Working Paper Series
ISSN 1403-266X

VERIFICATION OF DIAGNOSABILITY BASED ON COMPOSITIONAL BRANCHING BISIMULATION

Mona Noori Hosseini, Bengt Lennartson

Working Paper: R013/2014
June 24, 2014

©Mona Noori Hosseini, Bengt Lennartson

Department of Signals and Systems
Chalmers University of Technology
Göteborg
Sweden

Verification of Diagnosability Based on Compositional Branching Bisimulation

Mona Noori-Hosseini, Bengt Lennartson
Chalmers University of Technology
SE-412 96, Gothenburg, Sweden
{noori,bengt.lennartson}@chalmers.se

Abstract

This paper presents an efficient diagnosability verification technique, based on a general abstraction approach. We exploit branching bisimulation with explicit divergence (BBED), which preserves the temporal logic property that verifies diagnosability. Furthermore, using compositional abstraction for modular diagnosability verification offers additional state space reduction in comparison to the state-of-the-art techniques.

1. Introduction

Failure is a deviation of a system from its normal behavior. The task of identifying and isolating deviations from desired behavior is called failure diagnosis and the ability to deduce about previously occurred failures within a bounded number of observations is called diagnosability [20]. A system is diagnosable if each failure can be uniquely identified through a number of events in partial observations.

There are two approaches, namely, language specification [19, 21], and failure events [14], to show the faulty behavior in discrete event systems. In the language specification approach, a specification represents the non-faulty behavior of the system and every deviation from that specification leads to a failure. In the failure event approach, the failures are shown in the same model using events. For both approaches, there are polynomial diagnosability algorithms, [19, 23, 25, 11, 14, 24]. However, although polynomial time algorithms exists, the state space increases exponentially when modular systems are composed. Thus, it is often too complex to analyze systems of industrial size.

To tackle the computational complexity, abstraction-based diagnosability verification algorithms have been recently introduced for both automata and Petri net models [10, 15, 21], including techniques for modular systems [22]. In [21], the computational effort for diagnosability verification methods is reduced by determining sufficient conditions, such that diagnosability of the original system follows from diagnosability of an abstracted model. Moreover, it is shown that if the abstracted system is not diagnosable, then the original system is not diagnosable, if all observable events remain after abstraction. This requirement implies that in general only limited abstractions can be expected for non-diagnosable systems.

The aforementioned abstraction techniques used language specification. In some other works, e.g, [15], event-based abstraction technique are exploited, which are behaviorally equivalent to the original model. The classification of different behavioral equivalences is made in [6]. One of the most well known behavioral equivalences is weak bisimulation, also called observation equivalence, [16], which holds a very coarse equivalence. Another slightly more restricted one is *branching bisimulation*, [8]. Unlike weak bisimulation, branching bisimulation preserves the branching structure of processes, in the sense that it preserves computations together with the potentials in all intermediate states that are passed through, even if silent moves are involved. An extension of branching bisimulation, which is also critical for our purposes, is *branching bisimulation with explicit divergence* (BBED). It means that silent loops resulting from abstraction are preserved [7].

BBED has the important property that the temporal eventually operator (E) is preserved, which is not the case for weak bisimulation. Even more, the complete temporal logic CTL* [1], except for the next operator X, called CTL*-X, is preserved for BBED. This is critical for diagnosability verification, and means that BBED, but not weak bisimulation, can be used for diagnosability abstraction. BBED is applied on event labels on labeled transition systems (LTS), while CTL*-X expressions are based on state labels included in Kripke structures (KS). However, in [17] a translation between LTS to KS is established, so that CTL*-X can also be considered as a logic on LTS.

In this work, we use an abstraction-based verification method in order to reduce the computational complexity. Here, we induce the failure information to state labels based on a diagnosability algorithm. Then, exploiting BBED we perform event-based abstraction technique while preserving silent loops. Preserving all uncertain loops, where diagnosability can not be decided, is important to claim the diagnosability of a system correctly. Using the mentioned transformation in [17], we get the equivalent abstracted KS of the system with silent events, where we perform CTL based model checking for diagnosability verification.

The contribution of this paper is that BBED is proposed and developed for abstraction-based diagnosability verification. Compared to previous works on abstraction for diagnosability, our approach gives more efficient abstractions. One reason is that observable events can also be abstracted, still showing equivalence between the abstracted and the original system concerning diagnosability. Furthermore, unlike earlier language-based approach, where all transitions with the same event must be considered for abstraction, in the proposed approach transitions with the same event are abstracted individually, once again resulting in more efficient abstractions. Finally, compositional abstraction is applied for modular systems, which can be considered as a partitioning technique avoiding temporary state-space complexity in the abstraction phase.

The rest of the paper is organized as follows. Section II presents preliminary concepts. Section III gives the definition of diagnosability. In Section IV, the temporal logic formulation for diagnosability verification is presented. Section V is on branching bisimulation abstraction. In Section VI, compositional abstraction is described. Finally, conclusions are drawn in Section VII.

2. Preliminaries

The event observation projection [3], called observation mask in [14], is a mapping from the original event set Σ to a smaller observable event set $\Sigma_o \subseteq \Sigma$, i.e., $P : \Sigma \rightarrow \Sigma_o \cup \{\varepsilon\}$, which can be extended to Σ^* that is the set of all event traces generated from Σ. Here, ε shows unobservable events, so that we have $s \in \Sigma^*$, $\sigma \in \Sigma$: $P(s\sigma) = P(s)\, P(\sigma)$, with $P(\varepsilon) = \varepsilon$ and $P(\sigma) = \varepsilon$ for all $\sigma \in \Sigma_u$, the unobservable event set. Moreover, $\Sigma = \Sigma_o \dot{\cup} \Sigma_u$, and $\Sigma_o = \Sigma_s \dot{\cup} \Sigma_\ell$, with Σ_s denoting shared events which are involved in more than one module and Σ_u denoting local events which only belong to one module. $\Sigma_u = \Sigma_f \dot{\cup} \Sigma_n$, where Σ_f and Σ_n are failure and non-failure unobservable events, respectively.

Definition 1 (Kripke Structure (KS)) *[17]* Let AP be a fixed nonempty set of atomic proposition names, ranged over by $q, p, \ldots$. A *Kripke Structure* (KS) is a triple $K = (Q, L, \rightarrow)$, where Q is a set of states, $L : Q \rightarrow 2^{AP}$ is the proposition labeling, and $\rightarrow \subseteq Q \times Q$ is the transition relation; an element $(q, p) \in \rightarrow$, usually written as $q \rightarrow p$, is called a transition. □

Based on KS, we extend automata including state labels on $\mathcal{G}$. The transition system definition in the following includes both labels on states and transitions.

Definition 2 (Transition System) A *transition system* (TS) is a tuple $\mathcal{G} = \langle Q, \Sigma, \rightarrow, Q_0, AP, L\rangle$ where Q is a set of states, Σ is a set of events, $\rightarrow \subseteq Q \times \Sigma \times Q$ is a transition relation, also Q_0 denotes the set of initial states, AP is a set of atomic propositions, and $L : Q \rightarrow 2^{AP}$ represents a proposition labeling. □

Now, we define synchronous composition on transition systems as follows.

Definition 3 (Synchronous Composition) Let $\mathcal{G}_i = \langle Q_i, \Sigma_i, \rightarrow_i, Q_{0i}, AP_i, L_i\rangle$ for $i = 1, 2$ be two automata. The synchronous composition of $\mathcal{G}_1$ and $\mathcal{G}_2$ is defined as $\mathcal{G}_1 \parallel \mathcal{G}_2 = \langle Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \rightarrow, Q_1^0 \times Q_2^0, AP_1 \cup AP_2, L_1 \times L_2\rangle$ where

$$(q_1, q_2) \xrightarrow{a} (p_1, p_2) : a \in (\Sigma_1 \cap \Sigma_2), q_1 \xrightarrow{a}_1 p_1, q_2 \xrightarrow{a}_2 p_2$$
$$(q_1, q_2) \xrightarrow{a} (p_1, q_2) : a \in (\Sigma_1 \backslash \Sigma_2), q_1 \xrightarrow{a}_1 p_1$$
$$(q_1, q_2) \xrightarrow{a} (q_1, p_2) : a \in (\Sigma_2 \backslash \Sigma_1), q_2 \xrightarrow{a}_2 p_2.$$

□

3. Diagnosability of Discrete Event Systems

In this section, we define the diagnosability notion, along with a polynomial algorithm for diagnosability verification of modular systems. In the end, the algorithm is illustrated through an example.

Definition 4 (Failure Assignment Function) Failure assignment function is a mapping from the original event set Σ to state failure labels N or F, i.e., $\psi : \Sigma \rightarrow \{F, N\}$. It means that if $\sigma \in \Sigma$ is not a failure event, it is projected to N; otherwise it is projected to the F. All reachable states after a state with label F, keep it as their labels. □

For the sake of simplicity, here one failure class is considered. Assume a system $\mathcal{G}$ to be live without any cycle of unobservable events, and let the set of all traces generated by $\mathcal{G}$ be denoted $\mathcal{L}(\mathcal{G})$. Furthermore, consider a trace $s \in \mathcal{L}(\mathcal{G})$ ending with a failure, and a sufficiently long trace m obtained by extending s. The system $\mathcal{G}$ is then *diagnosable* if every trace w that is observation equivalent to m, also contains a failure. Formally, the diagnosability is defined as follows.

Definition 5 (Diagnosability) With respect to the event observation projection and the failure assignment function $\psi : \Sigma \rightarrow F \cup N$, a system $\mathcal{G}$ is diagnosable if

$$(\forall F)(\exists n_i \in \mathbb{N})(\forall s \in \mathcal{L}(\mathcal{G}),\ \psi(s_f) = F)$$
$$(\forall m = st \in \mathcal{L}(\mathcal{G}), \|t\| \geq n_i) \Rightarrow$$
$$(\forall w \in \mathcal{L}(\mathcal{G}),\ P(w) = P(m))(\exists r \in pr(\{w\}),\ \psi(r_f) = F)$$

Here, s_f and r_f are the last events in traces s and r, respectively, and $pr(\{w\})$ is the set of all prefixes of w. □

3.1. Diagnosability Algorithm

Diagnosability verification answers the question whether a failure can always be detected or not. In [14], a verifier is introduced that abstracts away all unobservable and failure transitions by first constructing an observable automaton, $\mathcal{G}_o$, whose definition is presented in Algorithm

1 in the following. Then, $G = \mathcal{G}_o \| \mathcal{G}_o$ is calculated. The verifier G is checked for the existence of possible uncertain cycles, i.e., cycles including states with different failure labels. All other states and their associated transitions are then deleted. If the remaining graph contains at least one cycle, the system is not diagnosable. This verifier has polynomial complexity $\mathcal{O}(n_q^4 n_t)$, where n_q and n_t are the number of states and transitions, respectively.

Algorithm 1: [14] For diagnosability verification in a modular system, let each module be $\mathcal{G}_i = (Q_i, \Sigma_i, \rightarrow_i, Q_{0i})$ for $i \in \mathcal{I} = \{1, \ldots, k\}$, where k is the number of modules. The following algorithm verifies the diagnosability of $\mathcal{G} = \|_{i \in \mathcal{I}} \mathcal{G}_i$:

1. Augment states of each $\mathcal{G}_i$ with failure labels (N, F), based on the failure assignment function, resulting in $\mathcal{G}_i = (Q_i, \Sigma_i, \rightarrow_i, Q_{0i}, AP, L_i)$

2. Obtain a non-deterministic automaton $\mathcal{G}_i^o = (Q_i^o, \Sigma_i^o, \rightarrow_i^o, Q_{0i}^o, AP, L_i^o)$, where $Q_i^o = \{(q_i, \ell_i) \,|\, q_i^o \in \bar{Q} \cup \{q_{0i}^o\}\}$ and $\bar{Q}_i = \{q_i \in Q_i | \exists\, (p_i, \sigma_i, q_i) \in \rightarrow_i$ with $P(\sigma_i) \neq \varepsilon\}$.

3. Compute the verifier automaton, $G_i = \mathcal{G}_i^o \parallel \mathcal{G}_i^o$.

4. Replace all state labels (NN, FF) and (NF, FN) with C (certain state labels) and U (uncertain state labels), respectively.

5. Verify the existence of uncertain cycles, i.e., loops over states with label (U) in $G = \|_{i \in \mathcal{I}} G_i$.

In the synchronization of $G = \|_{i \in \mathcal{I}} G_i$, we may get different combinations of k number of labels in each state. Having at least one U label in each state is enough to make that state uncertain. Therefore, all states with at least one U are replaced with U, and states with only C labels are replaced with C. For instance, with $k = 2$, we may get different combinations of CC, CU, UC and UU in G. Except CC which is replaced with C, the rest of CU, UC and UU are replaced with U. This rule is always applied in the synchronization.

3.2 Diagnosability Verification in Modular Structure

In diagnosability verification of a system in a modular structure, if all modules are diagnosable, based on the following, the total system is diagnosable. Therefore, we assume at least one non-diagnosable module in our evaluation.

Diagnosability verification in a modular structure contains three different cases [4]: (a) if the monolithic system ($\mathcal{G}$) is not diagnosable then necessarily one of the modules ($\mathcal{G}_i$) is not diagnosable; (b) if all local modules are diagnosable then the monolithic system ($\mathcal{G}$) is also diagnosable; and (c) if a local module is not diagnosable then the monolithic system may or may not be diagnosable. The above cases cover all possible outcomes that may occur when modules of a system are composed in parallel. We focus on Case (c), where it is not possible to evaluate the diagnosability of $\mathcal{G}$ by finding the diagnosability of each module separately. The following example includes a non-diagnosable automaton.

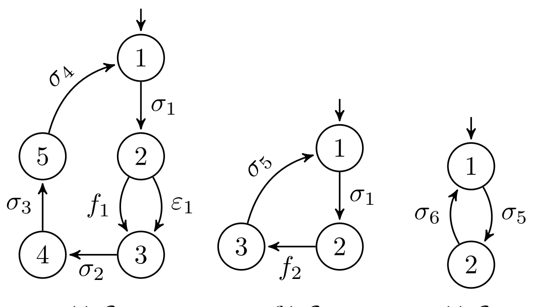

(a) $\mathcal{G}_1$ (b) $\mathcal{G}_2$ (c) $\mathcal{G}_3$

Figure 1 The three automata of Example 1.

Example 1 *Consider the three automata depicted in Fig. 1. Events descriptions are as follows;* $\Sigma_o = \{\sigma_1, \ldots, \sigma_6\}$, $\Sigma_{uo} = \varepsilon_1$, $\Sigma_f = \{f_1, f_2\}$, *and* $\Sigma_s = \{\sigma_1, \sigma_5\}$. *Both failures belong to one failure class.* $\mathcal{G}_1$ *is not diagnosable, and* $\mathcal{G}_2$ *and* $\mathcal{G}_3$ *are diagnosable. However, based on the above description, the diagnosability of the monolithic system* $\mathcal{G} = \mathcal{G}_1 \parallel \mathcal{G}_2 \parallel \mathcal{G}_3$ *is not clear yet. Therefore, based on Algorithm 1, we evaluate the diagnosability of* $\mathcal{G}$. *Steps 2 and, 3-4 of the algorithm are illustrated in Fig. 2 and Fig. 3, respectively. In Fig. 3, only the updated failure labels are shown in each state. The verifier G, has 78 states, 150 transitions and uncertain loops. Thus the system* $\mathcal{G}$ *is not diagnosable.*

4 Temporal Logic

Diagnosability test is shown to be an instance of temporal logic model-checking [12]. In the last step of diagnosability algorithm, it is needed to determine the existence of uncertain cycles using temporal logic. Thus, we describe the temporal logic and quantifiers, briefly.

There are particularly two common used temporal logics; *linear-temporal logic* (LTL) where time is linear, and *computation-tree logic* (CTL) where time is branching. Temporal logic includes temporal quantifiers, which in CTL are expressed in pairs. In CTL, as well as the temporal operators U, F, G and X, there are also quantifiers A (universal) and E (existential) which express “all paths” and “exists a path”, respectively, [13]. Furthermore, G is the universal quantifier and F is the existential quantifier, ranging over the states along a particular path. Moreover, X means next and U until.

In our case, a CTL* specification for diagnosability verification is used, cf. [18]. CTL* is an extension of CTL and LTL, which combines the features of both logics, and thus is more expressive than either of them. The specification is

$$\mathsf{AG}\ \mathsf{AF}(C) \tag{1}$$

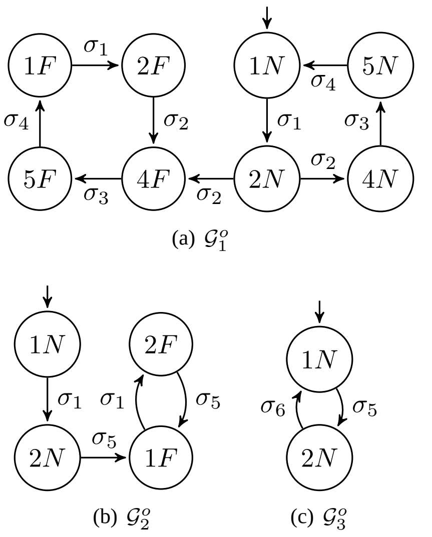


Figure 2 The $\mathcal{G}_i^o$ of the three automata of Example 1.

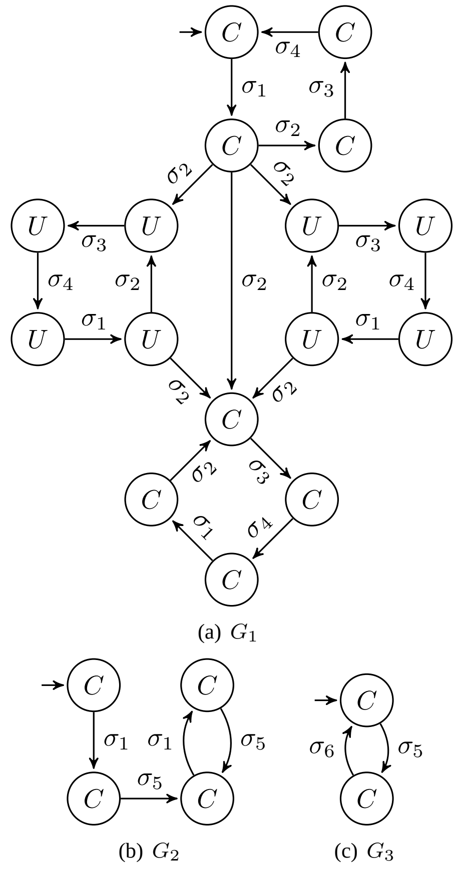


Figure 3 The G_i of the three automata of Example 1.

where C denotes certain labels. This implies that for all computation paths beginning from an initial state, there will be some future states where C holds infinitely often. If this specification holds, the system is diagnosable. This statement becomes clear by investigating its negation, meaning that a system is not diagnosable when

$$\neg \mathsf{AG}\ \mathsf{AF}\,(C) \equiv \mathsf{EF}(\neg \mathsf{AF}\,(C)) \equiv \mathsf{EF}\ \ \mathsf{EG}\,(U). \quad (2)$$

The last expression means that in at least one path, U will eventually be permanently true. This implies that the non-diagnosable system can continue forever in an uncertain cycle.

5. Abstraction

In this section, we outline the concept of abstraction-based diagnosability for modular systems. Since up to step 4 of the diagnosability Algorithm 1 is performed before applying abstraction in each module, the necessary information for diagnosability verification is preserved. Step 5 in Algorithm 1, is the most burdensome part. Thus, to overcome the problem of computational complexity, abstraction technique preserving loops, called *branching bisimulation with explicit divergence* ($\approx^d$) is presented.

Here, the abstraction will be applied on the verifier G_i, $i \in \mathcal{I}$. At this step, each state has the information of previously traversed transitions, thanks to the failure labels. The next step is to find uncertain cycles in $G = \|_{i\in\mathcal{I}}\ G_i$. To convey the failure occurrence information to the whole system, state labels are considered in the model . In this section we encounter three different events; local τ events, which are the replacements of local events between two similar state labels, local c/u events which are the replacements of local events between two states with different labels as illustrated in Fig. 4, and also shared events. Note that c/u events are introduced when there is a transformation of state labels to events. The goal is to abstract the local τ transitions.

There is stuttering bisimulation which abstracts the equally labeled states and branching bisimulation which abstracts local events, and we want to combine them with each other. Moreover, we want to preserve CTL* property, and based on [17] branching bisimulation does that, by showing the equivalence between stuttering and branching bisimulation, and also showing that CTL* is based on state labels as in stuttering bisimulation. Therefore, our technique uses branching bisimulation that preserves state labels to the end. By our technique to include events for the state labels changes, we only abstract states with the same label, which is a key point in our abstraction. All the mentioned definitions are explained in the following. From now on we can abstract based on events.

5.1 Branching bisimulation

Here, we first describe the branching bisimulation and then we add the definition for the branching bisimulation with explicit divergence.

Definition 6 (Branching Bisimulation) *[7]* Let $G = (Q, \Sigma, \rightarrow, Q_0)$ be a finite automaton. A relation $\mathcal{R} \subseteq Q \times Q$ is called a branching bisimulation ($\approx$) if it is

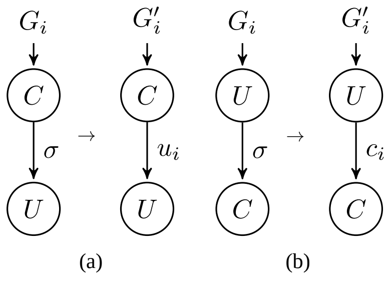


Figure 4 The relabeling of transitions of verifiers during abstraction.

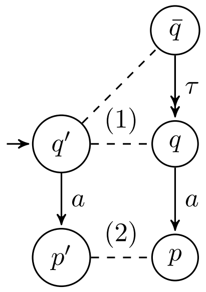


Figure 5 Branching bisimulation with τ

symmetric and satisfies the following transfer property: if $q'\mathcal{R}\bar{q}$ and $q' \xrightarrow{a} p'$, then either $a = \tau$ and $p'\mathcal{R}\bar{q}$, or $\exists q, p$ such that $\bar{q} \Rightarrow q \xrightarrow{a} p$, and $q'\mathcal{R}q$ and $p'\mathcal{R}p$. □

We consider G' as the abstracted model when all possible τ transitions have been removed. This means that G and G' are branching bisimilar when $G \approx G'$. We denote by $\Rightarrow$ reflexive-transitive binary closure and it indicates τ^*.

Now, the goal is to remove as many as possible τ transitions and the main algorithm for branching bisimulation is [9], but since this algorithm only works on systems that do not have cycles of silent events, here we use the distributed minimization algorithms proposed in [2], where there are signature-based algorithms. The algorithm works by successively refining the trivial partition, according to the local signature of the states with respect to the previous partition. Since the does not assume the absence of τ-cycles, we do not lose necessary information of the system. For more information on the algorithm please see [2].

For diagnosability verification we check whether there exist a loop over uncertain states or not. However, silent loops disappear in branching bisimulation and we need to keep these loops. For this reason we define another version of branching bisimulation that preserves silent loops.

Definition 7 ***(Branching Bisimulation with Explicit Divergence)****[7]* A relation $\mathcal{R} \subseteq Q \times Q$ is called a branching bisimulation with explicit divergence (BBED) ($\approx^d$) if it is a branching bisimulation and in addition satisfies the following condition for all states p, q:

If $p\mathcal{R}q$ and there is an infinite sequence of states $(p_k)_{k\in\Omega}$ such that $p = p_0, p_k \Rightarrow p_{k+1}$ and $p_k\mathcal{R}q, \forall k \in \Omega$, then there exists an infinite sequence of states $(q_\ell)_{\ell\in\Omega}$ such that $q = q_0$, $q_\ell \Rightarrow q_{\ell+1}$, $\forall \ell \in \Omega$, and $p_k\mathcal{R}q_\ell$, $\forall k, \ell \in \Omega$. □

Algorithmically every cycle is handled by adding a dummy state (div) to the model, which has a selfloop. There are also ingoing transitions from states belonging to loops, to div. All newly added transitions are labeled with τ. Then, the algorithm for branching bisimulation [2], is applied on the new model, as in Fig. 9. In the end, div and its corresponding transitions are removed and selfloops are added to all states connected to div.

5.2 Synchronization

The important fact that BBED is preserved by synchronization is shown in the following proposition.

Proposition 1 (BBED Synchronization) Let $G_i = (Q_i, \Sigma_i, \rightarrow_i, Q_{0i}, AP, L_i)$, $i = 1, 2$ denotes automata being bisimilar with explicit divergence and let $G'_i = (Q'_i, \Sigma'_i, \rightarrow'_i, Q'_{0i}, AP, L'_i)$, $i = 1, 2$ denotes these abstractions. Let $\mathcal{R}_i \subseteq Q_i \times Q'_i$ be a BBED for $(G_i, G'_i), i = 1, 2$. Then, the relation

$$\mathcal{R} = \{(\langle q_1, q_2\rangle, \langle q'_1, q'_2\rangle) | (q_1\mathcal{R}_1 q'_1) \wedge (q_2\mathcal{R}_2 q'_2)\}$$

is a BBED for $(G_1 \parallel G_2, G'_1 \parallel G'_2)$, i.e., $G_1 \approx^d G'_1$ and $G_2 \approx^d G'_2$, implies that $G_1 \| G_2 \approx^d G'_1 \parallel G'_2$.

Proof: Assume that there is a transition $\langle q_1, q_2\rangle \xrightarrow{a} \langle p_1, p_2\rangle$ in $G_1 \parallel G_2$. The following three event cases then need to be considered.

1. For $a \in \Sigma_s$. Since G_i is BBED there is a path $\bar{q}_i \Rightarrow q_i \xrightarrow{a}_i p_i$ in G_i, and a transition $q'_i \overset{a}{\rightarrow}'_i p'_i$ in G'_i where $(q'_i, q_i) \in \mathcal{R}_i$ and $(p'_i, p_i) \in \mathcal{R}_i$. Then, synchronization of G_1 and G_2 implies that if $(q'_1, q'_2)\mathcal{R}(\bar{q}_1, \bar{q}_2)$ and $\langle q'_1, q'_2\rangle \xrightarrow{a} \langle p'_1, p'_2\rangle$, $\exists q_1, q_2, p_1, p_2$ such that $(\bar{q}_1, \bar{q}_2) \Rightarrow (q_1, q_2) \xrightarrow{a} (p_1, p_2)$, where $(q'_1, q'_2)\mathcal{R}(q_1, q_2)$ and $(p'_1, p'_2)\mathcal{R}(p_1, p_2)$. Note that from $(\bar{q}_1, \bar{q}_2)$ to (q_1, q_2) there are mn interleavings between τ transitions, where m and n are the number of τ transitions in G_1 and G_2, respectively.

2. For $a \notin \Sigma_s \cup \{\tau\}$. In this case, there is an interleaving behavior between local non-τ-transitions, i.e., u/c transitions. By symmetry, and since G_1 is BBED, we may assume that $\bar{q}_1 \Rightarrow q_1 \xrightarrow{a}_1 p_1$, and G_2 stays in its current state. Since $q'_1\mathcal{R}_1 q_1$, there exists a transition $q'_1 \overset{a}{\rightarrow}'_1 p'_1$ in G'_1 with $p'_1\mathcal{R}_1 p_1$. If $(q'_1, q'_2)\mathcal{R}(\bar{q}_1, \bar{q}_2)$ and $\langle q'_1, q'_2\rangle \xrightarrow{a} \langle p'_1, q'_2\rangle$, then $\exists q_1, p_1$ such that $(\bar{q}_1, \bar{q}_2) \Rightarrow (q_1, \bar{q}_2) \xrightarrow{a} (p_1, \bar{q}_2)$, and $(q'_1, q'_2)\mathcal{R}(q_1, \bar{q}_2)$ and $(p'_1, q'_2)\mathcal{R}(p_1, \bar{q}_2)$.

3. For $a = \tau$. By symmetry, we may assume that a is in G_1, and G_2 stays in its current state. Argu-

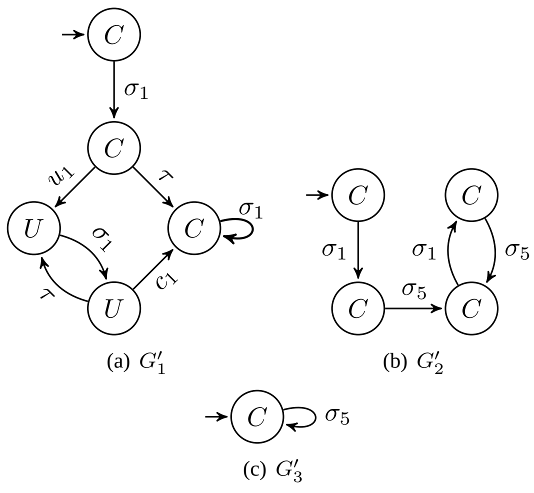


Figure 6 The G'_i, $i = 1, \dots, 3$ of Example 1.

ing in the same as in Case 2, if $(q'_1, q'_2)\mathcal{R}(\bar{q}_1, \bar{q}_2)$ and $\langle q'_1, q'_2\rangle \overset{a}{\rightarrow} \langle p'_1, q'_2\rangle$, then $(p'_1, q'_2)\mathcal{R}(\bar{q}_1, \bar{q}_2)$.

To conclude, for all three cases the synchronized system is BBED, i.e. $G_1 \| G_2 \approx^d G'_1 \parallel G'_2$. □

6 Compositional Abstraction

In ordinary modular abstraction, each module is abstracted once, and all abstracted modules are synchronized. Compositional abstraction means that after each synchronization of two modules, the abstraction is repeated. This implies normally a significant further state-space reduction as motivated below.

6.1 General Compositional Approach

A modular system consists of $G = G_1 \parallel \cdots \parallel G_m$. In the compositional algorithm of [5], the modular system G is abstracted step by step. Each automaton G_i is replaced by an abstracted version G'_i. Synchronous composition is computed step by step, and each intermediate result is abstracted again.

Eventually, the procedure leads to a single automaton G', the abstract description of the original system. Once G' is found, the final step is to use G' instead of the original system for diagnosability verification.

When abstracting an automaton G_i, in an attempt to replace it by G'_i, there will typically be some events used in G_i which do not appear in any other component $G_j, j \neq i$. These are called local events and are denoted as Σ^i_ℓ, and they are replaced by either τ (between similar state labels) or c/u (transition between states with different labels, see Fig. 4) in the abstraction. In other words, some events belong to a few modules, which after synchronization they become local events for the rest of modules, although they were not local from the beginning. In each iteration, more events become local which leads to more abstraction in comparison to merely abstracting all modules once in the beginning.

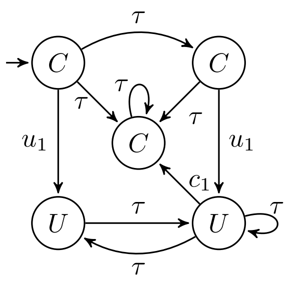


Figure 7 The $\widehat{G}$ of Example 1, where all σ_5 events are replaced by τ.

6.2 Diagnosability Verification of Compositional State-labeled Branching Bisimilar Automata

The ultimate goal is to determine if G satisfies CTL expression 1, where $G = \|_{i\in\mathcal{I}} G_i$. Each module G_i, includes C or U state labels.

Algorithm 2:

1. $\forall i \in \mathcal{I}$, follow the steps of 1 to 4 of Algorithm 1 and find G_i.

2. Based on events in Σ_s and Σ^i_ℓ, find the BBED of each G_i, denoted as G'_i. There are two possible cases regarding $a \in \Sigma^i_\ell$ as in the following.

   2.1. $q \overset{a}{\rightarrow} p$, $\ell_q \neq \ell_p$, replace a with ℓ_q and consider the index of the automata as the label index as in Fig. 4.

   2.2. $q \overset{a}{\rightarrow} p$, $\ell_q = \ell_p$, replace a with τ.

   After replacing $a \in \Sigma^i_\ell$ with the proper events, add dummy state div with its corresponding transitions, if there are cycles in G_i. Then, find G'_i. Each G'_i of Example 1 is depicted in Fig. 6.

3. Take two abstracted automata, G'_i and G'_j and find $\widehat{G}_{ij} = G'_i \parallel G'_j$.

4. Identify local events of $\widehat{G}_{ij}$ ($a \in \Sigma^{ij}_\ell$) and make a BBED of $\widehat{G}_{ij}$, denoted as $\widehat{G}'_{ij}$ as in Step 2.

5. Synchronize $\widehat{G}'_{ij}$ with the abstraction of the next automaton; $\widehat{G}_{ijk} = \widehat{G}'_{ij} \parallel G'_k$.

6. Continue Step 4 to 5, until all automata are synchronized and only one automaton remains. Fig. 7 shows the last $\widehat{G}$. Then find $\widehat{G}'$ as in Fig. 9.

7. Transform $\widehat{G}'$ from LTS to KS as in Fig. 10, and investigate the CTL expression (2) on it.

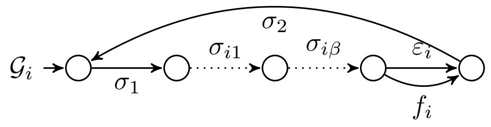


Figure 8 The considered automata of Example 3.

Table 1 Comparison of the verification methods introduced here and in [14] for the model depicted in Fig. 8.

		verifier in [14]		verifier after abstraction	
N	β	n_s	n_t	n_s	n_t
2	1	44	125	5	8
2	2	88	277	5	8
3	1	252	1672	5	8
3	2	616	4225	5	8

In Example 1, the infinite silent loops appears in the final monolithic automaton, see Fig. 8, where we find its BBED. Such infinite τ-loops may appear in any step of the algorithm, for which the same procedure applies.

Example 2 The $\widehat{G}'$ of the three automata of Example 1, is depicted in Fig. 10. It has 3 states and 5 transitions and is non-diagnosable due to a self-loop over U. □

The following example shows the efficiency of the diagnosability method introduced here in comparison to the method in [14].

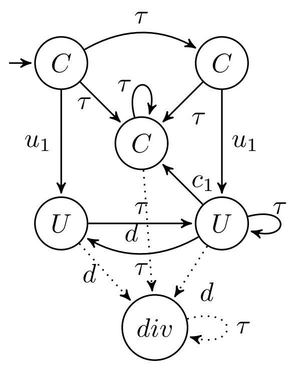
Example 3 Assume that there are N automata with $\Sigma_s = \{\sigma_1, \sigma_2\}$ and $\Sigma_\ell^i = \{\sigma_{i1}, \sigma_{i\beta}\}$, $i = 1, \ldots, N$ as depicted in Fig. 8. Table 1 shows the number of state n_s and transitions n_t of the verifier in [14] and the abstracted one in Algorithm 2. □

6.3 Abstraction-based Diagnosability

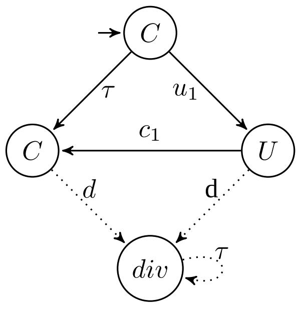
Finally, the correctness of the proposed diagnosability approach, Algorithm 2, is formulated in a theorem, saying that a model is diagnosable iff a corresponding BBED satisfies the CTL expression (1).

Theorem 2 (Diagnosability and $\approx^d$) The composed model $\mathcal{G} = \|_{i\in\mathcal{I}}\, \mathcal{G}_i$ is diagnosable, iff G' satisfies the CTL expression $\mathsf{AG}\ \mathsf{AF}(C)$.

Proof: According to [12], $\mathcal{G}$ is diagnosable iff $G = \|_{i\in\mathcal{I}}\, G_i$ satisfies the CTL expression (1). Then, since a G' is constructed incrementally combining BBED and synchronization, and synchronization, according to Proposition 1, is also BBED, we find that $G \approx^d G'$. The CTL expression (1) is based on state labels, but according to [1], Chapter 7, there is a strong bisimulation transforming between event based and state based labeling. Combing this bisimulation with BBED, and since BBED preserves CTL*-X, the temporal logic expression (1) is satisfied for G iff it is satisfied also for G'. □



(a) $\widehat{G}$ with dummy event div



(b) $\widehat{G}'$ before removing div

Figure 9 (a) Shows adding dummy state div to $\widehat{G}$ of Example 1 to find BBED. (b) Shows $\widehat{G}'$ before removing div.

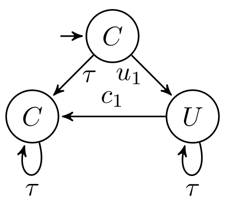


Figure 10 Evaluate the CTL expression (2), which is the same as identifying if there are any silent loops on the uncertain states.

7. Conclusion

This paper developed an efficient diagnosability verification technique based on a general abstraction approach. We exploited BBED, which preserves the main features for diagnosability verification, i.e., divergence sensitivity keeps loops in the system and does not abstract them away. Moreover, BBED is equivalent to CTL*-X, which means that model checking can be used to verify the abstracted model. As we showed, using compositional abstraction for modular diagnosability verification adds more efficiency to the approach. It avoids state space explosion by offering a significant reduction in comparison to the state-of-the-art techniques. Future work includes applying BBED and compositional abstraction technique for diagnosability verification of realistic industrial size systems.

References

[1] C. Baier and J. P. Katoen. *Principles of model checking*. Kluwer, Cambridge, MA, 2008.

[2] S. Blom and S. Orzan. Distributed branching bisimulation reduction of state spaces. *Electronic Notes in Theoretical Computer Science*, 89:32–46, 2003.

[3] C. Cassandras and S. Lafortune. *Introduction to discrete event models*. MIT Press, Norwell, MA, 1999.

[4] O. Contant, S. Lafortune, and D. Teneketzis. Diagnosability of discrete event systems with modular structure. *Discrete Event Dyn Sys*, 16:9–37, 2006.

[5] H. Flordal and R. Malik. Compositional verification in supervisory control. *Society for Industrial and Applied Mathematics*, 48(3):1914–1938, 2009.

[6] R. J. V. Glabbeek. The linear time - branching time spectrum II. the semantics of sequential systems with silent moves. *Proc. CONCUR93, Lecture Notes in Computer Science*, 43(3):555–600, 1993.

[7] R. J. V. Glabbeek, B. Luttik, and N. Trcka. Branching bisimilarity with explicit divergence. *Fundamenta Informaticae*, 93(4):371–392, 2009.

[8] R. J. V. Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the Association for Computing Machinary*, 715:66–81, 1996.

[9] J. F. Groote and F. W. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. *Automata, Languages and Programmig, ser. LNCS, M.S. Paterson, Ed.*, 443(4):626–638, 1990.

[10] S. Haar. Types of asynchronous diagnosability and the reveals-relation in occurrence nets. *IEEE Transactions on Automatic Control*, 55:2310–2320, 2010.

[11] S. Hashtrudi-Zad, R. Kwong, and W. Wonham. Fault diagnosis in discrete-event systems: framework and model reduction. *IEEE Transactions on Automatic Control*, 48:1199–1212, 2003.

[12] Z. Huang, S. Bhattacharyya, V. Chandra, S. Jiang, and R. Kumar. Diagnosis of discrete-event systems in rules-based model using first-order linear temporal logic. *Proceedings of American Control Conference*, pages 5114–5119, 2004.

[13] M. Huth and M. Ryan. *Logic in computer science, modelling and reasoning about systems*. Cambridge University Press, 2009.

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

[15] D. Lawesson. *An approach to post mortem diagnosability analysis for interacting finite state systems*. Doctoral Thesis, Department of Computer and Information Science, Linkoping University, 2005.

[16] R. Milner. *Communication and concurrency, Series in Computer Science*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1989.

[17] R. D. Nicola and F. Vaandrager. Three logics for branching bisimulation. *Journal of the Association for Computing Machinary*, 42(2):458–487, 1995.

[18] M. Noori-Hosseini. Diagnosis of discrete event systems. *Master's thesis, Signals and systems Dep., Chalmers University of Technology, Gothenburg, Sweden*, 2011.

[19] W. Qiu and R. Kumar. Decentralized failure diagnosis of discrete event systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 36(2):384–395, 2006.

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

[21] K. W. Schmidt. Abstraction-based failure diagnosis for discrete event systems. *Systems & Control Letters*, 46(9):1489–1494, 2010.

[22] K. W. Schmidt. verification of modular diagnosability with local specifications for discrete-event systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(5):1130–1140, 2013.

[23] T. Yoo and H. E. Garcia. Diagnosis of behaviors of interest in partially-observed discrete-event systems. *Systems & Control Letters*, 57:1023–1029, 2008.

[24] T. Yoo and S. Lafortune. Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Transactions on Automatic Control*, 47:1491–1495, 2002.

[25] C. Zhou, R. Kumar, and R. S. Sreenivas. Decentralized modular diagnosis of concurrent discrete event systems. *Proceedings of the 9th International Workshop on Discrete Event Systems*, pages 388–393, 2008.