Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning

Conflict-Based Diagnosis of Discrete Event Systems: Theory and Practice

Alban Grastien[1,2] and **Patrik Haslum**[2,1] and **Sylvie Thiébaux**[2,1]
[1] Optimisation Research Group, NICTA
[2] Artificial Intelligence Group, Australian National University

Abstract

We present a conflict-based approach to diagnosing Discrete Event Systems (DES) which generalises Reiter's Diagnose algorithm to a much broader class of problems. This approach obviates the need to explicitly reconstruct the system's behaviors that are consistent with the observation, as is typical of existing DES diagnosis algorithms. Instead, our algorithm explores the space of diagnosis hypotheses, testing hypotheses for consistency, and generating conflicts which rule out successors and other portions of the search space. Under relatively mild assumptions, our algorithm correctly computes the set of preferred diagnosis candidates. We investigate efficient symbolic representations of the hypotheses space and provide a SAT-based implementation of this framework which is used to address a real-world problem in processing alarms for a power transmission system.

1 Introduction

Discrete Event Systems (DES) are a popular model for dynamic systems when the states and events can be represented with acceptable accuracy at a discrete level (Cassandras and Lafortune 1999). Starting from a DES model and an observation of the system's behavior (a trace of observable events), the purpose of model-based diagnosis is to determine if the behavior is nominal, and if not, the preferred (e.g. most likely) fault modes it exhibits.

DES diagnosis problems are traditionally solved by reconstructing all system traces consistent with the observation, either explicitly (Zanella and Lamperti 2003; Aghasaryan et al. 1998) or through a precompiled structure (Sampath et al. 1995). This is in spite of the fact that, in many realistic situations, the high-level classification of behavior into nominal and different fault modes is all that is needed, and reconstructing in complete detail all behaviors that may have taken place is superfluous. The time or space complexity of these approaches puts a tight limit on the size of systems that can be diagnosed. To monitor large networks consisting of thousands of components, such as power grids, a different approach is required.

In contrast, diagnosis of static systems is usually performed by efficiently exploring the hypothesis space by means of simple tests. For instance, the algorithm by Reiter (1987), repeatedly tests a *diagnosis hypothesis* for consistency against the system model and observation, either proving it to be a *diagnosis candidate* or deriving a *conflict* which summarises the reasons why the hypothesis was rejected. The conflict is then used to generate successor hypotheses to be tested, ruling out any hypothesis that will fail the test for the same reasons.

In this paper, we generalise the notion of conflict, and present a new conflict-based diagnosis approach that applies to a much broader class of systems, including DES. In our framework, a conflict represents a set of hypotheses which it excludes; the notion of conflict established by Reiter for static systems is a special case. In this sense, we unify the two previously disparate threads of diagnosis research.

Using a sequence of relatively simple tests, our approach correctly computes the set of all preferred diagnosis candidates (under a few assumptions on the hypothesis space), and does so without needing to explicitly represent all possible system behaviors. This results in substantial efficiency gains, compared to state-of-the-art DES diagnosis methods. Conflicts help reduce the part of the hypothesis space that needs to be tested, but they also enable the presentation of new types of diagnostic information to users, e.g. that the occurrence of a certain fault excludes or implies that of another, or that some fault definitely occured before another.

Any practical implementation of our approach relies on representing and manipulating sets of hypotheses symbolically, i.e., via sets of properties. We state the requirements that a property space used to implement the algorithm must satisfy, and we identify a suitable property space. For DES, we provide a SAT-based implementation, where hypotheses are represented by Boolean formulae over atomic properties and consistency tests are performed by a SAT solver. We demonstrate the improved efficiency of our approach on the problem of processing alarms generated by a power transmission system operated by TransGrid, an Australian utility.

Summing up, our contributions are (1) a general characterisation of conflicts, (2) a conflict-based diagnosis algorithm for a broad class of systems, (3) the identification of the principles the symbolic representation must satisfy to facilitate an efficient implementation, and (4) a SAT implementation of the framework for DES systems which is used to solve a real-world problem.

The paper is organised as follows. Section 2 introduces a small example which will be used for illustration throughout the paper. Section 3 provides background on Reiter's con-

flict based algorithms and on DES. Section 4 defines the general conflict-based diagnosis framework and describes the conflict-based search algorithm. Section 5 establishes principles that an adequate symbolic representation of hypothesis sets should satisfy. Section 6 presents a SAT based implementation of the approach for DES, and Section 7 demonstrates its application to the alarm processing problem.

2 Running Example

This work was motivated by the monitoring of incidents on power networks. When incidents occur, the control room receives a flood of alarms that is often difficult for human operators to comprehend. We want to help them by providing minimal explanations of the alarm cascade.

We present a simple example that will be used for illustration throughout the paper; we will of course solve much more challenging problems in our Experiments section. Consider an electric line protected by a circuit breaker with single reclosing policy. The line may suffer from a permanent fault (line damaged) or a transient fault (electric arc between lines). If the fault is transient, then opening and closing the circuit breaker suffices to clear it; otherwise, the fault is permanent and cannot be cleared. The third possible fault is a temporary failure of the circuit breaker to open or close when commanded to; failure to open or to close are modeled by different events, but not distinguished in the diagnosis, which reports both as "failure to operate".

The reclosing policy is the logic that rules the opening and closing of the circuit breaker. When a fault is detected, it forces the circuit breaker to open, then recloses a short time later. If, on reclosing, the fault is still detected, the recloser assumes it is permanent and opens the breaker permanently. Otherwise, it confirms the clearance of the transient fault.

Models of the system components are shown in Figure 1. The 5 states of the circuit-breaker+line subsystem encode the fault mode of the line (transient=T, permanent=P or no fault=N) and the position (open=O or closed=C) of the breaker. The states of the recloser represent the various stages of the execution of the reclosing policy (OK, fault detected=F, opening=OP, reclosing=REC, permanent fault detected= F2, and permanent opening=OP2). In the circuit-breaker+line model, events t, p, o, c, fo, and fc represent "transient fault", "permanent fault", "open", "close", "failed to open", and "failed to close", respectively. The *open* (resp. *close*) event of the recloser synchronizes with the "open" event or the "failed to open" event (resp. the "close" event or the "failed to close" event) of the breaker, depending of the success of the command. The *no_fault* (resp. *detected*) events of the recloser are generated if the circuit breaker state is NC, NO, or PO (resp. TC or PC).

Events generated by the recloser are observable. Suppose we observe the sequence $[detected, open, close, no\_fault]$. There are two simple explanations: either a transient fault occurred, or a permanent fault occurred followed by the breaker failing to close. More complex explanations include a transient fault followed by a failure to close or a transient or permanent line fault (not yet detected by the sensors).

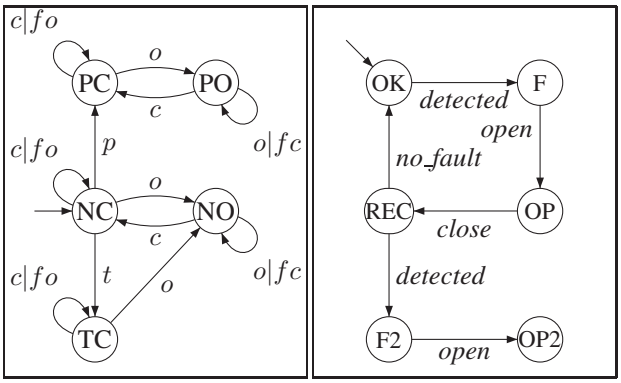

Figure 1: Models of the example: left, the circuit breaker and the line; right, the recloser.

3 Background

This section presents the well-known static diagnosis definitions and algorithms, and formulates the diagnosis of discrete event systems in such a way that the similarities between the two tasks becomes apparent.

3.1 Static Diagnosis

In the classical framework of diagnosis of static systems, a *diagnosis problem* is defined as a tuple $\mathsf{P} = \langle SD, Comps, Obs\rangle$ where *SD* is a model of the system expressed in first-order logic, *Comps* is the set of components, and *Obs* is an observation. The literals in *SD* include health variables $\mathrm{Ab}(c)$, for each component $c \in Comps$; these literals evaluate to *true* if the component c is faulty (*ab*normal). In the simplest case, the model represents only the nominal behaviour, meaning *SD* is a conjunction of formulas of the form $(\neg\mathrm{Ab}(c) \rightarrow \phi)$ where ϕ is a formula without Ab literal.

The task is to recover the current *health assignment* of the system, defined as the subset $\delta \subseteq Comps$ of components that are faulty; other components ($Comps \setminus \delta$) are considered non-faulty. Health assignment δ is a *diagnosis candidate* for diagnosis problem $\mathsf{P} = \langle SD, Comps, Obs\rangle$ iff it is consistent with the model and the observation, i.e.,

$$SD, Obs, \{\neg\mathrm{Ab}(c) \mid c \in Comps \setminus \delta\} \not\models \bot. \qquad (1)$$

The set of diagnosis candidates of problem P is denoted by $\Delta(\mathsf{P})$. Typically, diagnosis candidates that assume more faults than what is necessary to explain observation *Obs* are not considered interesting. We prune those uninteresting candidates. Thus, the task of the diagnoser is to find the *minimal diagnosis* $\Delta_{\min}(\mathsf{P})$, defined as the set of candidates that are minimal w.r.t. set inclusion, i.e.,

$$\Delta_{\min}(\mathsf{P}) = \min_{\subseteq}(\Delta(\mathsf{P}))$$

The set of health assignments is the power set of *Comps*. Therefore testing all health assignments is not feasible in general; the preferred approach is based on conflicts. A set of components C forms a *conflict* for problem P iff at least one of them must be faulty, i.e.,

$$SD, Obs, \{\neg\mathrm{Ab}(c) \mid c \in \mathsf{C}\} \models \bot. \qquad (2)$$

In other words, all of them being in nominal condition, contradicts the model and observation. Reiter (1987) proved that the (minimal) diagnosis candidates can be computed as the set of (minimal) hitting sets of all conflicts.

Explicitly generating all conflicts upfront is unpractical but conflict generation can be interleaved with the hitting set calculation. Consider Equation (1): when it does not hold, a contradiction is generated, as in Equation (2). The logical derivation leading to $\bot$ can be studied to determine the set of components involved in the contradiction. This set of components is by definition a conflict.

The diagnosis algorithm `diagnose` proposed by Reiter (1987) is therefore the following: starting from $\delta = \emptyset$, i.e., the unique minimal health assignment, the algorithm tests whether δ is a diagnosis candidate. If the current health assignment δ is not a diagnosis candidate, then a conflict C is derived and the following successors of δ are generated:

$$\{\delta_c = \delta \cup \{c\} \mid c \in \mathsf{C}\}.$$

Each successor is in turn tested. Two enhancements avoid the consideration of spurious hypotheses. First, in case a health assignment δ fails to hit a previously found conflict C, i.e., if $\delta \cap \mathsf{C} = \emptyset$, then C could be derived also from testing δ; therefore, the test is not applied and successors of δ are generated immediately. Second, if δ and δ' such that $\delta \subset \delta'$ have been generated, then δ' is not tested before δ, and δ' will be tested only if δ is not a candidate.

3.2 Diagnosis of Discrete Event Systems

A discrete event system is a model of a dynamic system which describes its possible evolutions in terms of the sequences of events that can occur. A DES is essentially a language over a (finite) set of events Σ.

Before presenting the diagnosis of DES, we introduce the following notations. A language $\mathcal{L}$ over an alphabet E is a (possibly infinite) subset of (finite) sequences of letters in E: $\mathcal{L} \subseteq E^\star$. The projection of language $\mathcal{L}$ over E onto the alphabet $E' \subseteq E$ is the language $Pr_{E'}(\mathcal{L})$ defined as the set of words of $\mathcal{L}$ where the letters from $E \setminus E'$ have been removed. The projection $Pr_{E'}(\sigma)$ of a single sequence σ on alphabet E' is similarly defined as the sequence where elements outside E' are omitted.

We now turn to the diagnosis problem. The first input to the problem is a DES model *Mod*, which is a complete description of all possible sequences of events that can occur. *Mod* defines a language $\mathcal{L}(Mod) \subseteq \Sigma^\star$.

The second input to the diagnosis problem is an observation. As the system evolves, it takes a sequence of events $\sigma \in \mathcal{L}(Mod)$. This sequence is partially observed, e.g. through sensors, alarms and log messages. This is modeled by distinguishing a subset $\Sigma_o \subseteq \Sigma$ of *observable events*, and letting the observation of the sequence σ be $o = Pr_{\Sigma_o}(\sigma)$.

Finally the last input to the diagnosis problem is a subset $\mathcal{F}$ of events whose occurrence we want to diagnose (this includes faults and other unobservable events of interest).

The purpose of diagnosis is to determine the sequences δ of events from $\mathcal{F}$ which are consistent with the observation, i.e., such that:

$$\exists \sigma \in \Sigma^\star : \sigma \in Mod \ \wedge \ Pr_{\Sigma_o}(\sigma) = o \ \wedge \ Pr_{\mathcal{F}}(\sigma) = \delta. \quad (3)$$

Equation (3) is to DES what Equation (1) is to static systems. Similarly to the static framework, we are interested in the minimal diagnosis candidates, defined as the candidates such that no proper subsequence are candidates.

In our running example, we take $\mathcal{F} = \{t, p, f\}$, meaning transient ($t$) or permanent ($p$) line fault, and intermittent failure of the breaker to operate (f, corresponding to either of events fo or fc in Figure 1).

4 Generalized Conflict-Based Diagnosis

We now present a generalisation of the conflict-based diagnosis approach. This generalisation will include both static diagnosis and diagnosis of discrete event systems as special cases, and we will use examples from both domains to illustrate some of the concepts thereafter.

This generalisation will require the manipulation of large – or even infinite – sets. Of course, a naive implementation of this approach is impractical; the next section is dedicated to the efficient implicit (symbolic) manipulation of such sets.

4.1 Diagnosis Problem

There are three components in a (generalised) diagnosis problem.

The system model defines the set of possible system behaviors. We will write M for the model and $\sigma \in M$ to state that the model allows for behavior σ.

The observation is a statement about the actual system behavior. Even together with the model, the observation is usually not sufficient to precisely retrieve the system behavior. We write o for the observation, and $o = obs(\sigma)$ if behavior σ generates observation o.

The last input to the diagnosis problem is a set $\mathbb{H}$ of possible solutions. We call this set the *hypothesis space*. In the static framework, the hypothesis space is the power set of *Comps*. In the DES framework, the hypothesis space is the Kleene closure $\mathcal{F}^\star$; for instance, assuming $\mathcal{F} = \{f, p, t\}$, the (infinite) set of hypotheses is $\mathbb{H} = \{[], [f], [p], [t], [f, f], [f, p], \dots\}$.

The hypothesis space is an implicit partition of the set of behaviors, i.e., there exists a function hypo that associates each behavior σ with exactly one hypothesis $\mathrm{hypo}(\sigma) \in \mathbb{H}$. For instance, in the static framework, a complete behavior is represented by an assignment of all state variables, and the hypothesis associated with this behavior is the restriction of this assignment to the Ab literals. In the DES framework, the hypothesis associated with a sequence of events σ is the projection of σ on $\mathcal{F}$.

Definition 1 *The* diagnosis problem *is the triple* $\langle M, o, \mathbb{H}\rangle$ *as presented above.*

The solution of the diagnosis problem is the set of hypotheses δ such that there exists a behavior σ i) authorized by the model, ii) that generates the observation, and iii) whose hypothesis is δ. (We will write δ for a diagnosis candidate, and h for a hypothesis which may not be a candidate.)

Definition 2 *The* diagnosis $\Delta(\langle M, o, \mathbb{H}\rangle)$ *of a diagnosis problem is the set of hypotheses:*

$$\{\delta \in \mathbb{H} \mid \exists \sigma \in M : obs(\sigma) = o \ \wedge \ \mathrm{hypo}(\sigma) = \delta\}. \quad (4)$$

In the following, we assume the diagnosis problem is fixed, and simply write Δ for the diagnosis.

The hypothesis space is further equipped with a partial order relation $\preceq$ that indicates that if two different candidates δ and δ' are such that $\delta \preceq \delta'$, then δ is preferred to δ' (e.g., because it is more plausible, hypothesises fewer faults), and δ' can be ignored in the minimal diagnosis. Thus, the objective of diagnosis is to find the set $\Delta_{\min}$ of candidates that are minimal according to this order. For every ignored δ', there should be $\delta \in \Delta_{\min}$ such that $\delta \preceq \delta'$; conversely, for any candidate δ' such that there exists another candidate δ with $\delta \preceq \delta'$, then δ' should not be in $\Delta_{\min}$. In the static diagnosis framework, where $\mathbb{H} = 2^{Comps}$, this order is $\subseteq$. In the DES framework, we have $\delta \preceq \delta'$ iff δ is a subsequence of δ'.

Definition 3 *The* minimal diagnosis *is* $\Delta_{\min} \subseteq \Delta$ *such that*

(i) $\forall \delta \in \Delta,\ \exists \delta' \in \Delta_{\min} : \delta' \preceq \delta$.

(ii) $\forall \delta \in \Delta_{\min},\ \forall \delta' \in \Delta,\ \delta' \preceq \delta \Rightarrow \delta = \delta'$.

We assume that $\preceq$ is a *well partial order* on $\mathbb{H}$, meaning that for any non-empty $H \subseteq \mathbb{H}$, the set $\min_{\preceq}(H)$ is non-empty and finite. This ensures that the minimal diagnosis exists and is finite.[1] The two examples given above (subsets and subsequence) are both well partial orders (Nash-Williams 1963), as are many other natural orders on hypothesis spaces. That $\preceq$ is a well partial order also implies other important properties, e.g., that the set of children of a hypothesis (defined below) is finite.

To simplify notation, we assume there is a single minimal hypothesis $h_0 \in \mathbb{H}$, i.e., $\forall h \in \mathbb{H},\ h_o \preceq h$. This assumption can easily be lifted, because $\preceq$ being a well partial order ensures that $\min_{\preceq}(\mathbb{H})$ is non-empty and finite.

Notations Given hypothesis $h \in \mathbb{H}$, we call:

- *descendants* of h, the set $\mathrm{desc}(h) = \{h' \in \mathbb{H} \mid h \preceq h'\}$;
- *children* of h, the set $\mathrm{chi}(h) = \min_{\preceq}(\mathrm{desc}(h) \setminus \{h\})$.

Ancestors (anc) and parents (par) are the inverse operations of descendants and children.

For instance in the DES framework, where $\mathcal{F} = \{f, p, t\}$ and $h \preceq h'$ iff h is a subsequence of h', hypothesis $[f, f]$ is a descendant and a child of hypothesis $[f]$; therefore $[f]$ is an ancestor and a parent of $[f, f]$. On the other hand, $[f, t, p]$ is a descendant of $[f]$, $[t]$, and $[p]$, but is a child of none of them; the parents of $[f, t, p]$ are $[f, t]$, $[f, p]$, and $[t, p]$.

Given two hypotheses, we write $h_1 \otimes h_2$ the set of minimal hypotheses that are descendant of h_1 and h_2. For instance, if $\mathbb{H} = \mathcal{F}^\star$, $h_1 = [p, t]$, and $h_2 = [t, f]$, then $h_1 \otimes h_2 = \{[p, t, f], [t, f, p, t]\}$. For this hypothesis space, the operation $\otimes$ can easily be implemented by intertwining elements of h_1 and h_2 while carefully avoiding non-minimal combinations (such as $[p, t, f, t]$ in the example). With slight abuse of notation, we write $h_1 \otimes H_2$ for $\bigcup_{h_2 \in H_2}(h_1 \otimes h_2)$.

[1] To ensure that the minimal diagnosis exists, it is sufficient to assume that $\preceq$ is well-founded, which is a slightly weaker requirement; however, if $\preceq$ is well-founded but not a well partial order, $\Delta_{\min}$ can be infinite, which obviously makes presenting it explicitly impossible.

4.2 The Preferred-First Algorithm

We now present an algorithm for solving a diagnosis problem as formulated before. The algorithm we present relies primarily on an operation called "test". A test is an operation that determines whether a given set of hypotheses contains a diagnosis candidate.

Definition 4 *Given a diagnosis problem* $\langle M, o, \mathbb{H}\rangle$ *and a set* $H \subseteq \mathbb{H}$ *of hypotheses, a* test *is the problem of determining whether*

$$H \cap \Delta(\langle M, o, \mathbb{H}\rangle) \neq \emptyset. \tag{5}$$

We call "test solver" the machine that decides the result of a test, and assume a test solver is available. (A practical implementation of a test solver, based on SAT, is presented in section 6.) A test is said to be "successful" if Condition (5) holds; otherwise, it "fails". In the static framework, Condition (1) is an example of a test where $H = \{h \in 2^{Comps} \mid h \subseteq \delta\}$.

The preferred-first algorithm is shown in Algorithm 1. We first describe it without conflicts, and add them in Section 4.3. Like Reiter's `diagnose`, the algorithm starts from the preferred hypothesis h_0, and repeatedly tests whether a hypothesis is a candidate (Line 10); if h is a candidate, then h is added to the result; otherwise, the children of h are stored in the `open` list to be tested.

An important difference with `diagnose` is that we deal with a potentially infinite hypothesis space. There is a risk that the algorithm tries to test an infinite sequence $h_1, h_2, \ldots$ of hypotheses such that h_{i+1} is a child of h_i for all i. However, if all minimal candidates have a finite "depth", then there is eventually a value i such that h_i (as well as all its descendants) is an ancestor of no minimal candidate; such "undesirable" hypotheses h_i can be removed from the open list and the infinite sequence of tests is prevented.

It is not possible to define a test which decides that a given hypothesis h is undesirable. We propose instead to identify a much wider class of hypotheses (called the *non-essential* hypotheses) but that will not affect the correctness of the diagnosis algorithm. The set `local` tested on Line 7 corresponds to the set of hypotheses that are descendant of h but that are not descendant of any other hypothesis from `res`$\cup$`open`. If `local` contains no candidate (i.e., if the test fails), then it means either that h is undesirable or that, for all minimal candidates δ descendant of h, δ is a descendant of another hypothesis h' from `res`$\cup$`open`; in both cases, h can be removed from `open` as it does not reduce the set of minimal candidates reachable from `res` $\cup$ `open`.

Illustration of the algorithm execution We now illustrate the execution of algorithm with the example presented in Section 2 where $\mathbb{H} = \mathcal{F}^\star$ and $\mathcal{F} = \{f, p, t\}$. The diagnosis is $\Delta = \{[t], [p, f], [t, f], [t, t], [t, p]\}$, and the minimal diagnosis is $\Delta_{\min} = \{[t], [p, f]\}$ (that is, either a transient fault occurred, or a permanent one followed by a failure to operate). The algorithm starts with `res` $= \emptyset$ and `open` $= \{[]\}$.

At the first iteration, hypothesis $[]$ is chosen; `local` $= \mathbb{H}$ and the first test succeeds. The second test fails since $[]$ is not a candidate, and $\{[f], [t], [p]\}$ is added to `open`.

Assume now $[f]$ is chosen. The set `local` of hypotheses is now the set of hypotheses that descend from $[f]$

Algorithm 1 Preferred-First Algorithm

```
1: input hypotheses H
2: open := {h_0}
3: res := ∅
4: while open is not empty do
5:    h := pop(open)
6:    Let local = desc(h) \ desc(res ∪ open)
7:    if ¬test(local) then
8:       continue
9:    end if
10:   if test({h}) then
11:      res := res ∪ {h}
12:   else
13:      open := open ∪ chi(h)
14:   end if
15: end while
16: return res
```

and descend neither from $[p]$ or $[t]$. Therefore, $\texttt{local} = \{[f], [f,f], [f,f,f], \dots\}$ There is no candidate in local, and hypothesis $[f]$ is therefore ignored. Notice that $[f]$ is ignored although several of its descendants are candidates (and one of them is even minimal); these candidates will be covered by the hypotheses that are in open. Notice also that if this test was not performed, all hypotheses $[f]$, $[f,f]$, $[f,f,f]$, etc. would be tested and the algorithm would never end. At this stage, $\texttt{res} = \emptyset$ and $\texttt{open} = \{[t],[p]\}$.

Assume now $[t]$ is chosen. local contains all hypotheses that mention t at least once and that do not mention p; local contains the three candidates $[t]$, $[t,f]$, and $[t,t]$. Next, $[t]$ is tested and proved to be a candidate. At this stage, $\texttt{res} = \{[t]\}$ and $\texttt{open} = \{[p]\}$.

Hypothesis $[p]$ is next chosen, leading to five new hypotheses $\{[f,p],[t,p],[p,f],[p,t],[p,p]\}$. The third one is proved to be a candidate and all others are ignored because their descendants do not include any minimal candidate. This leads to $\texttt{res} = \{[t],[p,f]\}$ and $\texttt{open} = \emptyset$.

Correctness of the algorithm To prove Algorithm 1 is correct, we first prove the following lemma:

Lemma 1 *The following proposition always holds when the condition of the loop (Line 4) is evaluated:*

$$(\mathit{res} \subseteq \Delta_{\min}) \wedge (\Delta \subseteq (\mathrm{desc}(\mathit{open} \cup \mathit{res}))) . \qquad (6)$$

Proof Sketch for Lemma 1: We prove this result inductively. The proposition clearly holds initially since $\texttt{res} = \emptyset$ and $\texttt{open} = \{h_0\}$.

First, observe that local represents the descendants of h that disappear from $\mathrm{desc}(\texttt{res} \cup \texttt{open})$ when h is removed from open. If local does not intersect Δ, then the second inclusion of (6) remains satisfied.

At each iteration step, there are 3 cases:

1. The first test (Line 7) fails. res is unchanged and the first conjunct of the proposition still holds. Because the test failed, we are in the situation above where local does not intersect Δ and the second inclusion remains satisfied.
2. Both tests succeed. h is a candidate and is moved from open to res, hence the second inclusion is still satisfied. As for the first, if h is not minimal then another hypothesis in $\mathrm{desc}(\texttt{res} \cup \texttt{open})$ covers it, contradicting the assumption that the first test succeeded.
3. Only the first test succeeds. As in case 1. res is unchanged and the first conjunct still holds. Since $\preceq$ is a well partial order, $\mathrm{desc}(\texttt{h}) = \mathrm{desc}(\mathrm{chi}(\texttt{h})) \cup \{\texttt{h}\}$; in other words, $\mathrm{desc}(\texttt{res} \cup \texttt{open})$ stays unchanged except for the removal of h, which does not belong to Δ. Hence, the second conjunct is still satisfied.

□

Theorem 1 *Algorithm 1 returns the minimal diagnosis.*

Proof of Theorem 1: Assume that, at the end of the algorithm (i.e., when $\texttt{open} = \emptyset$), $\texttt{res} \neq \Delta_{\min}$. Because $\texttt{res} \subseteq \Delta_{\min}$ (by Lemma 1), there exists $\delta \in (\Delta_{\min} \setminus \texttt{res})$. Notice that if $\texttt{open} = \emptyset$, then $\mathrm{desc}(\texttt{res} \cup \texttt{open}) = \mathrm{desc}(\texttt{res})$; therefore by Lemma 1, $\Delta \subseteq \mathrm{desc}(\texttt{res})$, which means that there exists $h \in \texttt{res}$ such that $h \preceq \delta$. Because $\texttt{res} \subseteq \Delta_{\min}$, h is a diagnosis candidate. Therefore, δ is not a minimal candidate. □

Termination of the algorithm Termination of the algorithm depends on a property on the depth of the hypotheses.

Definition 5 *The* depth *of hypothesis h is the maximal number k such that there exists hypotheses $h_1, \dots, h_k$ and $h_0 \prec h_1 \prec \cdots \prec h_k = h$.*

In many cases, e.g. with orders defining minimal sets of faulty components or minimal sequences of events as considered so far, but also those defining minimal cardinality sets and minimal multisets of events, every hypothesis has a finite depth. It is possible however to define sensible hypothesis spaces where some hypotheses have infinite depth. Assume for instance the set of events $\mathcal{F} = \{n, f\}$ such that n is a nominal event and f is a faulty event. Assume that $h \preceq h'$ if h contains strictly fewer occurrences of f than hypothesis h'. For instance, in this context, $[n,n,n,n] \preceq [f]$. Then hypothesis $[f]$ has infinite depth. Indeed, $[] \prec [n] \prec [n,n] \prec [n,n,n] \prec \cdots \prec [f]$.[2] Practically, assuming the minimal diagnosis is $\{[f]\}$, Algorithm 1 will successively test $[]$, $[n]$, $[n,n]$, $[n,n,n]$, etc. and will never test $[f]$.

Theorem 2 *Assuming each hypothesis has a finite depth, Algorithm 1 terminates.*

Proof Sketch for Theorem 2: The proof relies on three arguments based on the following set:

$$H = \mathrm{anc}(\Delta_{\min}) \cup \mathrm{chi}(\mathrm{anc}(\Delta_{\min})).$$

First, H can be shown finite. Second, it can be shown that at any stage $\texttt{open} \subseteq H$. (By contradiction, if $h' \in \mathrm{chi}(h) \setminus H$ is added to open, then h is either non-essential or h is a candidate – and h' should therefore not be added to open). Third, it can be shown that the set open cannot have the same value more than once.

[2] Notice that this space has hypotheses of infinite depth despite being a well partial order.

In summary, at each iteration of the loop, `open` is a different subset of finite set H. Hence, after a finite number of steps, `open` becomes empty and the algorithm terminates. □

4.3 Conflicts

The algorithm used in the static framework is very similar to Algorithm 1 but differs in the use of conflicts (Reiter 1987; Pulido and Alonso González 2004). A conflict can be used to reject more than one hypothesis, thus avoiding many tests.

A conflict is a generalisation of a test failure. In this paper, we define a conflict as a set of hypotheses all of which have been identified as non-candidates by the test solver.

Definition 6 *A* conflict *is an object* C *that implicitly represents a set* $\mathrm{hypos}(\mathsf{C})$ *of non-candidate hypotheses, i.e.,* $\mathrm{hypos}(\mathsf{C}) \cap \Delta = \emptyset$.

The classical notion of a conflict used for static systems can be seen as an instance of this more general definition. In static systems, a conflict is a set of components $\mathsf{C} \subseteq \mathit{Comps}$ one of which must be faulty; the conflict excludes all hypotheses that assume none of these components are faulty: formally $\mathrm{hypos}(\mathsf{C}) = \{h \in \mathbb{H} \mid h \cap \mathsf{C} = \emptyset\}$. For instance, if $\mathit{Comps} = \{c_1, \ldots, c_5\}$ and $\mathsf{C} = \{c_1, c_2, c_3\}$, then C rejects the hypotheses corresponding to the sets of faulty components $\emptyset$, $\{c_4\}$, $\{c_5\}$, and $\{c_4, c_5\}$.

We assume that, if a given set of hypotheses $H \subseteq \mathbb{H}$ contains no candidate, then the test solver is not only able to determine that the test of H will fail, but it is able to return a superset of H that contains no candidate: a conflict. (In the worst case, the solver may simply return the conflict C such that $\mathrm{hypos}(\mathsf{C}) = H$.) The practical computation of such a conflict is the topic of the next section.

We can use this conflict to improve Algorithm 1 in two ways. First, because a conflict does not intersect Δ, then if the current hypothesis `h` belongs to a previously-found conflict, we know that this hypothesis is not a candidate and the test on Line 10 may be skipped.

Second, we can use the conflict to refine the set of successors to be considered:

$$\min_{\preceq} \left(\mathrm{desc}(h) \setminus \mathrm{hypos}(\mathsf{C})\right).$$

Observe that the set of successors used in Algorithm 1 is a special case where $\mathrm{hypos}(\mathsf{C}) = \{h\}$. Also, note that the conflict does not just prune some children of h from the successor set, but might also lead to deeper descendants of h being considered. Consider for instance hypothesis $h = [f]$ and conflict $\mathrm{hypos}(\mathsf{C}_1) = \{[f], [f, p]\}$. The successors of $[f]$ through conflict C_1 are $[f, f]$, $[f, t]$, $[p, f]$, $[t, f]$, and $[f, p, p]$. Consider now conflict $\mathrm{hypos}(\mathsf{C}_2) = \{[f], [f, p], [f, p, p], [f, p, p, p], \ldots\}$ (i.e., all sequences consisting of f followed by any number of ps). The successors of $[f]$ through conflict C_2 are $[f, f]$, $[f, t]$, $[p, f]$, and $[t, f]$.

5 Symbolic Representation

In this section, we take advantage of finite sets of properties to implicitly represent and manipulate infinite hypothesis sets. For instance, the hypotheses $\{[t], [f, t], [p, t], [t, f], [t, t], \ldots\}$ may be represented as sharing the "property" that they contain at least one occurrence of t. We provide a principled way to choose properties so as to efficiently represent and test hypothesis sets encountered in Algorithm 1 and easily compute successor sets. Specifically the computation of `local` in lines 6, the tests in lines 7 and 10, and the computation of successors in line 13, are all performed symbolically.

Definition 7 *A* property p *is an object that implicitely represents the set of hypotheses* $\mathrm{hypos}(p)$ *that "satisfy" this property. A set of properties* P *represents the set of hypotheses* $\mathrm{hypos}(P)$ *that satisfy all properties in* P*:*

$$\mathrm{hypos}(P) = \bigcap_{p \in P} \mathrm{hypos}(p).$$

A set of properties can therefore be interpreted as a conjunction. We write $\mathrm{props}(h)$, the set of properties that hypothesis h satisfies, i.e., $\mathrm{props}(h) = \{p \mid h \in \mathrm{hypos}(p)\}$.

5.1 Representation of Tested Hypotheses

We start by examing how to represent the hypothesis sets to be tested. Let us write $\mathbb{P}$ for the set of properties that will be used to do so. $\mathbb{P}$ needs to be expressive enough to represent the sets `local` and $\{\mathrm{h}\}$. Observe that both sets are defined as $\mathrm{desc}(h) \setminus \mathrm{desc}(H)$ for some hypothesis h and some finite set of hypotheses H (in the second case, H can be defined as the set $\mathrm{chi}(h)$).

Requirement 1 *The property space* $\mathbb{P}$ *must be defined such that for any hypothesis* h *and any finite set of hypotheses* H*, there exists a finite set of properties* $P \subseteq \mathbb{P}$ *such that* $\mathrm{hypos}(P) = \mathrm{desc}(h) \setminus \mathrm{desc}(H)$.

Theorem 3 *There exists a property space that satisfies Requirement 1.*

Proof of Theorem 3: A property space that satisfies the requirement can be constructed using 2 atomic properties p_{desc} and $p_{\not\mathrm{desc}}$ per hypothesis, representing the set of hypotheses that are descendant of the given hypothesis, and the set of hypotheses that are *not* among its descendants, respectively:

$$\mathrm{hypos}(p_{\mathrm{desc}}(h)) = \mathrm{desc}(h)$$

and

$$\mathrm{hypos}(p_{\not\mathrm{desc}}(h)) = \mathbb{H} \setminus \mathrm{desc}(h).$$

Looking at $\mathrm{desc}(h) \setminus \mathrm{desc}(H)$, hypotheses $\mathrm{desc}(h)$ are directly captured by $p_{\mathrm{desc}}(h)$. Moreover, H being finite, the subtraction of $\mathrm{desc}(H)$ can be captured by a finite set of $p_{\not\mathrm{desc}}(h')$ atoms. Hence $\{p_{\mathrm{desc}}(h)\} \cup \{p_{\not\mathrm{desc}}(h') \mid h' \in H\}$ is a finite representation of $\mathrm{desc}(h) \setminus \mathrm{desc}(H)$. □

Observe that $\mathrm{desc}(h) \setminus \mathrm{desc}(H)$ can be represented by $\{p_{\mathrm{desc}}(h)\} \cup \{p_{\not\mathrm{desc}}(h') \mid h' \in \min_{\preceq}(H)\}$ which (a) is more compact (but requires to compute $\min_{\preceq}(H)$) and (b) allows to generalise the requirement in case H is infinite ($\min_{\preceq}(H)$ is always finite since we consider well partial orders).

Hence, in the following, we use the property space:

$$\mathbb{P} = \{p_{\mathrm{desc}}(h) \mid h \in \mathbb{H}\} \cup \{p_{\not\mathrm{desc}}(h) \mid h \in \mathbb{H}\}.$$

The property space used in the static diagnosis framework is a special case. For any component $c \in \mathit{Comps}$, the property $\neg\mathrm{Ab}(c)$ (the property that component c is nominal) equates

to $p_{\not\text{desc}}(\{c\})$. Other properties $p_{\not\text{desc}}(h)$ for larger sets of components are not used because they can be defined as conjunctions of the defined properties. The properties $p_{\text{desc}}(h)$ are not used in the weak-fault model because they cannot be used to infer anything.

In the Discrete Event System framework, hypotheses are sequences of fault events, and the descendants of a hypothesis h are those that contain h as a subsequence. Thus, for example, property $p_{\text{desc}}([t])$ represents the descendants of $[t]$, i.e., all hypotheses that contain at least one occurrence of t. The property $p_{\not\text{desc}}([t])$ represents the hypotheses that are not descendants of $[t]$, i.e., that do not mention t, such as $[]$, $[f]$, $[p]$, $[f, f]$, etc. The hypothesis $h = [t]$ can be represented by the following set of properties P: $p_{\text{desc}}([t])$, $p_{\not\text{desc}}([t, t])$, $p_{\not\text{desc}}([t, f])$, $p_{\not\text{desc}}([t, p])$, $p_{\not\text{desc}}([f, t])$, and $p_{\not\text{desc}}([p, t])$, and is the only hypothesis that satisfies P. The hypothesis $[t, f, f]$, for example, is a descendant of $[t, f]$ and, therefore, does not the satisfy third property in P. The hypothesis $[p]$, on the other hand, is not a descendant of $[t]$ and, therefore, does not satisfy the first property of P.

5.2 Symbolic Test

We reformulate the test presented in Definitions 2 and 4 using properties rather than hypotheses. Let H be a set of hypotheses and let P be a set of properties such that $\text{hypos}(P) = H$. The test of H can be written:

$$\exists \sigma \in M : obs(\sigma) = o \ \wedge \ P \subseteq \text{props}(\text{hypo}(\sigma)).$$

Assuming the properties can be enforced on the behaviors themselves (as we illustrate in Section 6), the hypotheses are not explicitely manipulated. The test solver will use the set of properties P to guide the search for a behavior σ consistent with the observation, or to disregard (possibly partially defined) behaviors. If the test fails, the test solver should be able to trace back all the properties it used in order to reach its conclusion. These properties defines a set of hypotheses that contains no candidate; this defines our conflict.

Formally, the test solver is given a set P of properties representing the hypotheses $H = \text{hypos}(P)$, returns *true* if $\text{hypos}(P) \cap \Delta \neq \emptyset$, and a conflict $\mathsf{C} \subseteq P$ such that $\text{hypos}(\mathsf{C}) \cap \Delta = \emptyset$ otherwise. Here, the conflict is a subset of properties and the set $\text{hypos}(\mathsf{C})$ of hypotheses excluded by C are those that satisfy all properties in C. In the worst case – if the solver is not able to extract a more useful cause for failure – it can always return $\mathsf{C} = P$.

The simplest symbolic representation of h defines h as the (single) hypothesis that is a descendant of h and of none of its children. We denote $P_0(h)$ this representation:

$$P_0(h) = p_{\text{desc}}(h) \cup \{p_{\not\text{desc}}(h') \mid h' \in \text{chi}(h)\}.$$

This is correct, i.e., $\text{hypos}(P_0(h)) = \{h\}$, and finite because the hypothesis space is a well partial order.

The choice of which properties to include in the descriptions of hypotheses has an influence on the generality of the conflict generated. Assume for instance we want to find a set of properties P to represent the set $\{[p]\}$ (cf. Figure 2). $P_0([p])$ is the set $\{p_{\text{desc}}([p]),\ p_{\not\text{desc}}([f, p]),\ p_{\not\text{desc}}([p, f]),\ p_{\not\text{desc}}([p, p]),\ p_{\not\text{desc}}([p, t]),\ p_{\not\text{desc}}([t, p])\}$. However, the two properties $p_1 = p_{\not\text{desc}}([f, p])$ and $p_2 = p_{\not\text{desc}}([p, f])$ may be replaced by the single property $p_3 = p_{\not\text{desc}}([f])$. (Observe that a third option is to include all three properties in P.) The second choice looks more appealing that the first because p_3 is more general than p_1 and p_2; in this example, it tells the test solver directly that f cannot occur in the trace. On the other hand, using only p_3 instead of p_1 and p_2 it is not possible to generate a conflict such that $[p, f]$ is in the set of successors whilst $[f, p]$ is not. Thus, no option dominates the others. In the experiments reported in Section 7, we used $P_0(h)$ as defined above.

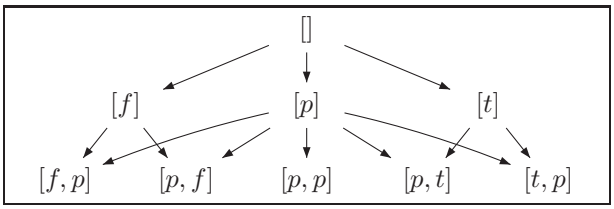

Figure 2: Part of the hypothesis space ($[p]$, its parent, its children, its siblings). The arrows show the ordering.

5.3 Computing Successors

The last step to implement is the computation of successors. Remember that the set of successors is defined as the mininal elements of the set $S' = \text{desc}(h) \setminus \text{hypos}(\mathsf{C})$. The set S' can be infinite, and therefore cannot be computed explicitly. Instead, we show (Theorem 4) how to compute the successors using only a finite number of $\otimes$ operations defined at the end of Subsection 4.1. We first illustrate the process with an example and then prove it works in the general case.

Assume that we have tested the hypothesis $h = [p]$, and that the test solver has returned the following conflict:

$$\{p_{\text{desc}}([p]),\ p_{\not\text{desc}}([p, p]),\ p_{\not\text{desc}}([f])\}.$$

Note that the conflict rejects the hypothesis $h = [p]$. An interpretation of this conflict is "if p occurred at least once, then p occurred at least twice or f occurred at least once". Therefore, the successors of h are $[p, p]$, $[f, p]$ and $[p, f]$. As mentioned above, a hypothesis is excluded by the conflict if it satifies all three properties. Therefore, an element of S' is an element that contradicts at least one of these properties. Clearly, all descendants of h satisfy the first property, so no element of S' can contradict this property. Therefore, all properties of the form $p_{\text{desc}}(h')$ can be ignored when computing successors. We now turn to the last two properties of the conflict. An element of S' is a descendant of h that contradicts one of those two properties. Therefore, in addition to being a descendant of h, an element of S' must be a descendant of $[p, p]$ or a descendant of $[f]$. Such elements include $[f, p]$, $[p, f]$, $[p, p]$, $[f, p, p]$, $[f, p, t]$, etc. It is easy to see that the minimal elements of S', i.e., the set of successors, is $\{[p, p], [p, f], [f, p]\}$.

We now generalise this result. Given a set of properties $P \subseteq \mathbb{P}$, we write $A(P) = \{h \in \mathbb{H} \mid p_{\text{desc}}(h) \in P\}$ and $\Omega(P) = \{h \in \mathbb{H} \mid p_{\not\text{desc}}(h) \in P\}$ (observe that both sets are finite since P is finite).

Theorem 4 *Let h be a hypothesis and let C be any conflict*

that rejects h, then

$$\min_{\preceq}(\mathrm{desc}(h) \setminus \mathrm{hypos}(\mathsf{C})) = \min_{\preceq}(h \otimes \Omega(\mathsf{C}))\,.$$

Proof Sketch for Theorem 4: Notice that $min_{\preceq}(A) = \min_{\preceq}(B)$ is equivalent to $\mathrm{desc}(A) = \mathrm{desc}(B)$, which is equivalent to $(A \subseteq \mathrm{desc}(B)) \wedge (B \subseteq \mathrm{desc}(A))$, which is what we prove now for $A = \mathrm{desc}(h) \setminus \mathrm{hypos}(\mathsf{C})$ and $B = h \otimes \Omega(\mathsf{C})$.

For the first conjunct, let h_s be an element of $\mathrm{desc}(h) \setminus \mathrm{hypos}(\mathsf{C})$. Then $h_s \in \mathrm{desc}(h)$. Remember that $\mathrm{hypos}(\mathsf{C}) = \bigcap_{p \in \mathsf{C}} \mathrm{hypos}(p)$. Thus, since $h_s \notin \mathrm{hypos}(\mathsf{C})$, $h_s \notin \mathrm{hypos}(p_s)$ for some $p_s \in \mathsf{C}$. If $p_s = p_{\mathrm{desc}}(h')$ for some $h' \in A(\mathsf{C})$, we have $h' \not\preceq h_s$ (since $h_s \notin \mathrm{hypos}(p_{\mathrm{desc}}(h'))$). Because all properties in the conflict C, including $p_{\mathrm{desc}}(h')$, are satisfied by h, we also have $h' \preceq h$, and hence $h' \preceq h \preceq h_s$. This contradicts $h' \not\preceq h_s$. Therefore p_s must be a property of the form $p_{\not\mathrm{desc}}(h')$ for some $h' \in \Omega(\mathsf{C})$. Since $h_s \notin \mathrm{hypos}(p_{\not\mathrm{desc}}(h'))$, we have $h' \preceq h_s$, and hence $h_s \in \mathrm{desc}(h \otimes h')$.

For the second conjunct, let h_s be an element of $h \otimes \Omega(\mathsf{C})$. Then, by definition of $\otimes$, there exists $h' \in \Omega(\mathsf{C})$ such that $h' \preceq h_s$ and $h \preceq h_s$. Therefore h_s does not satisfy property $p_{\not\mathrm{desc}}(h') \in \mathsf{C}$, hence $h_s \notin \mathrm{hypos}(\mathsf{C})$ and $h_s \in \mathrm{desc}(h) \setminus \mathrm{hypos}(\mathsf{C})$. □

6 Implementation

In this section, we briefly describe our Boolean satisfiability (SAT) implementation of the test solver in the case of DES. A test P amounts to determining if the system model allows for a sequence σ of events that generates the observation and satisfies all properties in P. We generate a propositional logic formula Φ that encodes all paths σ consistent with the model, the observation, and the given set of properties, then query a SAT solver to find satisfying assignment for Φ. If Φ is not satisfiable, the test fails and we use the clause learning capabilities of the SAT solver to generate a conflict.

How to enforce that the path is consistent with the model and the observation is well-known (Grastien et al. 2007). Of importance here is that we assume that σ has a maximum length of n events: n can be upperbounded by the product of the number of distinct observation time points (which is known from the input) and the maximum number of unobservable events that can take place between any two observations (which is a property of the model). For the system model we conducted experiments on, the latter number is 6.

For each event $e \in \Sigma$ and for each timestep $t \in \{1, \ldots, n\}$, a propositional variable $e@t$ that is true iff the t^{th} event in σ is e. The encoding of the model and the observation is a formula over the $e@t$ variables such that the set of satisfying assignments corresponds to the set of event/state sequences allowed by the system model and consistent with the observation. Similar encodings have been used for bounded planning (Kautz and Selman 1992) and model checking (Biere et al. 2003).

Each call to the test solver (in lines 7 and 10 of Algorithm 1) is made with a set of hypotheses, which also constrain the set of paths. As explained in the previous section, the set of hypotheses to be tested is represented by a finite set of properties, P, each of the form $p_{\mathrm{desc}}(h)$ or $p_{\not\mathrm{desc}}(h)$. To represent properties of relevant hypotheses in SAT, we create a propositional variable $\mathrm{desc}(h,t)$ for each h that is equal to, or a prefix of, some h' such that $p_{\mathrm{desc}}(h')$ or $p_{\not\mathrm{desc}}(h')$ is in P, and for each timestep $t \in \{1, \ldots, n\}$. In any satisfying assignment, $\mathrm{desc}(h,t)$ will be true iff the prefix σ_t (the first t events of sequence σ) is a descendant of h. In particular, $h \preceq \mathrm{hypo}(\sigma)$ iff $\mathrm{desc}(h,n)$ evaluates to true. The values of variables $\mathrm{desc}(h,t)$ is recursively constrained via the following formulae:

$$\mathrm{desc}(h.e,t) \leftrightarrow \mathrm{desc}(h.e,t-1) \vee (\mathrm{desc}(h,t-1) \wedge e@t)\,.$$

were $h.e$ appends event e to hypothesis h. Observe how these formulae further constrain the truth values of $e@t$. The base cases are that: (i) $\mathrm{desc}([],t)$ is always true; and (ii) $\mathrm{desc}(h,0)$ is false if $h \neq []$. The formula Φ that is given to the SAT solver is the conjunction of all the formulae above.

If Φ is unsatisfiable, we use the clause-learning capabilities of the SAT solver to generate a conflict: the variables representing the set of properties to be tested (P) are assigned at the beginning of the SAT search. If these assignments makes the problem unsatisfiable, the clause learning module returns a subset of those assignments that is sufficient to prove unsatisfiability, and that is the conflict.

7 Experiments

Input data for our case study is an alarm log from the operations center of TransGrid, the company that owns and operates the electricity transmission network in NSW and the ACT, Australia. The log contains alarms generated by automatic equipment (e.g., switch gear, voltage and power sensors and regulators) located throughout the transmission network, as well as commands issued by operators. It covers roughly fifteen hours: the first two thirds are routine operation, then a major fault situation arises and the rest of the log chronicles the operators' efforts to reconfigure the network to restore service. Restricted to relevant alarm types (i.e., those that appear in our model) the log has 731 entries.

The purpose of diagnosis is to provide operators with a comprehensible picture of the situation in the network. Therefore, events to be diagnosed include not only real faults (like transient or permanent line faults) but also events that, while in themselves not faulty, cannot be explained by the model (e.g., a sudden but non-critical voltage drop, if there is no discernable reason for it). The diagnostic summary of the network situation must be available in real-time. Therefore, we split the alarm log into (a) fixed size (one minute) time windows, and (b) variable sized chuncks separated by at least one minute where no alarm occurs. This gives us 129 problem instances. For each of those we extract a submodel, with only the parts of the network potentially relevant to reasoning about this set of alarms. This subnetwork is computed iteratively, starting from the components that emitted the alarms in the instance alarm log, adding components that may influence (according to the model) this already in the subnetwork, until a fixpoint is reached.

The behavior of the (sub-)network is captured by a set of component automata, similarly to the example in Section 2. The number of components varies from 3 to 105, with component models each ranging from 8 to 1 024 states and 44 to 92 800 transitions, which means that, in general, a single

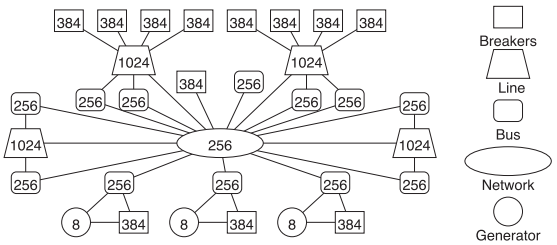

Figure 3: Subnetwork for problem window-347.

automaton model cannot be computed. Figure 3 shows the topology of the subnetwork for problem window-347, where each component type is drawn using a different node shape, and labeled with its number of states.

Centralised methods such as the unfolding approach (Zanella and Lamperti 2003) or the diagnoser approach (Sampath et al. 1995) require respectively exponential and double-exponential memory size in the number of components, which makes them inapplicable here. Instead, we compare the performance of our approach with a state-of-the-art method for DES diagnosis based on the distributed computation of all system behaviors consistent with the observation (Kan John and Grastien 2008). A junction tree is computed over the network topology, and a diagnosis is computed locally for each cluster of the junction tree; the local diagnoses are made consistent with each other until a fixed-point is reached (Su and Wonham 2005). We compare the preferred-first algorithm powered by a SAT tester (later called PF) against this junction tree algorithm (later called JT) with a time limit of 10 minutes. JT only computes the set of behaviors and is not required to extract the diagnosis. For the model at hand, the number of time points necessary for PF (as explained in the previous section) is 6 times the number of observation points, which ranges from 1 to 139.

Out of 129 problems, PF is able to solve 116 problems, while only 35 problems can be solved by JT. A sample of the results are presented in Table 1. For all problems that both approaches can solve, the runtimes are similar, although PF is sligthly faster. However, PF is able to solve most problems while JT very often runs out of time. Only for two problems does JT perform better than PF.

The only problems JT can solve do not involve transmission lines. In our model, a line state is represented by four binary variables and three four-valued variables (i.e., 1 024 states). Furthermore, there are 64 initial states, and the line is not directly observable. JT requires to project automata which necessitates to perform determinisation and minimisation. These operations turn out to be too expensive for JT to be effective. This is a nice feature of PF coupled with SAT: the solver does not have to compute behaviors starting from the initial state; therefore it is not very sensitive to imprecision wrt initial states.

The experiments also illustrate that PF is sensitive not only to the number of components in the diagnosis problem but also to the number of diagnosis candidates. We unsuccessfully tried to solve more problems by giving more time to the solver; at this stage, it seems that the number of candidates is simply too large and that enumerating all those candidates is impossible (this is similar to diagnosis of circuits where, apart from trivial problems, only minimal-cardinality diagnosis candidates are extracted).

	N	M	C	A	PF	JT
window-250	1	0	2	3	0.3	1.5
window-137	1	0	2	5	0.5	1.8
chunk-008	1	0	4	5	0.7	2.7
chunk-004	1	2	3	3	0.8	2
window-249	2	1	5	4	1	2.3
chunk-073	2	1	5	6	1.4	2.1
window-248	3	2	5	5	1.6	2.2
chunk-056	1	4	4	7	1.7	2.6
window-618	1	0	6	2	0.7	–time–
window-602	1	1	5	2	0.9	–time–
window-135	1	0	9	5	1	–time–
window-601	1	2	5	3	1.2	–time–
window-136	1	1	9	4	1.3	–time–
window-617	1	1	13	3	1.4	–time–
chunk-040	2	3	5	4	1.6	–time–
window-157	2	2	11	3	1.6	–time–
chunk-054	1	0	9	18	1.9	–time–
window-155	2	2	12	5	2.2	–time–
chunk-105	2	2	8	4	2.7	–time–
window-527	2	1	11	8	2.7	–time–
chunk-127	2	4	6	5	3.4	–time–
window-134	2	2	17	6	4	–time–
window-526	3	2	12	9	4.2	–time–
chunk-132	1	3	20	6	5.1	–time–
chunk-077	1	2	12	11	5.2	–time–
chunk-037	2	4	9	12	6	–time–
window-525	4	3	12	10	6.2	–time–
chunk-026	3	2	13	11	6.8	–time–
window-524	5	4	12	11	8.5	–time–
chunk-049	4	4	16	11	11.9	–time–
chunk-045	1	2	20	19	13	–time–
window-348	2	8	25	11	30.7	–time–
window-258	25	5	11	6	51	–time–
window-257	26	5	12	7	54.1	–time–
window-256	27	7	12	8	95.5	–time–
chunk-076	27	7	12	8	100.2	–time–
window-347	4	9	32	13	106.1	–time–
chunk-078	27	8	13	14	206.4	–time–
window-336	?	?	58	49	–time–	–time–
window-332	?	?	67	57	–time–	–time–
window-335	?	?	67	66	–time–	–time–
window-333	?	?	71	71	–time–	–time–
window-334	?	?	71	71	–time–	–time–
chunk-089	?	?	105	146	–time–	–memory–
window-410	?	?	19	13	–time–	5
window-409	?	?	22	14	–time–	5.3
Nb problems solved (/129)					116	35

Table 1: Results: N: number of minimal candidates, M: maximum number of faults in a minimal candidate, C: number of components in the problem, A: number of alarms, PF: runtime for PF running SAT, and JT: runtime for automata-based approach (in seconds; –time– means time out).

8 Related Work

We have already discussed in detail the relationship between our approach and the classical conflict-based theory

for static systems (Reiter 1987) on the one hand, and the main classes of approaches for diagnosing DES (Zanella and Lamperti 2003; Sampath et al. 1995) on the other. Here, we focus on the relationship between our work and techniques targeted at solving the simpler problem of finding a single diagnosis candidate, and on the dual role of symbolic representations in our framework.

8.1 Diagnosis of DES and Path-Finding Problems

This paper and those mentioned just above consider *exhaustive* diagnosis: the problem is to find *all* the minimal hypotheses consistent with the observation. This contrasts with the problem of finding a *single* diagnosis candidate or testing a single diagnosis hypothesis, which is related to the test operation in our framework.

For instance, McIlraith's (1998) notion of "potential explanatory diagnosis" essentially implements our test operation, but offers no guidance on the vital question of how to find in the very large, or even infinite, space of hypotheses all minimal explanations. We answer that question, by providing an algorithm for the systematic exploration of the hypothesis space, guided by the preference order, which is also able to take advantage of additional information provided by the test solver in the form of conflicts.

It has long been acknowledged that, in the case of discrete-event systems, finding a single diagnosis candidate amounts to finding an execution path of the system model (a sequence of events) which satisfies certain constraints (representing the observation), and that this task can be solved using planning and other path finding techniques (Cordier and Thiébaux 1994; McIlraith 1994). Compilations into planning (Haslum and Grastien 2011; Sohrabi, Baier, and McIlraith 2010) or SAT (Grastien et al. 2007) have indeed been used to find a single system trace with a minimal number of faults. Note that planning technology can also be used to implement the test operation in our algorithm. We have tried this, but in our experiments on the transmission network problems, the SAT test solver proved to be better in practice.

8.2 Symbolic Approaches in Diagnosis of DES

We use symbolic representations for two distinct purposes in the implementation of the diagnosis algorithm.

First, for each diagnosis test, we symbolically represent the set of hypotheses tested, via a set of "properties". The reasons for this are that i) this allows the test solver to generalise test failures, returning a subset of properties that make the test fail; and ii) when testing essentiality (line 7 in Algorithm 1), the set of hypotheses is infinite and must therefore be manipulated implicitly. There is no other approach in the literature on diagnosis of DES that uses symbolic representations in this way.

Second, we also use a symbolic representation (SAT) in the implementation of the test solver. Here, the purpose of symbolic techniques is to avoid the explicit manipulation of sets of system states, using partial state variable assignments to represent them. This use of symbolic representations is not a novelty of this paper. The use of CNFs and compiled symbolic representations such as BDDs is well-established for DES, both for computing a single candidate (Grastien et al. 2007) and an exhaustive diagnosis (Schumann, Pencolé, and Thiébaux 2007). For instance, Schumann et al. (2007) use BDDs to represent belief states and diagnosis information, and update them by means of BDD operations as observations are received. This approach, however, still computes a representation of all system behaviors consistent with the observation, which is what we avoid using hypothesis tests.

9 Conclusion and Future Work

In this paper, we presented a radically new approach which makes three important contributions to the state of the art. Firstly, it unifies two subareas of diagnosis, namely consistency-based diagnosis for static systems and diagnosis of DES, which had traditionally followed separate routes with little interaction. Secondly, not only does it generalise Reiter's work to DES, but it also opens the way to considering systems with infinite state spaces such as timed and hybrid systems. Finally it transfers the efficiency of conflict-based approaches to DES diagnosis, resulting in much more practical methods.

The generality of our framework is appealing: its only requirement is the ability of testing whether a set of hypotheses (taken from a well-ordered space where hypotheses have finite depth) contains a candidate. Moreover, the required tests can be performed symbolically using as few as two basic types of properties. As we demonstrated in the DES case, this approach also has significant potential to scale to large complex problems which are clearly out of reach of conventional diagnosis methods.

Our work is currently progressing in two different directions. The first is the use of conflicts in diagnosis explanation and model debugging (Belard, Pencolé, and Combacau 2011). Conflicts provide a justification for the diagnosis, a type of information that has never been exploited for DES. We believe that the justifications generated by our approach can be improved by incorporating observation or model information, to obtain conflicts such as "because the recloser follows a single reclosing procedure, and because the alarm `detected` has been raised, a fault of type p or of type t must have occurred".

The second direction is the improvement of the applicability of our results, in terms of efficiency and scope. Our current SAT implementation is still rudimentary and could be improved by several orders of magnitude, e.g., with decentralised pre-processing (Pencolé and Cordier 2005) or with dedicated SAT solvers (Rintanen 2010). Looking at more expressive classes of systems, we are particularly interested in hybrid systems and timed systems for which the set of behaviors consistent with an observation can be potentially infinite. We plan to investigate implementations of the corresponding test solver using SAT Modulo Theories (de Moura, Dutertre, and Shankar 2007).

Acknowledgments

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.

References

Aghasaryan, A.; Fabre, É.; Benveniste, A.; Boubour, R.; and Jard, C. 1998. Fault detection and diagnosis in distributed systems: an approach by partially stochastic Petri Nets. *Journal of Discrete Event Dynamical Systems* 8(2):203–231.

Belard, N.; Pencolé, Y.; and Combacau, M. 2011. MEDITO: A logic-based meta-diagnosis tool. In *22nd International Workshop on Principles of Diagnosis (DX-11)*, 130–137.

Biere, A.; Cimatti, A.; Clarke, E.; Strichman, O.; and Zhu, Y. 2003. Bounded model checking. *Advances in Computers* 58:118–149.

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

Cordier, M.-O., and Thiébaux, S. 1994. Event-based diagnosis for evolutive systems. In *5th International Workshop on Principles of Diagnosis (DX-94)*, 64–69.

de Moura, L. M.; Dutertre, B.; and Shankar, N. 2007. A tutorial on satisfiability modulo theories. In *19th International Conference on Computer Aided Verification (CAV)*, 20–36.

Grastien, A.; Anbulagan; Rintanen, J.; and Kelareva, E. 2007. Diagnosis of discrete-event systems using satisfiability algorithms. In *22nd Conference on Artificial Intelligence (AAAI-07)*, 305–310.

Haslum, P., and Grastien, A. 2011. Diagnosis as planning: two case studies. In *5th Scheduling and Planning Applications Workshop (SPARK-11)*.

Kan John, P., and Grastien, A. 2008. Local consistency and junction tree for diagnosis of discrete-event systems. In *18th European Conference on Artificial Intelligence (ECAI-08)*, 209–213.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In *10th European Conference on Artificial Intelligence (ECAI'92)*, 359–363.

McIlraith, S. 1994. Towards a theory of diagnosis, testing and repair. In *5th International Workshop on Principles of Diagnosis (DX'94)*, 185–192.

McIlraith, S. 1998. Explanatory diagnosis: Conjecturing actions to explain observations. In *6th International Conference on Principles of Knowledge Representation and Reasoning (KR-98), Trento, Italy, June 1998*, 167–177.

Nash-Williams, C. 1963. On well-quasi-ordering finite trees. *Mathematical Proceedings of The Cambridge Philosophical Society* 59(4):833–835.

Pencolé, Y., and Cordier, M.-O. 2005. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence (AIJ)* 164(1–2):121–170.

Pulido, B., and Alonso González, C. 2004. Possible conflicts: a compilation technique for consistency-based diagnosis. *IEEE Transactions on Systems, Man, and Cybernetics (TSMC)* 34(5):2192–2206.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence (AIJ)* 32(1):57–95.

Rintanen, J. 2010. Heuristic planning with SAT: beyond strict depth-first search. In *23rd Australasian Joint Conference on Artificial Intelligence (AI-10)*, 415–424.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1995. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control (TAC)* 40(9):1555–1575.

Schumann, A.; Pencolé, Y.; and Thiébaux, S. 2007. A spectrum of symbolic on-line diagnosis approaches. In *22nd Conference on Artificial Intelligence (AAAI-07)*, 335–340.

Sohrabi, S.; Baier, J.; and McIlraith, S. 2010. Diagnosis as planning revisited. In *12th International Conference on the Principles of Knowledge Representation and Reasoning (KR-10)*, 26–36.

Su, R., and Wonham, W. 2005. Global and local consistencies in distributed fault diagnosis for discrete-event systems. *IEEE Transactions on Automatic Control (TAC)* 50(12):1923–1935.

Zanella, M., and Lamperti, G. 2003. *Diagnosis of active systems*. Kluwer Academic Publishers.