Diagnosability Verification Using Compositional Branching Bisimulation

Diagnosability Verification Using Compositional Branching Bisimulation

Mona Noori-Hosseini, Bengt Lennartson

Abstract— **This paper presents an efficient diagnosability verification technique, based on a general abstraction approach. More specifically, branching bisimulation including state labels with explicit divergence (BBSD) is defined. This bisimulation preserves the temporal logic property that verifies diagnosability. Based on a proposed BBSD algorithm, compositional abstraction for modular diagnosability verification is shown to offer a significant state space reduction in comparison to state-of-the-art techniques. This is illustrated by verifying non-diagnosability analytically for a set of synchronized components, where the abstracted solution is independent of the number of components and the number of observable events.**

I. Introduction

Failure is the execution of a behavior that violates the nominal behaviors specification. The task of identifying deviations from desired behavior is called failure diagnosis and the ability to deduce previously occurred failures within a bounded number of observations is called diagnosability [1]. A system is diagnosable if each failure can be identified through a number of events in partial observations.

There are two approaches, namely, language specification [2], and failure events [3], 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. However, although polynomial time algorithms exist, 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 used for both automata and Petri net models [2], [4], including techniques for modular systems. In [2], 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 specifications. In some other works, event-based abstraction techniques are exploited, which are behaviorally equivalent to the original model. One of the most well known equivalences is weak bisimulation [6]. Another slightly more restricted one is *branching bisimulation* (BB), [7]. Unlike weak bisimulation, BB 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.

BB is an abstraction for labeled transition systems (LTSs) where transitions are labeled by events, also called actions in model checking [6]. LTSs correspond to ordinary automata in the discrete event community. A similar abstraction on systems with state labels, Kripke structures (KSs), is called *stuttering bisimulation* (SB) [8], [9]. Both BB and SB have 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* [6], except for the next operator X, called CTL*-X, is preserved for BB and SB [9].

For diagnosability verification based on a modular formulation, the resulting transition models are a combination of the two previously mentioned models, which include both state and event labels. We simply denote such systems as transition systems (TSs), and a bisimulation corresponding to BB and SB has also been formulated in [11], called visible bisimulation. However, their TS model involves a crucial restriction that a silent event also means that corresponding source and target state labels must be equal. For diagnosability verification that assumption is not always satisfied. Thus, in this paper no such restrictions are introduced and a corresponding bisimulation is defined, called *branching bisimulation including state labels and explicit divergence* (BBSD). Explicit divergence means that silent loops are not removed in the BBSD abstraction. This naming emphasizes the close relation between BB and BBSD. A simple state reduction algorithm is also introduced based on a generalization of the distributed BB reduction algorithm in [18].

In the proposed diagnosability algorithm, relevant failure information is handled by introducing specific state labels. Exploiting BBSD, a compositional event-based abstraction technique is then applied, while preserving state label information and silent loops. This means that local events are abstracted by BBSD, where the synchronization of components step by step generates more and more local events. This type of abstraction can be traced back to [10], but applied to local events it was more recently proposed in [21]. Note that when all components are synchronized all events are local, and for the resulting abstracted model a CTL based model checking is performed to identify if there are any

M. Noori-Hosseini and B. Lennartson are with Faculty of Automatic control, Automation and Mechatronics, Chalmers University of Technology, SE-412 96, Gothenburg, Sweden.

uncertain loops corresponding to a non-diagnosable system. The relation between diagnosability verification and temporal logic was proposed in [15].

A preliminary version of this work was presented as a work in progress paper in [12], without any proofs and algorithmic aspects on BBSD. The contribution of this paper is that BBSD is strictly defined, including a simple reduction algorithm, and then applied to compositional 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 approaches, where all transitions with the same event must be considered for abstraction, transitions with the same event are here abstracted individually, once again resulting in more efficient abstractions. Finally, observe that the proposed compositional approach can be applied to any verification problem that can be formulated as a CTL*-X expression on a system, composed of a number of synchronized components.

The whole verification concept is illustrated by a nontrivial example, including an arbitrary number of synchronized components. An analytical solution is obtained based on the proposed verification procedure, independent of the number of components and the number of observable events.

After some preliminaries, diagnosability for discrete event systems is presented in Section III. This is followed by abstraction based on BBSD in Section IV, applied to compositional abstraction and diagnosability verification in Section V.

II. Preliminaries

The event observation projection is a mapping from the original event set Σ to an observable event set $\Sigma^o \subseteq \Sigma$, i.e., $P : \Sigma \to \Sigma^o \cup \{\varepsilon\}$, 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 $\sigma \in \Sigma^u$, where Σ^u is the unobservable events set. Moreover, $\Sigma = \Sigma^o \dot\cup \Sigma^u$, and $\Sigma^o = \Sigma^s \dot\cup \Sigma^\ell$, where Σ^s is the set of shared events that are involved in more than one component, and Σ^ℓ is the set of local events that only belong to one component. $\Sigma^u = \Sigma^f \dot\cup \Sigma^n$, where Σ^f and Σ^n are failure and non-failure unobservable events, respectively. For our compositional BBSD, the Kripke structure, transition system and synchronization are now defined.

Definition 1 (Kripke Structure): [9] Let Λ be a set of state labels. A *Kripke Structure* (KS) is a triple $K = \langle Q, T, Q^0, \Lambda, L\rangle$, where Q is a set of states, $T \subseteq Q \times Q$ is the transition relation, Q^0 denotes the set of initial states, and $L : Q \to 2^\Lambda$ represents the state labeling. An element $(q, p) \in T$, usually written as $q \to p$, is called a transition. □

Based on the KS, we extend automata to include state labels in G.

Definition 2 (Transition System): A *transition system* (TS) is a tuple $G = \langle Q, \Sigma, T, Q^0, \Lambda, L\rangle$, where Σ is a set of events, $T \subseteq Q \times \Sigma \times Q$ is a transition relation and $(q, \sigma, p) \in T$, which is also denoted by $q \xrightarrow{\sigma} p$. □

Before the synchronous composition of two TSs is defined, the silent event τ is introduced. When local events are hidden, their event label is replaced by τ. A Kripke structure can therefore be considered as a TS where all event labels are equal τ. In the following synchronous composition, τ events are handled in the same way as local events, although silent τ events can be expected in both G_1 and G_2 that are now synchronized.

Definition 3 (Synchronous Composition): Let $G_i = \langle Q_i, \Sigma_i, T_i, Q_i^0, \Lambda_i, L_i\rangle$ for $i = 1, 2$ be two TSs. The synchronous composition of G_1 and G_2 is defined as $G_1 \parallel G_2 = \langle Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, T, Q_1^0 \times Q_2^0, \Lambda_1 \cup \Lambda_2, L\rangle$ where

$$(q_1, q_2) \xrightarrow{\sigma} (p_1, p_2) : \sigma \in (\Sigma_1 \cap \Sigma_2) \setminus \{\tau\}, q_1 \xrightarrow{\sigma}_1 p_1, q_2 \xrightarrow{\sigma}_2 p_2,$$
$$(q_1, q_2) \xrightarrow{\sigma} (p_1, q_2) : \sigma \in (\Sigma_1 \backslash \Sigma_2) \cup \{\tau\}, q_1 \xrightarrow{\sigma}_1 p_1,$$
$$(q_1, q_2) \xrightarrow{\sigma} (q_1, p_2) : \sigma \in (\Sigma_2 \backslash \Sigma_1) \cup \{\tau\}, q_2 \xrightarrow{\sigma}_2 p_2,$$

and $L : Q_1 \times Q_2 \to 2^{\Lambda_1 \cup \Lambda_2}$. □

III. Diagnosability of Discrete Event Systems

In this section, the notion of diagnosability is defined, along with a polynomial algorithm for diagnosability verification of modular systems.

Definition 4 (Failure Assignment Function): A failure assignment function is a mapping from Σ to state failure labels N or F, i.e., $\psi : \Sigma \to \{F, N\}$. It means that if $\sigma \notin \Sigma^f$, it is projected to N, otherwise it is projected to F. All reachable states after an F-labeled state, keep it as their label. □

Note that, the diagnosability of a system does not imply that failures belonging to the same class can be distinguished. It means that if one or more transitions in a specific failure class are executed, after a finite number of observations we are able to establish that at least one transition of that class has executed. For the sake of simplicity, here one failure class is considered. Let the set of all traces generated by G be denoted by $\mathcal{L}(G)$, also assume failures are permanent. Formally, diagnosability is then defined as follows.

Definition 5 (Diagnosability): With respect to P and ψ, a system G is diagnosable if

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

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. □

A. Diagnosability Verification Algorithm

For diagnosability verification in a modular system, assume that the system model G consists of n components $G_1, \ldots, G_n$ synchronized according to Def. 3. Thus $G = \|_{i \in I_n} G_i$, where $I_n = \{1, \ldots, n\}$. All unobservable

events Σ_i^u in the individual components are assumed to be local. Also, faults are not distinguished by components.

The algorithm introduced in [14], verifies the diagnosability of G and is described in the following. According to Definition 4, augment the states of each G_i with failure labels from the set $\Lambda_i = \{N, F\}$, where the resulting TSs are denoted by G_i^F. Then construct the corresponding non-failure models G_i^N, where all failure states labeled by $\{F\}$ in G_i^F are removed. All unobservable events in Σ_i^u in G_i^N are also relabelled such that they are local in relation to G_i^F but also to all other components. Then, verify the existence of uncertain cycles, i.e., loops over states with state label $\{N, F\}$ in the verifier

$$G_v = G^N \parallel G^F \tag{1}$$

where $G^N = \|_{i \in I_n} G_i^N$ and $G^F = \|_{i \in I_n} G_i^F$. Such loops can be expressed by temporal logic, which will be explained below. If the model contains at least one uncertain cycle, it is not diagnosable. Also note that due to associative and commutative properties [13], the verifier can be rewritten as

$$G_v = (\|_{i \in I_n} G_i^N) \parallel (\|_{i \in I_n} G_i^F) = \|_{i \in I_n} (G_i^N \parallel G_i^F) \tag{2}$$

This reformulation is important, since each pair $G_i^N \parallel G_i^F$ may have a number of local events. The modular BBSD abstraction proposed in this paper may then generate a significant state space reduction, before synchronization with additional components is performed. This is further explained in Section V.

Also observe that after every synchronization in (2), the union between the state labels according to Def. 3, only results in the following two state labels

$$\{N\} \quad \text{and} \quad \{N, F\}$$

independent of the number of synchronizations. These sets are from now simply denoted by N and NF, respectively.

The complexity of the method in [14] is $\mathcal{O}(n_q^2(n_t - n_f))$, where n_q and n_t are the number of states and transitions respectively, and n_f is the number of failure transitions in G. It is claimed that this verifier has lower complexity than all other methods found in the literature. In this paper, we verify the diagnosability of (1) more efficiently applying BBSD technique. In [14] the coreachability of G_F is also performed before synchronization with G_N. Coreachability sometimes reduces the state space, but it is not applicable in our modular version. Thus, the coreachability procedure is not included in this paper, and in the final example it does not influence the result, since all states in G_F are then coreachable.

If all components in a modular structure are diagnosable, the total system is also diagnosable. Therefore, we assume that at least one component is non-diagnosable when the diagnosability of synchronized components is evaluated.

B. Temporal Logic

In [15] it is pointed out that the existence of uncertain cycles in a verifier can be formulated as a model checking problem, where a temporal logic expression is verified [16]. In our case, the system G is non-diagnosable if the following CTL expression, cf. [17]

$$\texttt{EFEG}\ (NF) \tag{3}$$

is satisfied for the verifier G_v in (1), where NF is the failure state label in G_v. This implies that at least one path in G_v will eventually permanently have the state label NF. Thus, this path will continue forever in an uncertain cycle. Note that, the negation of (3), i.e., $\texttt{AGAF}\ (N)$, does not hold for diagnosability verification of G_v in (1), due to the inclusion of empty set in the definition of A. In other words, using A quantifier, CTL can not distinguish between two similar (terminating and non-terminating) behaviors. However, the main goal in diagnosability verification is to ensure there is no non-terminating uncertain behavior.

IV. Bisimilar Abstractions

In this section a state reduction method for transition systems including both event and state labels is presented. It is based on BB, where first crucial event label information such as failure events is introduced as state labels, cf. G_i^F in (2). Local events, both observable and unobservable, including failure events, are then replaced by silent τ events. As many as possible of the corresponding τ transitions are then removed, still preserving relevant state label information and alternative choices. The latter is important to able to trace individual branches also after the state and transition reduction. The states in the reduced system consists of blocks of states from the original system. These blocks partition the original state space, and an algorithm that computes the largest possible blocks of states is one of the main results of this section. But first the relevant bisimulations are presented.

A. Branching Bisimulation Including State Labels

The notion of BB [5] is now generalized by adding state labels. A repeated number of silent τ events $\xrightarrow{\tau^*}$ is also denoted by the arrow $\twoheadrightarrow$.

Definition 6 (BB including state labels): Let $G = \langle Q, \Sigma, T, Q^0, \Lambda, L\rangle$ be a finite TS. A relation $\mathcal{R} \subseteq Q \times Q$ is a *branching bisimulation including state labels* ($\approx$) if it is symmetric and satisfies the following conditions for all states $q', q \in Q$. As depicted in Fig. 1(a), if $q' \mathcal{R} q$, $q' \xrightarrow{\sigma} p'$, $L(q') = L(q)$, $L(q_i) = L(q)$ for $0 \leqslant i \leqslant n$, and $L(p') = L(p)$, then there exist states q_n, p such that $q \twoheadrightarrow q_n \xrightarrow{\sigma} p$, $q' \mathcal{R} q_i$ for $0 \leqslant i \leqslant n$, and $p' \mathcal{R} p$. Furthermore, if $\sigma = \tau$ and $L(p') = L(q')$, then also $p' R q$. □

Note that, if $\sigma = \tau$ and $L(q') \neq L(p')$, as is depicted in Fig. 1(b), it implies SB. If $\sigma \neq \tau$ and $L(q') = L(p')$, the result is BB, as shown in Fig. 1(c). Moreover, if $\sigma \neq \tau$ and $L(q') \neq L(p')$, it implies the combination of SB and BB, here called BB including state labels and illustrated in Fig. 1(d). Indeed, this combination is also included in visible bisimulation [11], but with the restriction that a τ transition $q \xrightarrow{\tau} p$ implies $L(p) = L(q)$. For diagnosability verification, it is crucial to allow $L(p) \neq L(q)$ when the silent event is the first failure event, in which case $L(q) = \{N\}$, while

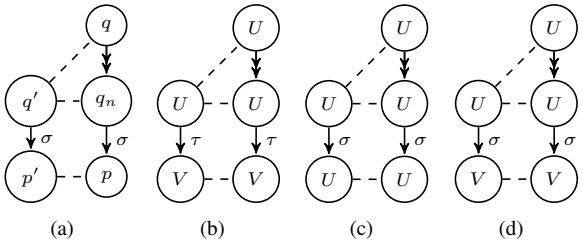

Fig. 1 BB including state labels. The first figure shows the state names, but in the rest of the figures they are replaced by state labels. 1(a) shows a path in both G' and G, cf. Def. 6, 1(b) shows an SB ($\sigma = \tau$ and $L(q') \neq L(p')$), 1(c) shows a BB ($\sigma \neq \tau$ and $L(q') = L(p')$), and 1(d) shows the combined notion of SB and BB ($\sigma \neq \tau$ and $L(q') \neq L(p')$).

$L(p) = \{F\}$. The TS and corresponding bisimulation BBSD introduced in this paper include this important behavior.

In the verification of diagnosability, the existence of loops over uncertain states is checked. However, silent loops disappear in BB, and we need to keep these loops through the abstraction. Thus, we define another version of BB including state labels that also preserves silent loops. This definition follows directly from Def. 6 and the results in [5], [9].

Definition 7 (BBSD): Let $G = (Q, \Sigma, T, Q^0, \Lambda, L)$ be a finite TS. A relation $\mathcal{R} \subseteq Q \times Q$ is a *branching bisimulation including state labels with explicit divergence* (BBSD) ($\approx^d$), if it is symmetric, a BB including state labels according to Def. 6, and in addition satisfies the following conditions for all states $q', q \in Q$. If there is an infinite sequence of states $(q_i)_{i\in\Omega}$, $\Omega = \{i | i \geq 0\}$ such that $q = q_0, q_i \xrightarrow{\tau} q_{i+1}$, $L(q_i) = L(q)$ and $q' \mathcal{R} q_i$ for all $i \in \Omega$, then there exists a state p' such that $q' \xrightarrow{\tau} p'$ and $p' \mathcal{R} q_i$ for all $i \in \Omega$. □

Remark 1 (Implementation of BBSD): In order to preserve divergence according to Def. 7, algorithmically every silent loop is handled by adding a dummy state to the model, with a separate unique state label. There are also ingoing transitions from states belonging to silent loop, to the dummy state. All added transitions to the dummy state are labeled with τ. Then, Algorithm 1 in Table IV-B is applied. In the end, the dummy states and their corresponding transitions are removed and τ selfloops are added to all states that were connected to the dummy state. See Example 7.148 in [6] for more details. □

B. Generation of BBSD Partition

Now, assume that G includes some silent τ events, and G' is the BBSD abstracted version of G, where as many τ transitions as possible have been removed. The BBSD relation between G' and G is denoted by $G' \approx^d G$. In the abstracted model G', the states are partitioned into blocks of states, where the states in each block satisfy the BBSD relation $\mathcal{R}$ in Def. 7. For BBSD to be an equivalence relation it is required to obtain the largest possible blocks, called the coarsest partitioning of the states in G. These notions are now formally defined.

Definition 8 (Partition): A *partition* π_k of a set Q is the family of pairwise disjoint subsets $\pi_k = \{B_i | i \in I_{n_q}\}$, where each B_i is nonempty. Thus, $Q = \dot{\bigcup}_{i\in I_{n_q}} B_i$ and $B_i \cap B_j = \varnothing$ for all $i, j \in I_{n_q}$ such that $i \neq j$. A subset $B_i \in \pi_k$ is called a *block*. □

A partial ordering between different partitions is also introduced, by considering *refinement* as the order relation.

Definition 9 (Refinement): Consider two partitions π_k and π_{k+1} of a set Q. The partition π_{k+1} is then a *refinement* of π_k, if for every block $B^{k+1} \in \pi_{k+1}$ there is a block $B^k \in \pi_k$ such that $B^{k+1} \subseteq B^k$. The partition π_{k+1} then refines π_k, denoted by $\pi_{k+1} \preceq \pi_k$, and π_{k+1} is said to be *finer* than π_k, while π_k is said to be *coarser* than π_{k+1} (smaller blocks in π_{k+1} than π_k). □

For a TS G according to Def. 2 with state space Q, we are searching for an abstracted system G' with a partition π of the set Q such that the BBSD relation is satisfied. Given such a partition, each state $q \in Q$ belongs to a block $\pi(q)$. This gives the prerequisites to formally define the abstracted TS G'.

Proposition 2 (Abstracted Transition System): Let $G = \langle Q, \Sigma, T, Q^0, \Lambda, L\rangle$ and assume that a partition π satisfies the BBSD relation in Def. 7. Then $G' \approx^d G$, and $G' = \langle Q_\pi, \Sigma, T_\pi, Q^0_\pi, \Lambda, L\rangle$ where

$$
\begin{aligned}
&Q_\pi = \{\pi(q) | q \in Q\},\\
&T_\pi(q) = \{(\pi(q), \sigma, \pi(p)) \mid \sigma \in \Sigma \wedge \\
&\qquad (\exists q_n : q \twoheadrightarrow q_n \xrightarrow{\sigma} p) \wedge (\pi(q) \neq \pi(p) \vee \sigma \neq \tau)\}.\\
&Q^0_\pi = \{\pi(q^0) | q^0 \in Q^0\}.
\end{aligned}
\tag{4}
$$

□

We observe that every silent loop, introduced in Def. 7, and treated as in Remark 1, is preserved by Equation (4) which defines the set of block transition relations T_π according to Def. 6. The partition π, where the states in each block satisfy the BBSD relation, is from now called the *BBSD partition*. To compute π, first consider the partition π_0, where all states with the same state label are grouped into one block. Since states with different state labels do not satisfy the BBSD relation, we conclude that the requested BBSD partition $\pi \preceq \pi_0$, i.e. π is equal or finer than π_0. Assuming that $\pi \preceq \pi_0$ means that the state label conditions for BBSD are automatically satisfied. π is described as

$$\pi(q) = \{p \in Q \mid T_\pi(q) = T_\pi(p)\}, \tag{5}$$

The definition of $\pi(q)$ in (5) expresses the obvious demand on a block state that the transitions to other blocks must be the same for all individual states in the same block. If that condition is not satisfied, the corresponding block needs to be divided. This is done in a recursive way where, based on the current partition π_k, the actual block transitions in T_{π_k} are determined by (4), and an updated partition π_{k+1} is computed based on (5). This recursion with initial partition π_0, determined by the state labels, is iterated until a fix point $\pi_{k+1} = \pi_k$ is reached. This procedure is summarized in Algorithm 1 in Table IV-B.

Before it is shown that this algorithm generates the coarsest BBSD partition, an example illustrates how Algorithm 1 works. First, the algorithm requires a special handling of block states $\pi(q)$ with no outgoing transitions. Then, the corresponding transition relation is denoted by $(\pi(q), \tau, \perp)$.

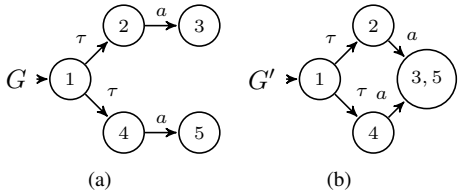

Fig. 2 Original model G and abstracted model G' in Example 1.

Example 1: In the automaton G in Fig. 2(a), the state labels for all states except state 4 are assumed to be equal, which results in the initial partition $\pi_0 = \{\pi_0^1, \pi_0^2\}$, where $\pi_0^1 = \{1, 2, 3, 5\}$ and $\pi_0^2 = \{4\}$. This generates the following initial block transitions $T_{\pi_0}(1) = \{(\pi_0^1, a, \pi_0^1), (\pi_0^1, \tau, \pi_0^2)\}$, $T_{\pi_0}(2) = \{(\pi_0^1, a, \pi_0^1)\}$, $T_{\pi_0}(3) = T_{\pi_0}(5) = \{(\pi_0^1, \tau, \perp)\}$ and $T_{\pi_0}(4) = \{(\pi_0^2, a, \pi_0^1)\}$.

Since $T_{\pi_0}(3) = T_{\pi_0}(5)$ the new block states are $\pi_1^1 = \{1\}$, $\pi_1^2 = \{2\}$, $\pi_1^3 = \{3, 5\}$, and $\pi_1^4 = \{4\}$, and the new block transition relations become $T_{\pi_1}(1) = \{(\pi_1^1, \tau, \pi_1^2), (\pi_1^1, \tau, \pi_1^4)\}$, $T_{\pi_1}(2) = \{(\pi_1^2, a, \pi_1^3)\}$, $T_{\pi_1}(3) = T_{\pi_1}(5) = \{(\pi_1^3, \tau, \perp)\}$ and $T_{\pi_1}(4) = \{(\pi_1^4, a, \pi_1^3)\}$. Based on these transition relations, the new partition $\pi_2 = \pi_1 = \{\{1\}, \{2\}, \{3, 5\}, \{4\}\}$, and the resulting abstracted model G' is shown in Fig. 2(b) □

The proposed algorithm has similarities with the signature algorithm by Blom and Orzan [18] for abstractions based on BB. Their signatures S_{π_k} correspond to the block transition relations in T_{π_k}, when the source blocks in T_{π_k} are neglected. For BBSD the source block is however crucial. When for instance the source block is neglected in $T_{\pi_1}(2) = \{(\pi_1^2, a, \pi_1^3)\}$ and $T_{\pi_1}(4) = \{(\pi_1^4, a, \pi_1^3)\}$ we obtain the equal signatures $S_{\pi_1}(2) = S_{\pi_1}(4) = \{(a, \pi_1^3)\}$. Thus, the signature algorithm generates the partition $\pi = \{\{1\}, \{2, 4\}, \{3, 5\}\}$, which means that also the states 2 and 4 are merged, which is wrong, because they have even different state labels. The signature algorithm is correct when the initial partition $\pi_0 = Q$, which is the case for BB.

However, the complexity of the signature algorithm and Algorithm 1 is the same, with a worst case complexity of $\mathcal{O}(n_q^2 n_t)$ in time and $\mathcal{O}(n_q n_t)$ in space, where n_q and n_t are the number of states and transitions, respectively. These algorithms are also easy to implement in a distributed version and the typical complexity is rather $\mathcal{O}(\log(n_q)(n_q + n_t))$ in time and $\mathcal{O}(n_q + n_t)$ in space. This is the same typical complexity as the most well known BB and SB algorithm by Groote and Vaandrager [19], which can also be adapted to BBSD, but is significantly more complex to understand and implement. As mentioned earlier, the coarsest BBSD partition is requested. The following theorem also shows that Algorithm 1 generates this result.

Theorem 3 (Coarsest BBSD partition): Algorithm 1 computes the coarsest BBSD partition π for the abstracted TS in Def. 2.

Proof: First we remind that $\pi \preceq \pi_0$, where π_0 is the initial partition for which all states with the same state label are grouped into one block. We will now prove that $\pi_{k+1} \preceq \pi_k$, which means that we need to show that any block $\pi_{k+1}(q) \in \pi_{k+1}$ does not include elements from any other block than $\pi_k(q) \in \pi_k$, since refinement means that $\pi_{k+1}(q) \subseteq \pi_k(q)$ for all states $q \in Q$.

Therefore, consider two arbitrary states $q_1, q_2 \in Q$ and corresponding block transitions $T_{\pi_k}(q_i) = \{(\pi_k(q_i), \sigma_i, \pi_k(p_i))\}$, for $i = 1, 2$. If $\pi_k(q_1) \neq \pi_k(q_2)$, then $T_{\pi_k}(q_1) \neq T_{\pi_k}(q_2)$ and, according to (5), $\pi_{k+1}(q_1) \neq \pi_{k+1}(q_2)$. Thus, in the next iteration no state will be included in $\pi_{k+1}(q)$ from any other block than $\pi_k(q)$, and therefore $\pi_{k+1}(q) \subseteq \pi_k(q)$. This implies that $\pi_{k+1} \preceq \pi_k$, and Tarski's famous fixed point theorem [20] says that this monotonic behavior results in the coarsest fixed point π, when the iteration starts with a π_0 that is not finer than the fixed point π. □

TABLE I Algorithm that computes the coarsest BBSD partition π based on (4) and (5).

Algorithm 1 Coarsest BBSD partition

```
1:  input Q, Σ, T, π_0
2:  π := π_0
3:  repeat
4:      π' := π
5:      for q ∈ Q
6:          T_π(q) := {(π(q), σ, π(p)) | σ ∈ Σ ∧
                    (∃q_n : q ↠ q_n -σ-> p) ∧ (π(q) ≠ π(p) ∨ σ ≠ τ)}
7:      endfor
8:      for q ∈ Q
9:          π(q) := {p ∈ Q | T_π(q) = T_π(p)}
10:     endfor
11: until π = π'
12: return π
```

V. Compositional Abstraction

The BBSD abstraction will now be applied to modular TSs $G = \|_{i \in I_n} G_i$ to reduce the state space before temporal logic properties are verified.

A. General Compositional Approach

In ordinary modular abstraction, each component is abstracted once, and all abstracted components are synchronized. However, compositional abstraction means that after each synchronization of two abstracted components, the abstraction is repeated on the result. This implies normally a significant further state-space reduction. In the compositional algorithm of [21], the modular system $G = \|_{i \in I_n} G_i$ is abstracted in this way. Each G_i is replaced by an abstracted version G_i'. Synchronous composition is computed step by step, and the choice of the next TS for the synchronization is made by some suitable heuristics such as maximal number of shared events between G_i and G_j. From now on, we assume that the components are given in an order $G_1, G_2, \ldots$ that

should be abstracted. Moreover, each intermediate result is abstracted again.

When abstracting a TS G_i, in an attempt to substitute it by G_i', there will typically be some events used in G_i which do not appear in any other component $G_j, j \neq i$. They are called local events (Σ_i^ℓ), and are replaced by τ. Some events belong to a few components, which after synchronization become local events compared to the rest of components, 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 components once in the beginning.

Eventually, the procedure leads to a single TS 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 verification of temporal logic expressions. Before this concept is applied to diagnosability verification, it is first shown that abstractions can be made before synchronization, which is the main tool to reduce computational complexity in a compositional approach.

B. Synchronization

The important fact that BBSD is preserved by synchronization is shown in the following proposition. Note that τ events, which are considered as local events, are interleaved in the synchronization.

Proposition 4 (BBSD Synchronization): Let $G_i = (Q_i, \Sigma_i, T_i, Q_i^0, \Lambda_i, L_i)$, $i = 1, 2$ be two TSs and $G_i' = (Q_i', \Sigma_i', T_i', Q_i'^0, \Lambda_i, L_i')$, $i = 1, 2$ be their abstractions. Let $\mathcal{R}_i \subseteq Q_i' \times Q_i$ be a BBSD 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 BBSD 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 $G_1 \| G_2 \approx^d G_1' \| G_2'$.

Proof: According to Def. 3, an event is either shared or local (including silent τ event) in the first or second component. Due to symmetry it is enough to consider a local event in the first component.

- Shared event $\sigma \in (\Sigma_1 \cap \Sigma_2) \setminus \{\tau\}$: Since G_i is BBSD, there is a path $q_i \twoheadrightarrow q_{i,n_i} \xrightarrow{\sigma}_i p_i$ in G_i including n_i silent τ transitions before σ, and a transition $q_i' \xrightarrow{\sigma'}_i p_i'$ in G_i', where $q_i' \mathcal{R}_i q_{i,j}$, $L_i(q_{i,j}) = L_i(q_i)$ for $j \in \mathcal{N}_{n_i}$ and $q_i = q_{i,0}$. Furthermore, $p_i' \mathcal{R}_i p_i$, and $L_i(p_i') = L_i(p_i)$. Then, synchronization of G_1 and G_2 implies that there is a path $(q_1, q_2) \twoheadrightarrow (q_{1,n_1}, q_{2,n_2}) \xrightarrow{\sigma} (p_1, p_2)$ in $G_1 \parallel G_2$, including $n_1 + n_2$ silent τ transitions before σ, and a transition $(q_1', q_2') \xrightarrow{\sigma} (p_1', p_2')$ in $G_1' \| G_2'$, where $(q_1', q_2') \mathcal{R} (q_{1,j}, q_{2,\ell})$, $L(q_{1,j}, q_{2,\ell}) = L(q_1, q_2)$ for $(j, \ell) \in \mathcal{N}_{n_1} \times \mathcal{N}_{n_2}$ and $(q_1, q_2) = (q_{1,0}, q_{2,0})$. Furthermore, $(p_1', p_2') \mathcal{R} (p_1, p_2)$, and $L(p_1', p_2') = L(p_1, p_2)$.
- Local event $\sigma \in (\Sigma_1 \setminus \Sigma_2) \cup \{\tau\}$: First, if $\sigma = \tau$, G_2 is assumed to stay in its current state. Since G_1 is BBSD, there is a path $q_1 \twoheadrightarrow q_{1,n_1} \xrightarrow{\sigma}_1 p_1$ in G_1 including n_1 silent τ transitions before σ, and a transition $q_1' \xrightarrow{\sigma'}_1 p_1'$ in G_1', where $q_1' \mathcal{R}_1 q_{1,j}$, $L_1(q_{1,j}) = L_1(q_1)$ for $j \in \mathcal{N}_{n_1}$ and $q_1 = q_{1,0}$, $p_1' \mathcal{R}_1 p_1$, and $L_1(p_1') = L_1(p_1)$. Furthermore, if $\sigma = \tau$ and $L_1(p_1') = L_1(q_1')$, then also $p_1' \mathcal{R} q_1$. Synchronization of G_1 and G_2 then implies that there is a path $(q_1, q_2) \twoheadrightarrow (q_{1,n_1}, q_2) \xrightarrow{\sigma} (p_1, q_2)$ in $G_1 \parallel G_2$, including n_1 silent τ transitions before σ, and a transition $(q_1', q_2') \xrightarrow{\sigma} (p_1', q_2')$ in $G_1' \parallel G_2'$, where $(q_1', q_2') \mathcal{R} (q_{1,j}, q_2)$, $L(q_{1,j}, q_2) = L(q_1, q_2)$ for $j \in \mathcal{N}_{n_1}$ and $q_1 = q_{1,0}$, $(p_1', q_2') \mathcal{R} (p_1, q_2)$, and $L(p_1', q_2') = L(p_1, q_2)$. Furthermore, if $\sigma = \tau$ and $L(p_1') = L(q_1')$, then also $L(p_1', q_2') = L(q_1, q_2)$.

To summarize, for both cases the synchronized system is BBSD, i.e. $G_1 \| G_2 \approx^d G_1' \parallel G_2'$. □

C. Diagnosability Verification by Compositional BBSD Abstraction

The proposed compositional abstraction approach based on TSs with both state and event labels is now applied to verification of diagnosability. Consider the verifier in (2) that will be abstracted by repeatedly applying the BBSD partitioning in Algorithm 1. iff the CTL expression (3) is satisfied for the abstracted one G_v', the original system G is diagnosable.

The abstracted verifier is generated by the following recursive formula, where the hiding of local events by silent τ events is considered as an initial step of the abstraction operator (the prime operator).

$$G_v^i = G_v^{i-1\prime} \parallel \left(G_i^{N\prime} \parallel G_i^{F\prime} \right)' \tag{6}$$

where the initial value is $G_v^1 = G_1^{N\prime} \parallel G_1^{F\prime}$, and the final verifier for n components is $G_v{}' = G_v^{n\prime}$ The following example illustrates the proposed verification method.

Example 2: Consider the system in Fig. 3, where $n_b = n$. Hence an increased number of components n also means that the number of observable but local events $n_b - 1$ increases. Note that the events a and d are shared by all components, while the events c_{i-1}, c_i are shared with the closest neighbor, except for c_0 and c_n that are local in the first and the last components, and thus abstracted. All b events $\{b_i^0, \ldots, b_i^{n_b}\}$ are local but observable, while the failure event f_i and u_i are both local and unobservable. Note that all b events are local with respect to other automata but they are shared between G_i^N and G_i^F in each automaton, thus they are not abstracted yet in Fig. 3(b) and Fig. 3(c).

Table 1 shows the number of states n_q and transitions n_t for the non-abstracted verifier G_v, and the abstracted verifier G_v' for $n = 2, 3, 4$. Different partial results of our algorithm are shown in Fig. 3. Note that local events in each step are replaced by τ, before each BBSD abstraction. As it is shown,

TABLE II Comparison of the verifier G_v [14] and its abstracted version G_v' for the model depicted in Fig. 3(a).

		G_v from [14]		G_v'	
n	n_b	n_q	n_t	n_q	n_t
2	2	64	166	2	3
3	3	298	1180	2	3
4	4	1364	7346	2	3

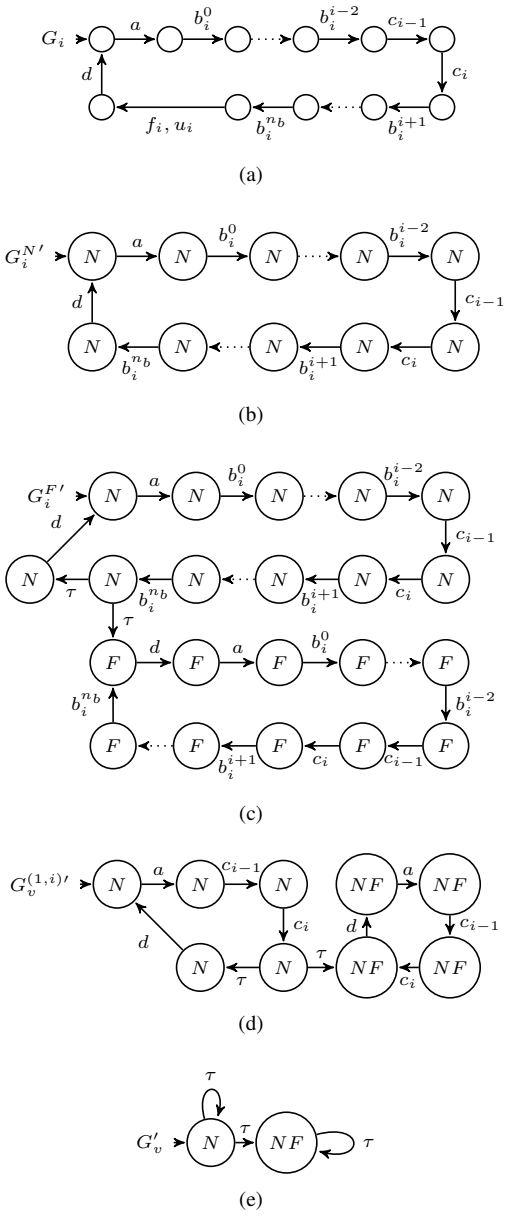

Fig. 3 The components G_i and partial results in the computation of the verifier G'_v in Example 2. Note that, in 3(a), $i > 1$ and $i < n_b$ for the upper and lower traces, respectively.

the system is not diagnosable, due to a loop over a state including the uncertain NF label. □

This example, although not completely realistic, was formulated to illustrate the different parts and properties of the algorithm. Note that local observable events are abstracted as well as events that are only shared by some components. Here we remind that existing abstraction techniques have limitations concerning both these aspects.

As a final result, the correctness of the proposed diagnosability approach is formulated in a theorem, saying that a model is diagnosable iff a corresponding BBSD abstraction satisfies the CTL expression in (3).

Theorem 5 (Diagnosability and $\approx^d$): The composed model $G = \|_{i \in I_n} G_i$ is non-diagnosable, iff the abstracted verifier G'_v generated by (6) satisfies the CTL expression $\texttt{EFEG}\ (NF)$.

Proof: According to [14], [15], G is non-diagnosable iff the verifier $G_v = G^N \parallel G^F$ satisfies the CTL expression $\texttt{EFEG}\ (NF)$. Based on the reformulation of (2) and the generation of the BBSD abstraction G'_v according to (6), we observe that $G_v \approx^d G'_v$, since the synchronization according to Proposition 4 preserves the BBSD abstraction. Moreover, since BBSD preserves CTL*-X properties, the expression in (3) is satisfied for G_v iff it is satisfied also for G'_v. □

VI. Conclusions

An efficient diagnosability verification technique has been formulated in this paper, based on a general abstraction approach. A modified and more general bisimilarity called BBSD has been introduced, including an efficient abstraction algorithm. Since BBSD preserves CTL*-X expressions and diagnosability can be expressed as a CTL formula, this abstraction is used in a compositional framework, which is shown to give significant state space reduction for diagnosability verification. The proposed method is general and can be used to verify any CTL*-X expression for a set of synchronized components, especially if the coupling between many components only include a few number of shared global events. An analytical example illustrates this behavior. Future work involves evaluation of industrial problems with high complexity.

References

[1] M. Sampath and R. Sengupta and S. Lafortune and K. Sinnamohideen and D. Teneketzis, Diagnosability of discrete-event systems, *IEEE Trans. Autom. Control*, vol. 40, 1995, pp. 1555-1575.

[2] K. W. Schmidt, Abstraction-based failure diagnosis for discrete event systems, *Systems & Control Letters*, vol. 59, 2010, pp. 42-47.

[3] S. Jiang and Z. Huang and V. Chandra and R. Kumar, A polynomial algorithm for testing diagnosability of discrete-event systems, *IEEE Trans. Autom. Control*, vol. 46, 2001, pp. 1318-1321.

[4] M.P. Cabasino and A. Giua and C. Seatzu, Diagnosability of discrete-event systems using labeled Petri nets, *IEEE Trans. Autom. Sci. Eng.*, vol. 1, 2014, pp. 144-153.

[5] R.J.Van Glabbeek and B. Luttik and N. Trcka, Branching bisimilarity with explicit divergence, *Fundam Inform*, vol. 93, 2009, pp. 371-392.

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

[7] R.J.Van Glabbeek and W.P. Weijland, Branching time and abstraction in bisimulation semantics, *J. of the ACM*, vol. 43, 1996, pp. 555-600.

[8] M.C. Browne and E.M. Clarke and O. Grumberg, Characterizing finite Kripke structures in propositional temporal logic, *J. Theor. Comput. Sci.*, vol. 59, 1988, pp. 115-131.

[9] R. De Nicola and F. Vaandrager, Three logics for branching bisimulation, *J. of the ACM*, vol. 42, 1995, pp. 458-487.

[10] S. Graf and B. Steffen and G. Luttgen, Compositional Minimization of Finite State Systems Using Interface Specifications, *Formal Aspects of Computing*, vol. 8, 1996, pp. 607-616.

[11] R. Gerth and R. Kuiper and D. Peled and W. Penczek, A partial order approach to branching time logic model checking, *J. of the ACM*, vol. 150, 1999, pp. 132-152.

[12] M. Noori-Hosseini and B. Lennartson, "Verification of diagnosability based on compositional branching bisimulation", *in 19th Conference on ETFA*, Barcelona, Spain, 2014.

[13] C.G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, Kluwer Academic Publishers, Norwell, MA; 1999.

[14] M.V. Moreira and T.C. Jesus and J.C. Basilio, Polynomial time verification of decentralized diagnosability of discrete event systems, *IEEE Trans. Autom. Control*, vol. 56, 2011, pp. 1679-1684.

[15] Z. Huang and S. Bhattacharyya and R. Kumar and S. Jiang and V. Chandra, Diagnosis of discrete-event systems in rules-based model using first-order linear temporal logic, *Asian J. Control*, 2008, pp. 1-9.

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

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

[18] S. Blom and S. Orzan, Distributed branching bisimulation reduction of state spaces, *Electron Notes Theor Comput Sci*, vol. 89, 2003, pp. 99-113.

[19] J.F. Groote and F.W. Vaandrager, An efficient algorithm for branching bisimulation and stuttering equivalence, *40th Int'l Coll. Automata, Languages, and Programming*, vol. 443, 1990, pp. 626-638.

[20] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, *Pacific J. Math*, vol. 5, 1955, pp. 285-309.

[21] H. Flordal and R. Malik, Compositional verification in supervisory control, *SIAM J. Control Optim*, vol. 48, 2009, pp. 1914-1936.