Diagnosis of Discrete-Event Systems using Binary Decision Diagrams

Anika Schumann[1], **Yannick Pencolé**[1] and **Sylvie Thiébaux**[1,2]

Abstract. We improve the efficiency of Sampath's diagnoser approach by exploiting compact symbolic representations of the system and diagnoser in terms of binary decision diagrams. We present an algorithm for synthesising the symbolic diagnoser with promising results on test cases derived from a telecommunication application.

1 INTRODUCTION

Approaches to the diagnosis of discrete-event systems usually suffer from poor on-line performance or space explosion. At one extremity of the spectrum lie on-line simulation-based approaches such as [1]. Given a decentralised model of the system (a model of the individual components and their interactions as communicating automata), these approaches track the possible system behaviors as observations become available. The reliance on a decentralised model makes them space efficient, but the set of possible behaviors is so large that on-line computation is time inefficient. At the other extremity lie diagnoser-based approaches [12]. They compile, off-line, a centralised system model into another finite state machine (the diagnoser) which efficiently maps observations to possible failures. Here the space required by the centralised model, let alone that required by the diagnoser, constitutes a major problem.

Clearly, approaches that offer a more appropriate space/time tradeoff are needed. One current research direction focuses on extending the diagnoser approach to handle decentralised models and produce local subsystem diagnosers whose results are merged at run-time [10]. The present paper follows a different path: we remain within the classical diagnoser approach [12], but use symbolic representations based on binary decision diagrams (BDDs) [3] to improve its space efficiency and scale up to larger systems. BDDs are compact representations of boolean functions which enable the encoding and implicit manipulation of sets of states and transitions, without the need for explicit enumeration. In a range of areas, such as verification, controler synthesis, or AI planning, BDD-based representations have given rise to algorithms capable of exploiting the structure of the system, resulting in significant space and time gains. Experiments with our approach on test cases derived from a telecommunication network application yield similarly promising results.

The idea of exploiting symbolic representations in the context of discrete-event systems diagnosis is not new [4, 8, 15], but has traditionally been applied to different problems, e.g. checking diagnosability or off-line diagnosis, using off-the-shelf model-checkers. An exception is [9], which recasts a form of diagnoser synthesis in terms of polynomial equations. In contrast to that work, we focus on the BDD-level encoding of the system and diagnoser, and describe, directly in terms of BDD operations, fully implemented algorithms for synthesizing the symbolic diagnoser.

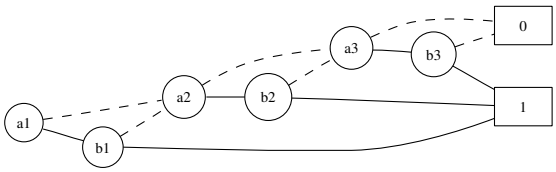

Figure 1. BDD for $(a_1 \wedge b_1) \vee (a_2 \wedge b_2) \vee (a_3 \wedge b_3)$. Dash = low edges.

2 BINARY DECISION DIAGRAMS

We start with a brief introduction to BDDs. An ordered binary decision diagram (OBDD, or BDD for short), is a rooted directed acyclic graph representation of a boolean function $\mathcal{B}^n \mapsto \mathcal{B}$. As can be seen in Figure 1, a BDD has one or two leaf nodes marked with 0 or 1, and a set of internal nodes each marked with one of the function's variables. Internal nodes have two outgoing edges: *high* and *low* edge. Given a truth assignment of the variables, the value of the function is determined by traversing the BDD top down and following the high edge if the variable marking the current node is true, and the low edge otherwise. The function's value is *true* if the leaf marked 1 is reached, and *false* otherwise.

OBDDs obey the constraint that every path in the graph respects a linear ordering of the variables. A BDD is also reduced such that (1) no two distinct nodes are marked with the same variable and have identical low and high successors, and (2) no node has identical low and high successors. These reductions make the BDD a canonical representation of the function given the chosen variable ordering, hence checking equality is a constant time operation. In fact, any boolean operation $f \star g$ on two BDDs f and g can be carried out in $\mathcal{O}(|f||g|)$ at most. Furthermore, while the BDD representation still requires exponential space in n in the worst case, the reductions often make the BDD of a function exhibiting some structure much smaller than its disjunctive normal form (DNF). The size of a BDD is sensitive to the chosen variable ordering. Finding an optimal ordering is a co-NP-complete problem [3], but fortunately, standard BDD packages feature good heuristics [14].

BDDs are useful in compactly representing finite state machines. The set of events $\Sigma = \{\sigma_1, \ldots, \sigma_p\}$ labelling the transitions can be encoded with $\lceil \log_2 p \rceil$ boolean variables $b_1^\Sigma \ldots b_{\lceil \log_2 p \rceil}^\Sigma$. Similarly, the set of states $X = \{x_1, \ldots, x_m\}$ can be encoded with $\lceil \log_2 m \rceil$

[1] Computer Sciences Laboratory, The Australian National University, Canberra, ACT 0200, Australia. email: {anika, pencole, thiebaux}@csl.anu.edu.au.
[2] Knowledge Representation & Reasoning Program, National ICT Australia.
NICTA is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

boolean variables $b_1^X \ldots b_{\lceil \log_2 m \rceil}^X$; the initial state of the automaton is then simply given by a boolean function (represented by a BDD) over these state variables. For instance, in a 6 state automaton, the state x_2 would be given by the conjunction $\neg b_3^X \wedge b_2^X \wedge \neg b_1^X$, and the set of states $\{x_2, x_5\}$ by the DNF $\neg b_3^X \wedge b_2^X \wedge \neg b_1^X \vee b_3^X \wedge \neg b_2^X \wedge b_1^X$. Transitions require the introduction of another set of $\lceil \log_2 m \rceil$ boolean variables $b_1^{X'} \ldots b_{\lceil \log_2 m \rceil}^{X'}$, called the primed variables, which are used to represent the target states of the transitions. Each transition can then be given as a conjunction involving the state variables, event variables, and primed variables. For instance, in an automaton of 6 states and 3 events, the transition $x_2 \xrightarrow{\sigma_1} x_5$ would be given by $\neg b_3^X \wedge b_2^X \wedge \neg b_1^X \wedge \neg b_2^\Sigma \wedge b_1^\Sigma \wedge b_3^{X'} \wedge \neg b_2^{X'} \wedge b_1^{X'}$. The transition relation, i.e, a set of transitions, can then be given e.g., as a DNF which the BDD data structure will hopefully greatly reduce.

3 OVERVIEW OF THE APPROACH

Our aim is to exploit BDD representations of finite state machines to improve the efficiency of Sampath's diagnoser approach [12], and in particular its space requirements. Starting from a symbolic system representation in terms of BDDs, we want to compute a symbolic diagnoser using BDD operations.

Sampath's framework starts with a set of individual component models. These are composed to obtain a centralised global system model from which the diagnoser is computed. Here, to simplify the computation of our symbolic diagnoser and its presentation, we find it useful to add an intermediate step to the process: instead of computing the diagnoser directly from the global model, we first abstract the global model from irrelevant features such as unobservable non-failure events and order of failures.

The three next sections detail the three steps of our approach: computation of the global model by composition of the individual component models, abstraction of the global model, and computation of the diagnoser from the abstracted model. The various steps are formalised into logic, suitable for the derivation of BDD algorithms. For each step, we describe the mapping between the logical specification and the algorithm. This description constitutes an informal proof of correctness.

4 COMPONENT AND GLOBAL MODELS

Let $G_i = \langle X_i, \Sigma_o, \Sigma_u, \Sigma_f, x_{0_i}, T_i \rangle$, $i = 1 \ldots n$, be n individual component models characterised by their set of states $X_i = \{x_{1_i}, \ldots, x_{m_i}\}$, their set of events $\Sigma = \{\sigma_1, \ldots, \sigma_p\}$ which is partitioned into observable Σ_o and unobservable Σ_u events, their set of failure events $\Sigma_f \subseteq \Sigma_u$, their initial state x_{0_i}, and their transition relation $T_i \subseteq X_i \times \Sigma \times X_i$. For simplicity of the presentation, and without loss of generality, we assume that all components share the same event set Σ. The report [13] shows how to handle events that are local to components and take advantage of them to speed up computations.

The global model $G = \langle X, \Sigma_o, \Sigma_u, \Sigma_f, x_0, T \rangle$ is defined as the synchronous composition of the component models: $X = \Pi_{i=1}^n X_i$, $x_0 = \Pi_{i=1}^n x_{0_i}$, $T = \{(x_1, \ldots, x_n) \xrightarrow{\sigma} (x'_1, \ldots, x'_n) \mid \forall i \in 1 \ldots n, x_i \xrightarrow{\sigma} x'_i \in T_i\}$. The composition ensures that a transition is possible in a given global state $(x_1, \ldots, x_n)$ iff it is possible in the respective individual states of all components.

Now, symbolically, the individual components are encoded as[3] $G_i = \langle b_i^X, b_i^{X'}, b^\Sigma, X_i, \Sigma_o, \Sigma_u, \Sigma_f, x_{0_i}, T_i \rangle$,

[3] We give identical names to sets and to the corresponding boolean functions, e.g. Σ_o, X, etc. This should not cause confusion.

Algorithm 1 $ComputeGlobal(G_i, i \leftarrow 1 \ldots n)$

1: INPUT: symbolic component models G_i
2: $x_0 \leftarrow \wedge_{i=1}^n x_{0_i}$
3: $X \leftarrow x_0$
4: $newX \leftarrow x_0$
5: $T \leftarrow false$
6: **while** IsDef($newX$) **do**
7: $\quad newT \leftarrow newX \wedge \wedge_{i=1}^n T_i$
8: $\quad T \leftarrow T \vee newT$
9: $\quad newX \leftarrow$Swap(Extract($newT$, $b^{X'}$), b^X, $b^{X'}$)
10: $\quad newX \leftarrow newX \wedge \neg X$
11: $\quad X \leftarrow X \vee newX$
12: **end while**
13: OUTPUT: symbolic reachable global states X and transitions T

IsDef(bdd) returns true iff bdd does not represent *false*. Extract(bdd, B) deletes from bdd all occurrences of variables not in B. Swap(bdd, $\{a_1, \ldots, a_k\}, \{b_1, \ldots, b_k\}$) renames, in bdd, variable a_i with b_i, $i = 1 \ldots k$, and vice versa.

where $b_i^X = \{b_{1_i}^x, \ldots, b_{\lceil \log_2 m_i \rceil_i}^x\}$ are the state variables, $b_i^{X'} = \{b_{1_i}^{x'}, \ldots, b_{\lceil \log_2 m_i \rceil_i}^{x'}\}$ are the primed variables, $b^\Sigma = \{b_1^\sigma, \ldots, b_{\lceil \log_2 p \rceil}^\sigma\}$ are the event variables, X_i is the boolean function over b_i^X characterising the set of states, Σ_o, Σ_f, and Σ_u are the boolean functions over b^Σ characterising the set of observable, unobservable, and failure events, x_{0_i} is the boolean function over b_i^X characterising the initial state, and T_i is the boolean function over $b_i^X \cup b^\Sigma \cup b_i^{X'}$ characterising the transition relation. The symbolic global model $G = \langle b^X, b^{X'}, b^\Sigma, X, \Sigma_o, \Sigma_u, \Sigma_f, x_0, T \rangle$ is easy to define: $b^X = \cup_{i=1}^n b_i^X$, $b^{X'} = \cup_{i=1}^n b_i^{X'}$, $X = \wedge_{i=1}^n X_i$, $x_0 = \wedge_{i=1}^n x_{0_i}$, and $T = \wedge_{i=1}^n T_i$. The latter conjunction implements strong synchronisation by guaranteeing that the event variables, which are shared across components, have consistent values.

In practice, it is very useful to restrict X (resp. T) to the states (resp. transitions) that are reachable from x_0, as these form a small subset of the above. Algorithm 1 computes such an X and T. X and the set $newX$ of newly reached states are first initialised to x_0 (lines 2-4). Then until a fixed point is reached (line 6), the set of transitions $newT$ that can be fired from the newly reachable states are computed and added to T (lines 7-8). The set of newly reachable states is computed by extracting the set of target states from those transitions. The variables used to encode those states need to be renamed (unprimed), before the new states are added to X (lines 9-11).

5 ABSTRACTED MODEL

In order to speed up the computation of the diagnoser, we first abstract the global model in the following way. We note that only observable events and possible sets of failures that have occurred before an observable event have an impact on the diagnoser. In particular, the construction of the diagnoser does not depend on the kind of unobservable events which are not failures, on the order of successive failures, and on the global states encountered within a sequence of unobservable events. Therefore, our abstracted global model consists only of those states that are the origin or target of an observable transition (and of the initial state). It has two types of transitions: (1) the observable transitions of the global model, and (2) failure transitions, each of which is labelled with a *set* of failure events that has occurred

on some path from the origin state of the transition to its target state. Figure 2 gives an example.

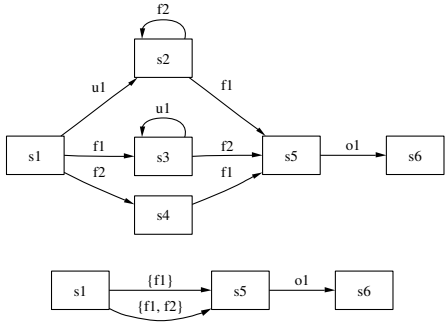

Figure 2. Global model (top) and its abstraction (bottom). $\Sigma_o = \{o_1\}, \Sigma_u = \{u_1\}, \Sigma_f = \{f_1, f_2\}$

Formally, let $G = \langle X, \Sigma_o, \Sigma_u, \Sigma_f, x_0, T\rangle$ be the global model. The abstracted model is $\widetilde{G} = \langle \widetilde{X}, \widetilde{\Sigma}_o, \widetilde{F}, \widetilde{x}_0, \widetilde{T}_o, \widetilde{T}_F\rangle$, where the set of abstracted states is $\widetilde{X} = \{x_0\} \cup \{x \in X \mid \exists \sigma \in \Sigma_o, \exists x' \in X \text{ s.t. } x \xrightarrow{\sigma} x' \in T \text{ or } x' \xrightarrow{\sigma} x \in T\}$, the set of observable transition labels is $\widetilde{\Sigma}_o = \Sigma_o$, the set of failure transition labels is $\widetilde{F} = 2^{\Sigma_f}$, the initial state is $\widetilde{x}_0 = x_0$, the set of observable transitions is $\widetilde{T}_o = \{x \xrightarrow{\sigma} x' \in T \mid \sigma \in \Sigma_o \text{ and } x, x' \in \widetilde{X}\}$, and the set of failure transitions $\widetilde{T}_F \subseteq \widetilde{X} \times \widetilde{F} \times \widetilde{X}$ is defined as follows:

$$\{x_1 \xrightarrow{l} x_k \mid (\exists x, x' \in \widetilde{X}, \exists \sigma, \sigma' \in \widetilde{\Sigma}_o \text{ such that } (x_1 = \widetilde{x}_0 \text{ or } x \xrightarrow{\sigma} x_1 \in \widetilde{T}_o) \text{ and } x_k \xrightarrow{\sigma'} x' \in \widetilde{T}_o), \text{ and } (\exists \sigma_1 \dots \sigma_k \in \Sigma_u, \exists x_2 \dots x_{k-1} \in X \text{ such that } \forall j = 1 \dots k, x_j \xrightarrow{\sigma_j} x_{j+1} \in T \text{ and } \sigma_j \in l \text{ iff } \sigma_j \in \Sigma_f)\}$$

In the symbolic setting, the abstracted model $\widetilde{G} = \langle b^X, b^{X'}, b^\Sigma, b^F, \widetilde{X}, \widetilde{\Sigma}_o, \widetilde{F}, \widetilde{T}_o, \widetilde{T}_F\rangle$ is encoded using the same boolean variables as the global model and an additional $|\Sigma_f|$ variables $b^F = \{b^f_1, \dots, b^f_{|\Sigma_f|}\}$ needed for the failure transition labels in $\widetilde{F}$. There is a one to one correspondence between failure events and these variables, and a failure transition label is encoded as a conjunction of literals over b^F whose signs depend on whether the corresponding failure belongs to the label.

The BDDs $\widetilde{X}$, $\widetilde{T}_o$ and $\widetilde{T}_F$ representing the abstracted states and transitions are computed using Algorithms 2 and 3. Computing $\widetilde{T}_o$ and $\widetilde{X}$ is straightforward (lines 2-4 of Algorithm 2). Next, all origin states of failure transitions are computed by retrieving all states that are origins of unobservable transitions in the global model *and* that are target states of observable transitions (line 5). Finally, for each of those origin states, all failure transitions are determined and added to $\widetilde{T}_F$ (lines 6-10). This is achieved by retrieving each origin state $\widetilde{x}$ in turn and calling the function GetFailTrans of Algorithm 3. Treating states separately is necessary to avoid the introduction of additional boolean variables. This ensures that memory usage during execution is commensurable with that needed to represent the result. Recall indeed that our aim is to minimise space requirements which are the bottleneck for these off-line algorithms.

GetFailTrans($\widetilde{x}$) in Algorithm 3 returns a BDD $bddT\_\widetilde{x}$ representing the abstracted failure transitions originating at $\widetilde{x}$. In the algorithm, $bddP$ represents information about irredundant[4] prefixes of paths of the global model originating at $\widetilde{x}$ and free of observable events. This information is encoded as a set of triples $\langle \sigma, x', l\rangle$ consisting of the last event, the last state, and the set of failures encountered along the prefix. This is initialised with prefixes of length 1, that is with transitions originating at $\widetilde{x}$, and with an empty failure label (line 3). For those triples such that σ is a failure event, the event is added to l (lines 7-13). This is done by examining the subsets $bddS$ of triples of $bddP$ labelled with each given σ in turn (lines 8-9), retrieving the variable b^f_j such that j is the index of σ in the event set (line 10), and setting this variable to true by removing it from and adding a positive occurrence of it to $bddS$ (line 11). In lines 12-15, the prefixes so modified are added to the set $bddN$ of pairs $\langle x', l\rangle$ consisting of the end state and label of *new* irredundant prefixes (the transition $\widetilde{x} \xrightarrow{l} x'$ is not yet in $bddT\_\widetilde{x}$). In line 16, these prefixes are further extended for the next iteration step: the unobservable transitions originating at x' are triggered and the corresponding triples stored in $bddP$. Once all transitions $\widetilde{x} \xrightarrow{l} x'$ are computed, only those such that x' is the origin of an observable transition are kept and returned (line 18).

[4] A path prefix is redundant iff it contains the same transition $x \xrightarrow{\sigma} x'$ more than once.

Algorithm 2 $AbstractGlobal(G{=}\langle b^X, b^{X'}, b^\Sigma, X, \Sigma_o, \Sigma_u, \Sigma_f, x_0, T\rangle)$

1: INPUT: symbolic global model G
2: $\widetilde{T}_o \leftarrow T \wedge \Sigma_o$
3: $bddTarg \leftarrow$Swap(Extract($\widetilde{T}_o, b^{X'}$), $b^X, b^{X'}$)
4: $\widetilde{X} \leftarrow x_0 \vee$ Extract($\widetilde{T}_o, b^X$) $\vee$ $bddTarg$
5: $bddOrig \leftarrow$Extract($T \wedge \Sigma_u, b^X$) $\wedge$ $(x_0 \vee bddTarg)$
6: **while** IsDef($bddOrig$) **do**
7: $\quad \widetilde{x} \leftarrow$GetDisj($bddOrig$)
8: $\quad bddOrig \leftarrow bddOrig \wedge \neg\widetilde{x}$
9: $\quad \widetilde{T}_F \leftarrow \widetilde{T}_F \vee GetFailTrans(\widetilde{x})$
10: **end while**
11: OUTPUT: symbolic abstracted states $\widetilde{X}$; transitions $\widetilde{T}_o$ and $\widetilde{T}_F$

GetDisj(bdd): returns an arbitrary disjunct of the DNF represented by bdd.

Algorithm 3 $GetFailTrans(\widetilde{x})$

1: INPUT: abstracted state $\widetilde{x}$
2: $bddT\_\widetilde{x} \leftarrow false$
3: $bddP \leftarrow$ Abstract($\widetilde{x} \wedge T \wedge \Sigma_u, b^X$)$\wedge$ bddEmptyLabel
4: **while** IsDef($bddP$) **do**
5: $\quad bddN \leftarrow bddP \wedge \neg\Sigma_f$
6: $\quad bddP \leftarrow bddP \wedge \neg bddN$
7: $\quad$ **while** IsDef($bddP$) **do**
8: $\quad\quad bddS \leftarrow$Extract(GetDisj($bddP$), b^Σ) $\wedge\, bddP$
9: $\quad\quad bddP \leftarrow bddP \wedge \neg bddS$
10: $\quad\quad b^f_j \leftarrow$GetFailVar($bddS$)
11: $\quad\quad bddS \leftarrow$Abstract($bddS, \{b^f_j\}$) $\wedge\, b^f_j$
12: $\quad\quad bddN \leftarrow bddN \vee bddS$
13: $\quad$ **end while**
14: $\quad bddN \leftarrow$Extract($bddN, b^{X'} \cup b^F$) $\wedge \neg bddT\_\widetilde{x}$
15: $\quad bddT\_\widetilde{x} \leftarrow bddT\_\widetilde{x} \vee bddN$
16: $\quad bddP \leftarrow$Abstract(Swap($bddN$, b^X, $b^{X'}$) $\wedge\, T \wedge \Sigma_u, b^X$)
17: **end while**
18: $bddT_\widetilde{x} \leftarrow bddT_\widetilde{x} \wedge$Swap(Extract($\widetilde{T}_o, b^X$) b^X, $b^{X'}$) $\wedge\, \widetilde{x}$
19: OUTPUT: set $bddT\_\widetilde{x}$ of abstracted failure transitions originating at $\widetilde{x}$

bddEmptyLabel $= \wedge_{i=1}^{|\Sigma_f|} \neg b^f_i$. Abstract($bdd, B$) deletes from bdd all occurrences of variables in B. GetFailVar(bdd) returns the boolean variable b^f_j such that j is the index of the (unique) failure event encoded by bdd.

6 DIAGNOSER

A diagnoser is a deterministic finite state machine whose transitions correspond to observations and whose states correspond to the set of system states and failures that are consistent with the observations. More explicitly, its transitions are labelled with observable events, and its states are labelled with sets of pairs (v, l) denoting a state and a failure label of the abstracted model. On-line, the diagnoser efficiently maps observations to sets of possible system states and failures: it suffices to follow the path labelled by the actual observations and look up the label of the resulting diagnoser state.

Formally, let $\widetilde{G} = \langle \widetilde{X}, \widetilde{\Sigma}_o, \widetilde{F}, \widetilde{x}_0, \widetilde{T}_o, \widetilde{T}_F \rangle$ be the abstracted model and let $V = \{\widetilde{x_0}\} \cup \{x' \in \widetilde{X} \mid \exists \sigma \in \widetilde{\Sigma}_o, \exists x \in \widetilde{X} \text{ s.t. } x \xrightarrow{\sigma} x' \in \widetilde{T}_o\}$ be the set of target states of observable transitions. The diagnoser is $\hat{G} = \langle \hat{X}, \hat{\Pi}, \hat{\Sigma}, \hat{x}_o, \hat{R}, \hat{T} \rangle$, where $\hat{X} = \{\hat{x}_1, \ldots, \hat{x}_q\}$ is the set of diagnoser states, $\hat{\Pi} = V \times \widetilde{F}$ is the set of pairs that can belong to diagnoser state labels, $\hat{\Sigma} = \widetilde{\Sigma}_o$ is the set of diagnoser transition labels, $\hat{x}_0$ is the initial diagnoser state, $\hat{R} \subseteq \hat{X} \times \hat{\Pi}$ is the diagnoser state labelling relation which associates a state to the pairs in its label and verifies $\hat{R}(\hat{x}_0) = \{(\widetilde{x}_0, \emptyset)\}$ (by abuse of notation, we use the function notation $\hat{R}(\hat{x})$ for the label of state $\hat{x}$), and $\hat{T} \subseteq \hat{X} \times \hat{\Sigma} \times \hat{X}$ is the set of diagnoser transitions, which verify: $\hat{x} \xrightarrow{\sigma} \hat{x}' \in T$ iff

$$\begin{array}{l}\hat{R}(\hat{x}') = \\ \{(v', l') \mid \quad \exists (v, l) \in \hat{R}(\hat{x}) \text{ such that either} \\ \qquad\qquad v \xrightarrow{\sigma} v' \in \widetilde{T}_o \text{ and } l' = l, \text{ or} \\ \qquad\qquad \exists v \xrightarrow{l''} v'' \in \widetilde{T}_F \text{ and } \exists v'' \xrightarrow{\sigma} v' \in \widetilde{T}_o \text{ and } l' = l \cup l''\}\end{array}$$

In practice, we need of course to restrict the set of states and transitions to those reachable from the initial diagnoser state, i.e., to restrict attention to the smallest $\hat{X}$ containing $\hat{x}_0$ and closed under $\hat{T}$; however, for readability, we here again spare the reader with the logical definition of closure.

In the symbolic setting, the diagnoser $\hat{G} = \langle b^S, b^{S'}, b^X, b^{X'}, b^\Sigma, b^F, \hat{X}, \hat{\Pi}, \hat{\Sigma}, \hat{x}_o, \hat{R}, \hat{T} \rangle$ is encoded using the same boolean variables as the abstracted model, together with an additional $2\lceil \log_2 q \rceil$ variables $b^S = \{b^s_1, \ldots, b^s_{\lceil \log_2 q \rceil}\}$ and $b^{S'} = \{b^{s'}_1, \ldots, b^{s'}_{\lceil \log_2 q \rceil}\}$ needed to encode diagnoser states in their role of origin and target of transitions. A complication here is that the number q of diagnoser states, and therefore the number of variables needed, is *a priori* unknown and even difficult to estimate. In the worst case $q = 2^{|\hat{\Pi}|}$ and therefore $2|V||\widetilde{F}|$ new variables are theoretically needed. However, in practice, q will be much smaller and introducing that many variables will lead to an unnecessarily costly representation. To remedy this, we start with one single variable to encode the initial diagnoser state, and continually increase the number of variables, as needed during execution. Every time a new variable b^s_j is needed, we update all BDDs containing variables in b^S (resp. $b^{S'}$) by conjoining them with $\neg b^s_j$ (resp. $\neg b^{s'}_j$).

Algorithms 4 and 5 show the symbolic computation of the set of states $\hat{X}$, and of the labelling and transition relations $\hat{R}$ and $\hat{T}$ of the diagnoser. In Algorithm 4, the initial diagnoser state $\hat{x}_0$ is first computed (line 2) using the function GetNewState which returns the encoding of a new diagnoser state and if needed, handles the introduction of new variables to b^S as discussed above. Then, the diagnoser labelling relation $\hat{R}$, the set of diagnoser states $\hat{X}$, and the set $new\hat{X}$ of diagnoser states from which transitions still need to be computed are initialised (lines 3-5). Until a fixed point is reached, a new diagnoser state $\hat{x}$ is retrieved from $new\hat{X}$ (lines 7-8), and information $bddTrans\_\hat{x}$ about all diagnoser transitions originating at $\hat{x}$ is computed (line 9) using the function GetTrans given in Algorithm 5. GetTrans returns a BDD representing the set of all observable events involved in these transitions, together with the labels of the respective diagnoser states reached. To finish processing $\hat{x}$, we first need to obtain all the system states and sets of failures consistent with each single observable event $bddSingleObs$ in turn; these will form the label $bddLab\_\hat{x}'$ of the diagnoser state $\hat{x}'$ successor of $\hat{x}$ via $bddSingleObs$ (lines 10-14). Then, $\hat{x}'$ is computed using the function GetExistingState (line 15). This function returns the diagnoser state with label $bddLab\_\hat{x}'$ if it exists, and false otherwise.[5] In the latter case (line 16), a new state needs to be introduced (line 17) and $\hat{X}$, $new\hat{X}$ and $\hat{R}$ updated. Finally, the computed transition is added to $\hat{T}$ (line 22).

Algorithm 4 $BuildDiag(\widetilde{G}{=}\langle b^X, b^{X'}, b^\Sigma, b^F, \widetilde{X}, \widetilde{\Sigma}_0, \widetilde{F}, \widetilde{x}_0, \widetilde{T}_o, \widetilde{T}_F \rangle)$

1: INPUT: symbolic abstracted model $\widetilde{G}$
2: $\hat{x}_0 \leftarrow$ GetNewState()
3: $\hat{R} \leftarrow \hat{x}_0 \wedge \widetilde{x}_0 \wedge$ bddEmptyLabel
4: $\hat{X} \leftarrow \hat{x}_0$
5: $new\hat{X} \leftarrow \hat{x}_0$
6: **while** IsDef($new\hat{X}$) **do**
7: $\quad \hat{x} \leftarrow$GetDisj($new\hat{X}$)
8: $\quad new\hat{X} \leftarrow new\hat{X} \wedge \neg\hat{x}$
9: $\quad bddTrans\_\hat{x} \leftarrow$ GetTrans($\hat{x}$)
10: $\quad bddObs \leftarrow$Extract($bddTrans\_\hat{x}, b^\Sigma$)
11: $\quad$ **while** IsDef($bddObs$) **do**
12: $\quad\quad bddSingleObs \leftarrow$GetDisj($bddObs$)
13: $\quad\quad bddObs \leftarrow bddObs \wedge \neg bddSingleObs$
14: $\quad\quad bddLab_\hat{x}' \leftarrow$Abstract($bddSingleObs \wedge bddTrans\_\hat{x}, b^\Sigma$)
15: $\quad\quad \hat{x}' \leftarrow$ GetExistingState($bddLab\_\hat{x}'$)
16: $\quad\quad$ **if** $\neg$IsDef($\hat{x}'$) **then**
17: $\quad\quad\quad \hat{x}' \leftarrow$ GetNewState()
18: $\quad\quad\quad \hat{R} \leftarrow \hat{R} \vee (\hat{x}' \wedge bddLab\_\hat{x}')$
19: $\quad\quad\quad \hat{X} \leftarrow \hat{X} \vee \hat{x}'$
20: $\quad\quad\quad new\hat{X} \leftarrow new\hat{X} \vee \hat{x}'$
21: $\quad\quad$ **end if**
22: $\quad\quad \hat{T} \leftarrow \hat{T} \vee (\hat{x} \wedge bddSingleObs \wedge$Swap($\hat{x}', b^S, b^{S'}$))
23: $\quad$ **end while**
24: **end while**
25: OUTPUT: symbolic diagnoser states, labelling and transition relations $\hat{X}, \hat{R}, \hat{T}$

GetTrans, shown in Algorithm 5, takes a diagnoser state $\hat{x}$ as input, and returns the set $bddTrans$ of observable events labelling the diagnoser transitions originating at $\hat{x}$ together with the sets of labels of the resulting diagnoser states. To compute $bddTrans$, we are interested in two types of transitions of the abstracted model: the observable transitions $bddObsTrans$ originating at any of the system states in the label $bddLabel$ of $\hat{x}$, and two step transitions $bddFailTrans$ consisting of a failure transition originating at any of the system states in $bddLabel$ followed by an observable transition. The algorithm starts by computing the label $bddLabel$ of $\hat{x}$ (line 2), and $bddObsTrans$ (line 3), that is the set $\{(\sigma, (v', l)) \mid \exists v \text{ s.t. } (v, l) \in \hat{R}(\hat{x}) \text{ and } v \xrightarrow{\sigma} v' \in \widetilde{T}_o\}$. (lines 4-16), that is the set $\{(\sigma, (v', l \cup l'')) \mid \exists v, v'' \text{ s.t.} (v, l) \in \hat{R}(\hat{x}),\ v \xrightarrow{l''} v'' \in \widetilde{T}_f, \text{ and } v'' \xrightarrow{\sigma} v' \in \widetilde{T}_o\}$. This is done as follows. For each failure set $bddFailures$ in $\hat{x}$'s label $bddLabel$ (lines 5,6 and 8), the set

[5] The symbolic implementation of GetExistingState does not enumerate individual states: the set of existing states whose label is a subset of the given one is computed, and then filtered to select those states whose label is a superset of the given one; see [13] for details.

$bddStates$ of system states paired with these failures in $bddLabel$ is determined (lines 7 and 9). Then, the set $bddNewFailTrans$ of failure transitions originating at those states in the abstract model are triggered, $bddFailures$ is added to the failure set labelling each of those transitions, and the resulting transitions are added to $bddFailTrans$ (lines 10-13). After all first steps of the two-step transitions have been considered, the second steps are considered by triggering observable transitions in the system states resulting from the first step (lines 15-16). Finally, the function returns the union of $bddObsTrans$ and $bddFailTrans$, with appropriate naming of the state variables (line 17).

Algorithm 5 $GetTrans(\hat{x})$

1: INPUT: diagnoser state $\hat{x}$
2: $bddLabel \leftarrow$Abstract$(\hat{x} \wedge \hat{R}, b^S)$
3: $bddObsTrans \leftarrow$Abstract$(bddLabel \wedge \widetilde{T}_o, b^X)$
4: $bddFailTrans \leftarrow false$
5: **while** IsDef($bddLabel$) **do**
6: $bddFailures \leftarrow$Extract(GetDisj$(bddLabel), b^F)$
7: $bddStates \leftarrow bddFailures \wedge bddLabel$
8: $bddLabel \leftarrow bddLabel \wedge \neg bddStates$
9: $bddStates \leftarrow$ Extract$(bddStates, b^X)$
10: $bddNewFailTrans \leftarrow bddStates \wedge \widetilde{T}_F$
11: **if** IsDef($bddNewFailTrans$) **then**
12: $bddFailTrans \leftarrow bddFailTrans \vee$ AddFail$(bddFailures, bddNewFailTrans)$
13: **end if**
14: **end while**
15: $bddFailTrans \leftarrow$Swap(Abstract$(bddFailTrans, b^X), b^X, b^{X'})$
16: $bddFailTrans \leftarrow$Abstract$(bddFailTrans \wedge \widetilde{T}_o, b^X)$
17: $bddTrans \leftarrow$Swap$(bddObsTrans \vee bddFailTrans, b^X, b^{X'})$
18: OUTPUT: set $bddTrans$ of the observable events that can be triggered from $\hat{x}$ together with the labels of the corresponding target diagnoser states.

Where $bddL$ represents a set l of failures and $bddT$ a set $\{v_1 \stackrel{l_1}{\rightarrow} v'_1, \ldots, v_k \stackrel{l_k}{\rightarrow} v'_k\}$ of failure transitions of the abstracted model, AddFail$(bddL, bddT)$ returns a bdd representing the set of failure transitions $\{v_1 \stackrel{l_1 \cup l}{\rightarrow} v'_1, \ldots, v_k \stackrel{l_k \cup l}{\rightarrow} v'_k\}$.

7 RESULTS

Our symbolic approach has been implemented in C++ on top of the CUDD BDD package [14]. In order to compare the performance of our algorithms with Sampath's original approach, we used our own enumerative implementation for two reasons: first, the original one, the UMDES-LIB software library, was only able to compute the first of our examples and second, it did not report the distinct space requirements for the diagnoser representation. Our implementations enable us to present experimental evidence that the symbolic approach yields important gains in time and space, taking a system consisting of a switch and two control stations of the Transpac telecommunication network as example [11].

In this example, there are 9 observable events, 11 failure types, and 8 other unobservable events. The switch model has 12 states and 18 transitions, the primary control station 13 states and 15 transitions and the backup control station 19 states and 28 transitions. This yields a global model of 1062 states and 2911 transitions. In order to observe how the two approaches scale, we also considered "lighter" versions of the example, where groups of failure types are fusioned. This yields 5 different Versions $V_1 \ldots V_5$, with a number of failure types ranging from 3 to the original 11. Table 1 shows the number of states and transitions of the diagnoser for the various versions.

Table 1 Diagnoser size

	$\mathbf{V}_1$	$\mathbf{V}_2$	$\mathbf{V}_3$	$\mathbf{V}_4$	$\mathbf{V}_5$
$\lvert\hat{X}\rvert$	353	921	2500	4355	18474
$\lvert\hat{T}\rvert$	2183	5774	16530	31024	120698

Experiments were run on a 1GHz pentium III with 512 Mbytes of memory. The top chart in Figure 3 compares the time taken by the symbolic and enumerative methods to produce the diagnoser. After 2 days of computation, the enumerative approach was unable to compute a diagnoser for the two larger versions, while the symbolic approach remained feasible. In order to determine the space requirements for the enumerative diagnoser representation for these examples, we retrieved Sampath's diagnoser on the basis of the symbolic one. This was done by examining the paths of the BDDs used to represent the diagnoser.

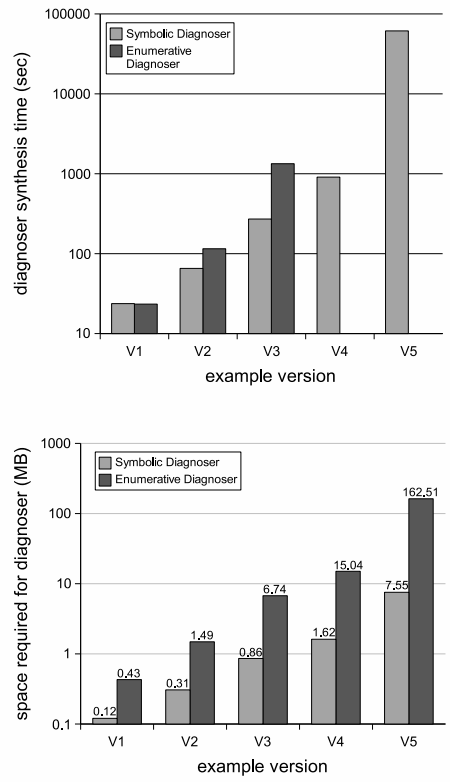

Figure 3. Time and space performance comparison

The bottom chart in Figure 3 compares the space needed to store the resulting diagnoser.[6] The superiority of the symbolic method increases with the model size, and exceeds an order of magnitude for the largest version. From the results, it can be conjectured that the space requirements of the symbolic approach for large models will often only represent a neglectable portion of those of the enumerative

[6] Given the space requirements to represent one BDD node, the size of the symbolic diagnoser was determined by counting the nodes of the BDDs representing the diagnoser.

setting. [13] shows that this remains true even if we omit the system's states in the labels of the diagnoser states.

We found that, for both approaches, most of the space was used to store labels. In particular, our symbolic encoding only performs marginally better than the enumerative approach with respect to storing diagnoser transitions. This is due to the fact that the boolean variables introduced to represent transition sets allow the representation of all transitions that are theoretically possible, while in practice, the diagnoser is deterministic and not all observations are possible in each given diagnoser state. For instance, the diagnoser for version V_5 only contains 0.004% of the theoretically possible transitions. [13] discusses alternative encodings which may remedy some of these problems at the expense of computation time.

Finally, it is worth mentioning that the symbolic representation appears to preserve the real-time property of the diagnoser for on-line diagnosis. We found that there was little variation in diagnosis time across the different example versions, and in our experiments, the time taken to treat 1000 observations never exceeded 100 ms.

8 CONCLUSION, RELATED & FUTURE WORK

We have presented a symbolic framework based on binary decision diagrams for the diagnosis of discrete-event systems. The framework enables the synthesis of a symbolic version of Sampath's diagnoser [12], while requiring considerably lower space and time than the enumerative approach. This results from the fact that BDDs are particularly suitable to compactly represent the large sets of system states and failures labelling the diagnoser states.

Our framework is not limited to fault diagnosis using the diagnoser approach. In [13], we also give algorithms for checking diagnosability, as well as fault diagnosis algorithms based on all the models presented here: abstracted model, global model, and component model. While the diagnoser yields much superior diagnosis time performance[7], the ability to diagnose based on the other models is also important since the diagnoser will not always be computable. In particular, the abstracted model appears to be an interesting alternative to the diagnoser.

In the past, off-the-shelf symbolic model-checkers have been used for failure diagnosis by Cordier and Largouët [8] and for checking diagnosability by Cimatti and colleagues [4]. In [8], the model checker is the diagnoser, so to speak. It checks whether traces of the system exist which satisfy a temporal logic formula stating what the observations are and expressing constraints between their time of occurrence. While this approach is very attractive for off-line diagnosis, it is much less suited to on-line diagnosis because symbolic model-checkers do not provide facilities for incremental computation.

The work by Marchand and Rozé [9] is the closest to ours. They present a theoretical framework for modelling a system in terms of polynomial equations which are generalisations of boolean formulae, and define a form of non-deterministic diagnoser within this framework. The framework is state-based, meaning that observations and failures are directly associated to states (by means of polynomial equations). The non-deterministic diagnoser is obtained by gathering the states of the global model into observational/failure equivalence classes. This non-deterministic diagnoser is quite different from ours and still requires computations of non-trivial complexity to be performed on-line. Arguably, the traditional event-based view we adopt is more natural than the state-based view for modelling the systems typically considered in model-based diagnosis. Also, our event-based framework lends itself to a straightforward extension to intermittent faults along the lines of [7], while extending a state-based framework in this direction is more difficult. However, it would be interesting to implement the approach by Marchand and Rozé ([9] does not report any implementation), and compare the results to ours.

In the future, we plan to improve our results by experimenting with alternative encodings described in [13], dedicated heuristics for variable ordering, and transition relation partitioning techniques [5]. The symbolic framework can also be used for decentralised diagnosis, as in [10], and we shall investigate this possibility in detail. Another line of work is to extend our framework to stochastic systems and compute probability distributions on diagnoses, using for instance algebraic decision diagrams which are generalisation of BDDs to real-valued functions over the booleans (see e.g. [14]). Finally, integrating diagnosis and planning for repair or reconfiguration actions is one of the most significant challenges faced by the field of model-based diagnosis [6]. Given the recent success of planning techniques based on symbolic model-checking (see e.g. [2]), we believe that our framework will prove a good basis for addressing this challenge.

[7] For instance, we found that the ratio of the diagnosis time based on the component model over that based on the diagnoser was even higher than the ratio of the space occupied by the diagnoser over that occupied by the component model

References

[1] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella, 'Diagnosis of large active systems', *Artificial Intelligence*, **110**(1), 135–183, (1999).

[2] P. Bertoli, A. Cimatti, J. Slaney, and S. Thiébaux, 'Solving power supply restoration problems with planning via symbolic model-checking', in *Proc. 15^{th} European Conference on Artificial Intelligence (ECAI-02)*, pp. 576–580, (July 2002).

[3] R. E. Bryant, 'Graph-based algorithms for boolean function manipulation', *IEEE Transactions on Computers*, **C-35**(8), 677–691, (1986).

[4] A. Cimatti, C. Pecheur, and R. Cavada, 'Formal verification of diagnosability via symbolic model checking', in *18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, (2003).

[5] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*, MIT Press, 1999.

[6] L. Console and O. Dressler, 'Model-based diagnosis in the real world: lessons learned and challenges remaining', in *16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, (1999).

[7] O. Contant, S. Lafortune, and D. Teneketzis, 'Failure diagnosis of discrete event systems: The case of intermittent faults', in *Proc. 41st IEEE Conference on Decision and Control (CDC-02)*, pp. 4006–4011, (2002).

[8] M. O. Cordier and C. Largouët, 'Using model-checking techniques for diagnosing discrete-event systems', in *12th International Workshop on Principles of Diagnosis (DX-01)*, pp. 39–46, (2001).

[9] H. Marchand and L. Rozé, 'Diagnostic de pannes sur des systèmes à événements discrets: une approche à base de modèles symboliques', in *13ème Congrès AFRIF-AFIA de Reconnaissances des Formes et Intelligence Artificielle*, pp. 191–200, (2002). In French.

[10] Y. Pencolé, M. O. Cordier, and L. Rozé, 'A decentralized model-based diagnostic tool for complex systems', *International Journal on Artificial Intelligence Tools*, **11**(3), 327–346, (2002).

[11] L. Rozé and M.-O. Cordier, 'Diagnosing discrete-event systems: extending the "diagnoser" approach to deal with telecommunication networks', *Journal of Discrete Event Dyanmic Systems: Theory and Applications*, **12**(1), (2002).

[12] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, 'Failure diagnosis using discrete event models', *IEEE Transactions on Control Systems Technology*, **4**(2), 105–124, (1996).

[13] A. Schumann, *Is symbolic technology applicable to the diagnosis of discrete event systems?*, Master's thesis, Faculty of Business, Administration, Economics and Social Sciences, University of Rostock, 2003.

[14] F. Somenzi. CUDD: CU Decision Diagram Package Release 2.3.0. University of Colorado at Boulder, 1998.

[15] J. Sztipanovits and A. Misra, 'Diagnosis of discrete event systems using ordered binary decision diagrams', in *Seventh International Workshop on Principles of Diagnosis*, (1996).