salve i and salve $j > i$ is unknown if $j \leq i + 5$. Hence we generated $10 \times 20 \times 3 = 600$ scenarios.[5]

The diagnostic problem is defined as finding a path on the model consistent with the observations and that minimizes the number of faults (corresponding to the transition between state O and state F). We gave the solver 30 minutes (1 800 seconds). The translation to SAT is very fast compare to solving the problem and most of it can be done off-line; hence we only give the runtime for the SAT solver. We chose to use MINISAT 2.0 *core* version[6]. This solver was chosen as its source code is free and very simple; this allows for further experiments and diagnostic specific implementations in the future. Notice also that all best CSDL-DPLL SAT solvers have very similar performances. The experiments were performed on an Intel Xeon CPU E5405 2.00 GHz with 4 GB memory running Linux.

B. Results

The runtime are presented in Figure 11. The problems are sorted by category and ordered by increasing runtime. The mean and medium runtime is given with respect to the problems solved in 1 800 seconds.

- Figure 11-a shows that the type of observations has a clear impact on the runtime. Diagnostic problems using partially ordered observations are about 10 times harder to solve than other diagnostic problems.
- Figure 11-b illustrates the effect of the difficulty mentionned at the beginning of this section. In this case hard instances are also 10 times harder that the easy instances.
- Figure 11-c shows that the number of faults is also an important factor for solving the diagnostic problem. The complexity is more than linear with respect to this parameter.

C. Analysis

The SAT-based approach is able to compute a diagnosis for most problems. It fails to return a diagnosis only when all hard conditions are met: imprecise observations, a hard scenario and a big number of faults. These results show that our approach is suitable for solving diagnostic problems.

We now discuss performance and answer why certain parameters affect running time. We mentioned that the main power of SAT is its ability to propagate efficiently inferences. Consider again the DES in Figure 3 where u is unobservable and consider the sequence of ordered observations $[a, b]$. A diagnosis algorithm based on unfolding will first try to explain observation a and next b; therefore the algorithm will branch before finding that there is only one explanation. A SAT-based algorithm will immediately determine that observation b came from transition $2 \xrightarrow{b} 0$; therefore the solver will determine

[5] Experimental data is available at: www.grastien.net/ban/data/tac11/.

[6] Available at www.minisat.se.

that the state before $\xrightarrow{b}$ is state 2 and that observation a was emitted by transition $0 \xrightarrow{a} 2$. More generally, the SAT solver may instantiate the variables of the paths in any order.

If the timestep when an observable event occurred is known, then many inferences can be made. If the observable event o did not take place at timestep j, then no rule associated to o took place at time j. If the observable event o did take place at timestep j, then one of these rules (and no other) was triggered. If the precondition or the effects of the possible rules share similarities (or if there is only one rule possible), then inferences can be made on the state before the transition or after the transition. However if the timestep when the observable event occurred is unknown, inferences are more difficult. For instance in Figure 3 if the occurrence date of a is 1 and the occurrence date of b is in $\{2, 3\}$, then the SAT solver cannot automatically infer the transition associated to a.

This is typically the case with partially ordered observations: the solver cannot easily propagate the information. It is also illustrated with the difficulty of the scenario. When the scenario is easy the fact that a given component is not in state ○ is easy to determine and the list of components responsible for a given fault is much reduced.

Finally the last parameter is the number of faults k. This number affects the runtime in several ways:

- Because alarms are generated only when faults take place, the length of the scenario is linear in the number of faults (about $8 \times k$) which means the SAT instances have size at least linear in k. This feature holds only for this benchmark.
- The number of diagnostic questions to answer is linear in k. Admittedly however, the diagnostic question $\mathcal{Q}_i$ for $i \ll k$ is usually trivially false and solved with little or no search.
- The complexity of solving cardinality constraints increases quickly with k. The encoding we use [49] has size $o(n \log n)$ where n is linear in k (first point above). We studied the impact of cardinality constraints, and in particular how they should be encoded, in [43].

VII. Extensions

The work presented in this article is the first approach to solve diagnosis of DES problems using SAT. The approach described here is somewhat simplistic, addressing a simplistic diagnostic problem and using SAT in a very trivial manner. This section is dedicated to showing possible improvements in both directions.

A. Expressiveness

In this paper, we proposed to solve the problem of finding a preferred explanation to the observations. However, our approach is much more ambitious and aims at solving more general problems such as returning all possible sets of faults. We already took some steps in this direction [24], [25]. The diagnostic problem then translates as finding solution in a space of diagnostic hypotheses (each hypothesis being a potential diagnostic candidate), and the diagnostic questions must be wisely chosen to enable an efficient exploration of this space. Dealing with more complex definitions of a fault [18] defined as pattern of events, although not illustrated in this paper, is also feasable.

A SAT-based approach also allows to consider notions such as *conflicts* [50]. A conflict is a diagnostic information about the system behaviour, such as "a fault must have occurred in component c_1 or in component c_2". Being a different type of information about the system, a conflict can be useful to help a human operator understand the current situation.

We also want to extend our approach to a larger class of systems. Hybrid systems, i.e., systems with a compound of discrete and continuous behaviours, and in particular non-linear systems, are particularly difficult to diagnose. We want to use similar techniques based on SAT modulo Theories (SMT, [51]). SMT is an extension of SAT including additional constraints not expressed in terms of propositional logic but possibly as continuous contraints.

B. Complexity

The approach proposed in this article allows to use powerful tools to solve diagnostic problems that are computationally too hard to solve with traditional techniques. There are many directions that can be used to improve the results.

We already mentioned that the encoding of cardinality constraints has a huge impact on the SAT solver performance [43]. Using dedicated solvers specifically designed to solve SAT problems with cardinality constraints could also be an alternative.

On-line diagnosis consists in diagnosing the system while it is running. One of the issues associated with on-line diagnosis is updating the diagnosis when new observations are available. Such incremental diagnosis must be efficient (*length independent* [52]) lest the number of observations be too large to handle. One approach is to consider that the diagnosis related to old observations is now correct and only update the diagnosis on the last observations. Related issues include correctness and were investigated in [53].

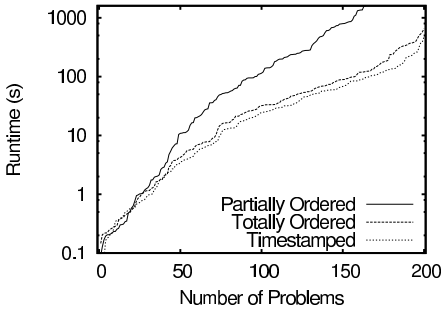
The SAT translations of the diagnostic questions associated with a given problem share many clauses. Techniques from incremental SAT [44], the problem of solving a SAT problem that share many clauses with an already solved SAT problem, should be used to improve performance.

Because we generated the SAT problem, we know the properties exhibited by these types of problems and we can design SAT solvers particularly efficient at solving them. A similar approach has been proposed for SAT planning [54].

We already mentionned that pre-processing can provide very useful clauses to insert in the SAT problem. A possible preprocess is inexpensive local diagnosis [9] that can be used to restrict the set of states and events at given timesteps.

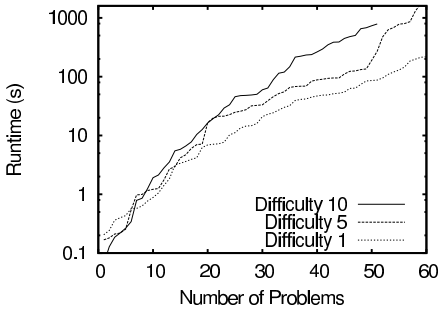
VIII. Conclusion

The approach for diagnosing DES in this paper is quite novel and quite exciting. Instead of computing all paths consistent with the observations, we propose to ask questions about the existence of certain paths explaining the observations. We proved that this approach can be quite efficient, and we believe



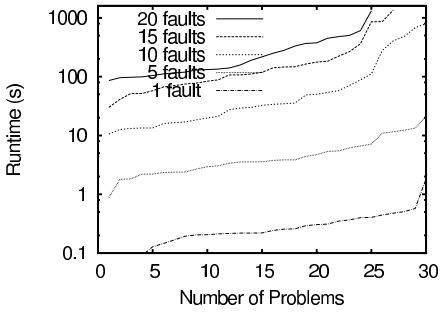
	Mn	Md	U
Tim.	52s	25s	0
T. O.	77s	32s	0
P. O.	506s	63s	34/200

a) Runtime for different observations



	Mn	Md	U
1	44s	23s	0
5	192s	33s	1/60
10	402s	48s	9/60

b) Runtime for different difficulties



	Mn	Md	U
1	<1s	<1s	0
5	5s	4s	0
10	117s	34s	0
15	381s	111s	3/30
20	530s	157s	5/30

c) Runtime for different number of faults

Fig. 11. Runtime for different parameters: Mn = mean time, Md = medium time, U = number of unsolved problems.

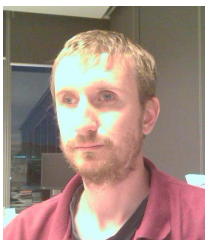
it can also be used for more general classes of systems, such as hybrid systems.

The use of SAT techniques is novel in diagnosis of DES. The strength of such algorithms is that the decomposition in Boolean variables makes it easier to infer informations through unit propagation and clause learning.

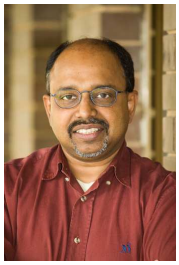
Application of such techniques for diagnosing the Smart Grid is investigated. Smart Grids involve thousands of components and non-linear continuous behaviours where SAT- and SMT-based approaches will be necessary to handle such constraints.

IX. ACKNOWLEDGEMENTS

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.



Alban Grastien got his Ph.D in 2005 at *Université de Rennes 1*, France. He moved to the *Canberra Research Laboratory* of NICTA (Australia) in 2006 where he is now a Senior Researcher in the Optimisation Research Group. He also holds an adjunct position in the AI Group at the Australian National University. Alban Grastien's main topic of research is the use of Artificial Intelligence paradigms and tools for diagnosis and diagnosability of large-scale discrete event and hybrid systems. His work is applied in the supervision of future energy systems.



Anbu Anbulagan received his Ph.D in Computer Science from *Université de Technologie de Compiègne* (UTC), France, in 1998. He was a research scientist at NICTA from 2003 to 2009. He was the founding director of Green Intelligence Software Solutions (Gissio) in Australia. He is currently an information analyst at the Australian National University. His research interests include SAT modeling and solving, optimisation, search in AI, big data analysis, and social network analysis.

REFERENCES

[1] W. Hamscher, L. Console, and J. de Kleer, *Readings in model-based diagnosis*. Morgan Kaufmann Publishers Inc., 1992.

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

[3] H. Kautz and B. Selman, "Pushing the envelope : planning, propositional logic, and stochastic search," in *Thirteenth Conference on Artificial Intelligence (AAAI-96)*, 1996, pp. 1194–1201.

[4] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model-checking without BDDs," in *Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-99)*, 1999, pp. 193–207.

[5] J. Rintanen and A. Grastien, "Diagnosability testing with satisfiability algorithms," in *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2007, pp. 532–537.

[6] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete-event systems," *IEEE Transactions on Automatic Control (TAC)*, vol. 40, no. 9, pp. 1555–1575, 1995.

[7] J. Rintanen, "Diagnosers and diagnosability of succinct transition systems," in *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, M. Veloso, Ed. AAAI Press, 2007, pp. 538–544.

[8] 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)*, vol. 164, pp. 121–170, 2005.

[9] R. Su and W. Wonham, "Global and local consistencies in distributed fault diagnosis for discrete-event systems," *IEEE Transactions on Automatic Control (TAC)*, vol. 50, no. 12, pp. 1923–1935, 2005.

[10] M.-O. Cordier and A. Grastien, "Exploiting independence in a decentralised and incremental approach of diagnosis," in *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2007.

[11] P. Kan John and A. Grastien, "Local consistency and junction tree for diagnosis of discrete-event systems," in *Eighteenth European Conference on Artificial Intelligence (ECAI'08)*, 2008, pp. 209–213.

[12] A. Schumann, Y. Pencolé, and S. Thiébaux, "A spectrum of symbolic on-line diagnosis approaches," in *Nineteenth National Conference on Artificial Intelligence (AAAI-07)*, R. Holte, Ed. AAAI Press, 2007.

[13] A. Benveniste, É. Fabre, S. Haar, and C. Jard, "Diagnosis of asynchronous discrete event systems, a net unfolding approach," *IEEE Transactions on Automatic Control (TAC)*, vol. 48, no. 5, pp. 714–727, May 2003.

[14] G. Jiroveanu, R.Boël, and B. Bordbar, "On-line monitoring of large Petri net models under partial observation," *Journal of Discrete Event Dynamical Systems (JDEDS)*, vol. 18, no. 3, pp. 323–354, 2008.

[15] M. P. Cabasino, A. Giua, and C. Seatzu, "Fault detection for discrete event systems using Petri nets with unobservable transitions," *Automatica (Automatica)*, vol. 46, no. 9, pp. 1531–1539, 2010.

[16] F. Basile, P. Chiacchio, and G. De Tommasi, "An efficient approach for online diagnosis of discrete event systems," *IEEE Transactions on Automatic Control (TAC)*, vol. 54, no. 4, pp. 748–759, 2009.

[17] M. Dotoli, M. P. Fanti, A. M. Mangini, and W. Ukovich, "On-line detection in discrete event systems by Petri nets and integer linear programming," *Automatica (Automatica)*, vol. 45, pp. 2665–2672, 2009.

[18] T. Jéron, H. Marchand, S. Pinchinat, and M.-O. Cordier, "Supervision patterns in discrete-event systems diagnosis," in *Seventeenth International Workshop on Principles of Diagnosis (DX-06)*, 2006, pp. 117–124.

[19] Y. Pencolé, D. Kamenetsky, and A. Schumann, "Towards low-cost diagnosis of component-based systems," in *IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes (SafeProcess-06)*, 2006.

[20] A. Grastien and G. Torta, "Reformulation for the diagnosis of discrete-event systems," in *21st International Workshop on Principles of Diagnosis (DX-10)*, 2010.

[21] L. Console, C. Picardi, and M. Ribaudo, "Diagnosis and diagnosability analysis using PEPA," in *Fourteenth European Conference on Artificial Intelligence (ECAI-00)*, 2000, pp. 131–135.

[22] L. Rozé and M.-O. Cordier, "Diagnosing discrete-event systems : extending the "diagnoser approach" to deal with telecommunication networks," *Journal of Discrete Event Dynamical Systems (JDEDS)*, vol. 12, no. 1, pp. 43–81, 2002.

[23] S. Jiang, R. Kumar, and H. Garcia, "Diagnosis of repeated/intermittent failures in discrete event systems," *IEEE Transactions on Robotics and Automation (TRA)*, vol. 19, no. 2, pp. 310–323, 2003.

[24] A. Grastien, P. Haslum, and S. Thiébaux, "Exhaustive diagnosis of discrete event systems through exploration of the hypothesis space," in *22nd International Workshop on Principles of Diagnosis (DX-11)*, 2011, pp. 60–67.

[25] A. Grastien, P. Haslum, and S. Thiébaux, "Conflict-based diagnosis of discrete event systems: Theory and practice," in *International Conference on Knowledge Representation (KR-12)*, 2012, to appear.

[26] A. Bauer, A. Botea, A. Grastien, P. Haslum, and J. Rintanen, "Alarm processing with model-based diagnosis of discrete event systems," in *22nd International Workshop on Principles of Diagnosis (DX-11)*, 2011, pp. 52–59.

[27] B. Williams and P. Nayak, "A model-based approach to reactive self-configuring systems," in *Thirteenth Conference on Artificial Intelligence (AAAI-96)*, 1996, pp. 971–978.

[28] A. Grastien, Anbulagan, J. Rintanen, and E. Kelareva, "Diagnosis of discrete-event systems using satisfiability algorithms," in *22nd Conference on Artificial Intelligence (AAAI-07)*, 2007.

[29] S. A. Cook, "The complexity of theorem proving procedures," in *Third ACM Symposium on Theory of Computing*, Ohio, USA, 1971, pp. 151–158.

[30] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving," *Communication of ACM*, vol. 5, pp. 394–397, 1962.

[31] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *38th Design Automation Conference (DAC-01)*, 2001, pp. 530–535.

[32] C. M. Li and Anbulagan, "Heuristics based on unit propagation for satisfiability problems," in *Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, 1997, pp. 366–371.

[33] L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik, "Efficient conflict driven learning in a Boolean satisfiability solver," in *International Conference on Computer Aided Design (ICCAD-01)*, 2001.

[34] N. Eén and N. Sorensson, "An extensible SAT-solver," in *Revised Selected Papers of Sixth Conference on Theory and Applications of Satisfiability Testing, LNCS 2919 Springer*, 2004, pp. 502–518.

[35] K. Pipatsrisawat and A. Darwiche, "Rsat 2.0: SAT solver description," Automated Reasoning Group, Computer Science Department, UCLA, Tech. Rep. D–153, 2007.

[36] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *21st International Joint Conference on Artificial Intelligence (IJCAI-09)*, Pasadena, California, USA, 2009, pp. 399–404.

[37] Anbulagan and J. Slaney, "Lookahead saturation with restriction for SAT," in *Eleventh International Conference on Principles and Practice of Constraint Programming (CP-05)*, 2005, pp. 727–731.

[38] O. Dubois and G. Dequen, "A backbone-search heuristic for efficient solving of hard 3-SAT formulae," in *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, Seattle, Washington, USA, 2001, pp. 248–253.

[39] M. Heule and H. van Maaren, "March_dl: Adding adaptive heuristics and a new branching strategy," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 2, pp. 47–59, 2006.

[40] S. Jiang, Z. Huang, V. Chandra, and R. Kumar, "A polynomial algorithm for diagnosability of discrete-event systems," *IEEE Transactions on Automatic Control (TAC)*, vol. 46, no. 8, pp. 1318–1321, 2001.

[41] J. Rintanen, "Planning with SAT, admissible heuristics and A*," in *22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, 2011, pp. 2015–2020.

[42] J. Huang, "The effect of restarts on the efficiency of clause learning," in *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, 2070, pp. 2318–2323.

[43] Anbulagan and A. Grastien, "Importance of variables semantic in CNF encoding of cardinality constraints," in *Eighth Symposium on Abstraction, Reformulation and Approximation (SARA-09)*, 2009.

[44] J. Hooker, "Solving the incremental satisfiability problem," *Journal of Logic Programming*, vol. 15, no. 1-2, pp. 177–186, 1993.

[45] J. Marques Silva and I. Lynce, "Towards robust CNF encodings of cardinality constraints," in *Thirteenth International Conference on Principles and Practice of Constraint Programming (CP-07)*, 2007, pp. 483–497.

[46] J. Rintanen, "Compact representation of sets of binary constraints," in *Seventeenth European Conference on Artificial Intelligence (ECAI-06)*, 2006, pp. 143–147.

[47] G. Jiroveanu and R. Boël, "Petri net model-based distributed diagnosis for large interacting systems," in *Sixteenth International Workshop on Principles of Diagnosis (DX-05)*, 2005, pp. 25–30.

[48] Y. Wang, T.-S. Yoo, and S. Lafortune, "New results on decentralized diagnosis of discrete event systems," in *42nd Annual Allerton Conference on Communication, Control, and Computing*, Illinois, USA, 2004.

[49] O. Bailleux and Y. Boufkhad, "Efficient CNF encoding of Boolean cardinality constraints," in *Ninth International Conference on Principles and Practice of Constraint Programming (CP-03)*, 2003, pp. 108–122.

[50] R. Reiter, "A theory of diagnosis from first principles," *Artificial Intelligence (AIJ)*, vol. 32, no. 1, pp. 57–95, 1987.

[51] L. de Moura, B. Dutertre, and N. Shankar, "A tutorial on satisfiability modulo theories," in *Nineteenth International Conference on Computer Aided Verification (CAV)*, 2007, pp. 20–36.

[52] A. Bauer and P. Haslum, "Ltl goal specifications revisited," in *Nineteenth European Conference on Artificial Intelligence (ECAI-10)*, 2010, pp. 881–886.

[53] A. Grastien and Anbulagan, "Incremental diagnosis of DES with a non-exhaustive diagnosis engine," in *20th International Workshop on Principles of Diagnosis (DX-09)*, 2009, pp. 345–352.

[54] J. Rintanen, "Heuristics for planning with SAT," in *Sixteenth International Conference on Principles and Practice of Constraint Programming (CP-10)*, 2010, pp. 414–428.