HAL
open science

Parallel Diagnosability Analysis with LTL-X Model Checking based on Petri Net Unfoldings

Laura Brandan-Briones, Agnes Madalinski, Hernán Ponce de León

To cite this version:

Laura Brandan-Briones, Agnes Madalinski, Hernán Ponce de León. Parallel Diagnosability Analysis with LTL-X Model Checking based on Petri Net Unfoldings. Workshop on Principles of Diagnosis, Sep 2014, Graz, Austria. 2013. hal-00915478v2

HAL Id: hal-00915478

https://inria.hal.science/hal-00915478v2

Submitted on 15 Dec 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed Diagnosability Analysis with Petri Nets *

L. Brandán-Briones[1] and **A. Madalinski**[2] and **H. Ponce-de-León**[3]
[1]CONICET and Fa.M.A.F. - Universidad Nacional de Córdoba, Argentina
[2]Facultad de Ciencias de la Ingeniería, Universidad Austral de Chile, Valdivia, Chile
[2]INRIA and LSV, École Normale Supérieure de Cachan and CNRS, France

Abstract

We propose a framework to distributed diagnosability analysis of concurrent systems modeled with Petri nets as a collection of components synchronizing on common observable transitions, where faults can occur in several components. The diagnosability analysis of the entire system is done in parallel by verifying the interaction of each component with the fault free versions of the other components. Furthermore, we use existing efficient methods and tools, in particular parallel LTL-X model checking based on unfoldings, for diagnosability verification.

1 Introduction

As systems become larger their behavior becomes more and more complex, consequently it becomes harder to detect faults. There are cases where faults cannot be ruled out at design stage since they intrinsically belong to the systems (they are inherent faults), or to the environment where the system is executed. Therefore, it becomes crucially important to have mechanisms in place to be able to detect and recover from such faults when they occur.

In the last years a lot of work have been done studying inherent faults: *Fault diagnosis* consists in detecting abnormal behaviors of a physical system. *Diagnosability* is the property that gives the possibility of detecting faults in a bounded time after they occur given a set of observations. If a system is diagnosable, it is always possible to determine if a fault has occurred by observing the system's behavior for a sufficiently long time, and then diagnosis can find possible explanations for the given sequence of observations. Otherwise there are scenarios in which it is impossible to tell whether a fault has occurred or not, no matter for how long the system is observed. Naturally, non-diagnosable systems usually indicate that the system should be augmented with additional sensors monitoring it.

A sound software engineering rule for building complex systems is to divide the whole system in smaller and simpler components, each performing a specific task. Moreover, they could be built by different groups of people or in different places. This means that, in general, complex systems are actually collections of simpler components running in parallel.

*This work has been supported by the European Union Sh Framework Program under grant agreement no. 295261 (MEALS), and the project SticAmSud No. 13STIC-04.

In this paper we propose a distributed diagnosability verification with LTL-X model checking based on Petri net unfoldings. We start modeling components as automata, but instead of making a composition of automata we consider the complete system as a Petri Net. Thus, taking the advantage of the compactness of the representation allowed by Petri nets compared to automata. Then, our system is modeled as a collection of components represented as Petri nets and synchronizing on common observable transitions. Also, we remove the assumption that a kind of fault can only occur in a single component (which is usually made in the diagnosability analysis of distributed systems), and allow the same kind of fault to occur in several components (moreover, we allow the same fault to occur in different components; an example of such a fault could be an electricity black out, which can happen in any component independently).

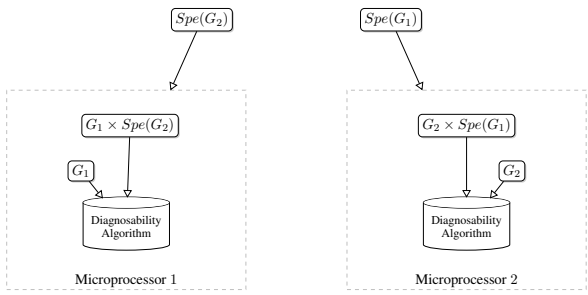


Figure 1: Distribution of the diagnosability analysis.

We distribute the diagnosability analysis (which is usually done interactively: the information from local components is combined until a global verdict is reached) as it is shown in Figure 1 where G_1, G_2 represent two components of the system. Suppose different groups are contracted to build different components of a system. Even if each component is diagnosable, it is not always the case that the resulting system has such property. We propose a framework where each component only shares with the others a fault free version of its own (e.g. the specification of its ideal behavior). Then, each component should not only be diagnosable, but also its interaction with the fault free version of the other components should be diagnosable. We prove that if that is the case then the complete system is diagnosable, resulting in a diagnosability analysis that is distributed.

Finally, we employ the efficient LTL-X model checking based on Petri net unfolding to verify the distributed diagnosability property. Not only we distribute the diagnosability verification but we are employing true concurrency (or

partial order) semantics to represent and to check the diagnosability property - which results in important memory savings since executions are considered as partially ordered sets of events rather than sequences.

Our approach extends the distributed diagnosability verification in the framework of automata [Ponce de León *et al.*, 2013]; and, it uses existing twin plant method deployed in [Madalinski and Khomenko, 2010] where PUNF [Khomenko, 2012] is applied to diagnosability verification.

The paper is organized as follows. First, we show related work in Section 2. Then, we present the formal model that we use for modeling the system together with basic notions about verification in Section 3. Section 4 introduces the notion of diagnosability and its verification, followed by our main contribution of this paper, the distributed diagnosability analysis, in Section 5. Finally, we conclude and discuss future work in Section 6.

2 Related Work

Diagnosability was initially developed in [Sampath *et al.*, 1995] under the setting of discrete event systems. In that paper, necessary and sufficient conditions for testing diagnosability are given. In order to test diagnosability, a special diagnoser is computed, whose complexity of construction is shown to be exponential in the number of states of the original system, and double exponential in the number of faults. Later, in [Jiang *et al.*, 2000], an improvement of this algorithm is presented, where the so-called twin plant method is introduced and shown to have polynomial complexity in the number of states and faults. None of the previous methods consider the problem when the system is composed of components working in parallel. An approach to this consideration is addressed in [Schumann and Pencolé, 2007; Debouk *et al.*, 2000; Pencolé, 2004; Schumann and Huang, 2008] where the diagnosability problem is performed by either local diagnosers or twin plants communicating with each other, directly or through a coordinator, and by that means pooling together the observations. [Ye and Dague, 2012] shows that, when considering only local observations, diagnosability becomes undecidable when the communication between component is unobservable. An algorithm is proposed to check a sufficient but not necessary condition of diagnosability. However, their results are based on the assumption that a fault can only occur in one of the components, an assumption that cannot always be made.

The state-based twin plant method usually suffers from the combinatorial *state space explosion* problem. That is, even a relatively small system specification can (and often does) yield a very large state space. To alleviate this problem Petri net *unfolding techniques* appear promising. The system is modeled as a Petri net, where each transition is labelled with the performed action. A *finite and complete prefix* of the unfolding gives a compact representation of all the state space. Executions are considered as partially ordered sets of transitions rather than sequences, which often results in memory savings. Since the introduction of the unfolding technique in [McMillan, 1992], it was improved [Esparza *et al.*, 2002], parallelized [Heljanko *et al.*, 2002], and applied to various practical applications such as distributed diagnosis [Fabre *et al.*, 2005] and LTL-X model checking [Esparza and Heljanko, 2001]. Also, the problem of diagnosability verification based on the twin plant method has been studied in [Madalinski and Khomenko, 2010] in the context of parallel LTL-X model checking based on Petri net unfoldings.

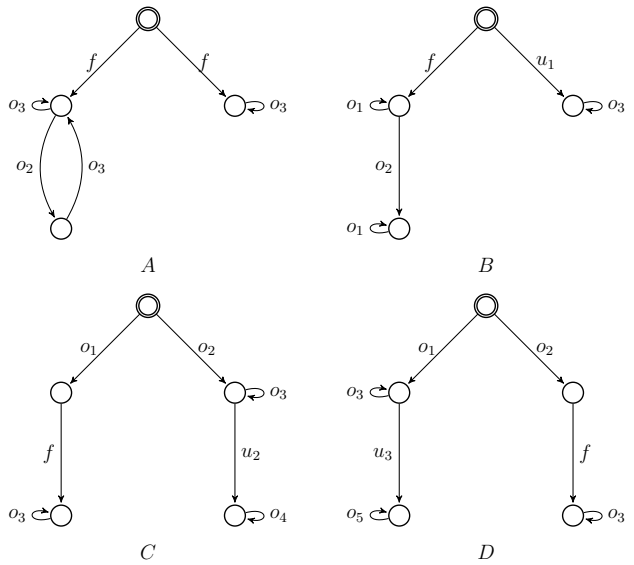


Figure 2: Components specifications as automata

3 Basic Notions

Model of the system. We consider distributed systems composed by several components that communicate with each other through their shared observable actions, as diagnosability is undecidable when communication is unobservable [Ye and Dague, 2012]. The local model of a component is defined as an automaton (Q, Σ, δ, q_0), where Q is a finite set of states, Σ is a finite set of actions, $\delta : Q \times \Sigma \to Q$ is the transition function and $q_0 \in Q$ is the initial state.

In diagnosability analysis, some of the actions of Σ are observable while the rest are unobservable. Thus, the set of actions Σ is partitioned as $\Sigma = \Sigma_o \uplus \Sigma_u$ where Σ_o represents the observable actions and Σ_u the unobservable ones. The faults to diagnose are considered unobservable, i.e. $\Sigma_F \subseteq \Sigma_u$, because faults that are observable can be easily diagnosed.

As usual in diagnosability analysis, we made the following assumptions about our systems.

Assumption 1. *We only consider (live) systems, where there is a transition defined at each state, i.e. the system cannot reach a point at which no action is possible.*

Assumption 2. *The system does not contain cycles of unobservable actions.*

Figure 2 shows four components modeled by automata A, B, C and D where $o_1, o_2, o_3, o_4, o_5 \in \Sigma_o$ and $u_1, u_2, u_3 \in \Sigma_u$. The special action $f \in \Sigma_F$ is the fault to be diagnosed.

The joint behavior of the system can be represented by a safe labelled Petri net. A *labelled net* is a tuple $N = (P, T, F, \lambda)$ where *(i)* $P \neq \emptyset$ is a set of *places*, *(ii)* $T \neq \emptyset$ is a set of *transitions* such that $P \cap T = \emptyset$, *(iii)* $F \subseteq (P \times T) \cup (T \times P)$ is a set of *flow arcs*, *(iv)* $\lambda : T \to \Sigma$ is a *labelling* function. A *marking* is a subset M of places, i.e. $M \subseteq P$. A *labelled Petri net* is a tuple $\mathcal{N} = (P, T, F, \lambda, M_0)$, where *(i)* (P, T, F, λ) is a finite labelled net, and *(ii)* $M_0 \subseteq P$ is an *initial marking*. Elements of $P \cup T$ are called the *nodes* of $\mathcal{N}$. For a transition $t \in T$, we call ${}^\bullet t = \{p \mid (p, t) \in F\}$ the *preset* of t, and $t^\bullet = \{p \mid (t, p) \in F\}$ the *postset* of t. In figures, we

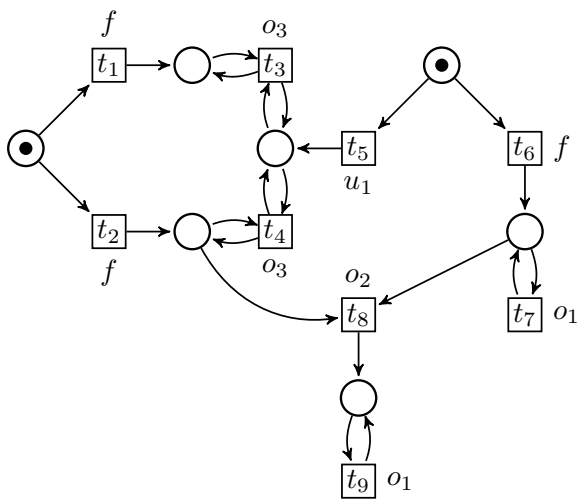
represent, as usual, places by empty circles; transitions by squares; F by arrows; and the marking of a place p by black tokens in p. A transition t is *enabled* in marking M, written $M \xrightarrow{t}$, if $\forall p \in {}^\bullet t,\ M(p) > 0$. This enabled transition can *fire*, resulting in a new marking $M' = M - {}^\bullet t + t^\bullet$. This firing relation is denoted by $M \xrightarrow{t} M'$. A *run* is a sequence $\rho = M_0 t_o M_1 t_1, \dots t_{n-1} M_n$ such that $M_0 \xrightarrow{t_0} M_1 \xrightarrow{t_1} \dots \xrightarrow{t_{n-1}} M_n$ and $\sigma = \lambda(t_0)\lambda(t_1)\dots\lambda(t_{n-1})$ is its associated *trace*, i.e trace$(\rho) = \sigma$. A marking M is *reachable* if there exists a run from M_0 to M. The set of markings reachable from M_0 is denoted $\mathbf{R}(M_0)$.

As our systems are live, we only consider infinite traces where the infinite repetition of action a is denoted by $\widehat{a}$. The set of runs and traces are denoted by $\mathrm{run}(\mathcal{N})$ and $\mathrm{traces}^\omega(\mathcal{N})$ respectively. As only some actions are observable, the observable projection of a trace is defined as

$$obs(\sigma) \quad = \quad \begin{cases} \epsilon & \text{if } \sigma = \epsilon \\ a{\cdot}obs(\sigma') & \text{if } \sigma = a{\cdot}\sigma' \wedge a \in \Sigma_o \\ obs(\sigma') & \text{if } \sigma = a{\cdot}\sigma' \wedge a \notin \Sigma_o \end{cases}$$

The translation from an automaton A to a labelled Petri net $\mathcal{N}_A$ is immediate: *(i)* places are the states of the automaton, i.e. $P = Q$; *(ii)* for every transition $(s_i, a, s_i') \in \delta$ we add t to T and set ${}^\bullet t = \{s_i\}, t^\bullet = \{s_i'\}$ and $\lambda(t) = a$; *(iii)* the initial state is the only place marked initially, i.e. $M_0 = \{q_0\}$.

The joint behavior of a system composed of automata $\{A_1, \dots, A_n\}$ is modeled by $\mathcal{N}_{A_1} \times \dots \times \mathcal{N}_{A_n}$ where $\times$ represents the product of labelled nets defined in [Winskel, 1985] synchronizing on shared observable transitions. Product of nets prevents us from the state explosion problem that usually arises in product of automata.

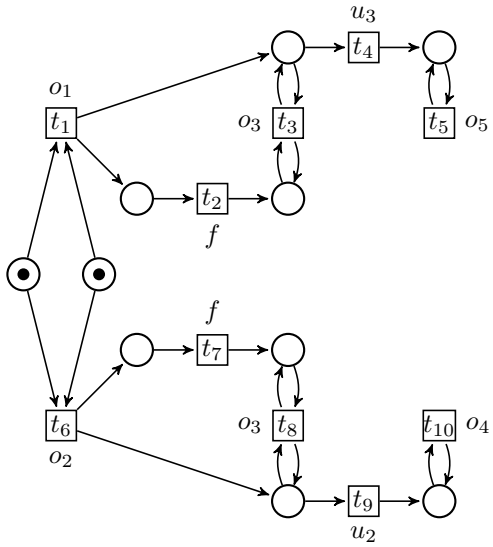


$\mathcal{N}_1 = \mathcal{N}_A \times \mathcal{N}_B$

Figure 3: Automata $\{A, B\}$ represented as Petri nets

The joint behavior of $\{A, B\}$ and $\{C, D\}$ from Figure 2 can be modeled by the corresponding Petri nets $\mathcal{N}_1 = \mathcal{N}_A \times \mathcal{N}_B$ and $\mathcal{N}_2 = \mathcal{N}_C \times \mathcal{N}_D$ of Figure 3 and Figure 4.

Consider the automata $\{A_1, \dots, A_n\}$ and its corresponding net $\mathcal{N}$. The projection of a run on component i is given



$\mathcal{N}_2 = \mathcal{N}_C \times \mathcal{N}_D$

Figure 4: Automata $\{C, D\}$ represented as Petri nets

by

$$P_i((q^1, \dots, q^n)) \;=\; q^i$$

$$P_i((q^1, \dots, q^n){\cdot}t{\cdot}\rho') \;=\; \begin{cases} q^i{\cdot}t{\cdot}P_i(\rho') & \text{if } \exists\, \delta^i(q^i, \lambda(t)) \\ P_i(\rho') & \text{otherwise} \end{cases}$$

For $\sigma \in \mathrm{traces}^\omega(\mathcal{N})$, we say that σ_i is its projection on component i, denoted $P_i(\sigma) = \sigma_i$, if and only if $\exists \rho \in \mathrm{trace}^{-1}(\sigma) : \mathrm{trace}(P_i(\rho)) = \sigma_i$.

Example 1. *Consider $\sigma = o_1 f o_3 u_3 \widehat{o}_5 \in \mathrm{traces}^\omega(\mathcal{N}_2)$. Its projection on components C and D are given by $P_C(\sigma) = o_1 f o_3$ and $P_D(\sigma) = o_1 o_3 u_3 \widehat{o}_5$. These projections are traces of the corresponding components C and D from Figure 2. Note that projections of an infinite trace from the net can be finite in one component.*

As the projection operator only erases actions in a trace, it is easy to see that every fault belonging to a trace of a component, also belongs to the trace of the net as it is shown by the following result.

Proposition 1. *Let $\mathcal{N} = \mathcal{N}_{A_1} \times \dots \times \mathcal{N}_{A_n}$, then for every $\sigma \in \mathrm{traces}^\omega(\mathcal{N})$ with $P_i(\sigma) = \sigma_i$, if $f \in \sigma_i$ then $\Rightarrow f \in \sigma$.*

When two traces of the net have the same observability and we project them on the same component, the resulting projections also have the same observability. This result is captured by the following proposition.

Proposition 2. *Consider the net $\mathcal{N} = \mathcal{N}_{A_1} \times \dots \times \mathcal{N}_{A_n}$ and $\sigma, \alpha \in \mathrm{traces}^\omega(\mathcal{N})$ with $P_i(\sigma) = \sigma_i$ and $P_i(\alpha) = \alpha_i$, we have $obs(\sigma) = obs(\alpha)$ implies $obs(\sigma_i) = obs(\alpha_i)$.*

Note that this result only holds because the product of nets synchronize on the set of shared actions.

Unfolding prefixes. The *unfolding* of a Petri net $\mathcal{N}$ is a (potentially infinite) acyclic net that can be obtained by starting from the initial marking of $\mathcal{N}$ and successively firing its transitions, as follows: (a) for each new firing a fresh transition (called an *event*) is generated; (b) for each newly produced token a fresh place (called a *condition*) is generated. Due to its structural properties, the reachable markings

of $\mathcal{N}$ can be represented using *configurations* of the unfolding. Intuitively, a configuration is a finite partially ordered execution, i.e. an execution where the order of firing of concurrent events is not important.

The unfolding is infinite whenever $\mathcal{N}$ has an infinite execution; however, if $\mathcal{N}$ is bounded (and thus has finitely many reachable states) then the unfolding eventually starts to repeat itself and can be truncated (by identifying a set of *cut-off events*) without loss of information, yielding a *finite and complete prefix.*

LTL-X and Büchi automata. *Linear time temporal logic* (LTL) [Pnueli, 1977] is a logic allowing to specify the properties of computations, and LTL-X is the fragment of LTL obtained by removing the *next-state* modality. LTL-X plays a prominent role in formal verification.

Deciding whether all computations of system S satisfy φ is equivalent to deciding whether some computation of S satisfies $\neg\varphi$ [Vardi and Wolper, 1986]. Formula $\neg\varphi$ is converted into a *Büchi automaton* $A_{\neg\varphi}$ accepting the computations satisfying $\neg\varphi$ [Gastin and Oddoux, 2001]. Then, S and $A_{\neg\varphi}$ are synchronized in such a way that the language of the resulting Büchi automaton $S \times A_{\neg\varphi}$ is the intersection of the language of $A_{\neg\varphi}$ and the set of all the possible computations of S. Hence, in this way one can reduce the original verification problem to checking if the language accepted by the Büchi automaton $S \times A_{\neg\varphi}$ is empty, which can be efficiently solved.

Unfolding based LTL-X model checking. In [Esparza and Heljanko, 2001] an efficient approach to model checking LTL-X properties of Petri nets based on unfolding prefixes was proposed. Its main differences from the automata-based approach outlined above are the following. The Büchi automaton $A_{\neg\varphi}$ for the LTL-X property φ is translated into a Petri net $\mathcal{N}_{\neg\varphi}$, called *Büchi net* (simply by replacing the automata states by places and automata transitions by transitions). Then its synchronization with the Petri net model of system S is performed at the level of Petri nets rather than reachability graphs, resulting in another Büchi net. The synchronization is defined such that the concurrency present in S is preserved as much as possible, which is important for the subsequent unfolding. Then the resulting synchronization is unfolded, and the cut-off events are defined such that the resulting finite and complete prefix can be viewed as a tableau proof, from which it is easy either to conclude that the property holds or to find a trace of S violating the property. This approach can significantly outperform methods based on explicit construction of reachability graphs in case of highly concurrent systems.

4 Diagnosability Analysis

We present now the notion of diagnosability. Informally, a fault $f \in \Sigma_F$ is diagnosable if it is possible to detect, within a finite delay, occurrences of such a fault using the record of observed actions. In other words, a fault is not diagnosable if there exist two infinite runs from the initial state with the same infinite sequence of observable actions but only one of them contains the fault.

Definition 1. *A fault f is diagnosable in $\mathcal{N}$ iff $\forall \sigma, \alpha \in \mathrm{traces}^\omega(\mathcal{N}) : obs(\sigma) = obs(\alpha)$ and $f \in \sigma$ implies $f \in \alpha$. $\mathcal{N}$ is diagnosable, denoted by $\boldsymbol{diag}(\mathcal{N})$, if and only if every fault $f \in \Sigma_F$ is diagnosable.*

As automata can be seen as nets with no concurrency and diagnosability is a property that consider (sequential) runs, the above definition can also be applied for automata.

Proposition 3. *Consider the automaton A and its corresponding net $\mathcal{N}_A$, we have $\boldsymbol{diag}(A) \Leftrightarrow \boldsymbol{diag}(\mathcal{N}_A)$.*

Example 2. *Consider the components A and B from Figure 2. The only pair of traces in A with the same observability are of the form $f\widehat{o}_3$ (one for each branch from the initial state). As both traces contain the fault f, system A is diagnosable. In the case of B, each observable trace corresponds to a unique run, therefore B is diagnosable. Now, consider the net $\mathcal{N}_1$ from Figure 3 and Figure 4, we can see that every trace contains a fault, therefore $\mathcal{N}_1$ is diagnosable. For net $\mathcal{N}_2$ from Figure 4 we have two traces, $o_2u_2\widehat{o}_4$ and $o_2fu_2\widehat{o}_4$ that have the same observability, but one of them contains a fault and the other does not, therefore $\mathcal{N}_2$ is not diagnosable.*

The product of automata is usually much bigger than the product of their corresponding nets as every possible interleaving is constructed, however there is an isomorphism between their runs [Baldan *et al.*, 2010] and we have the following result.

Proposition 4. *Let $\{A_1, \ldots, A_n\}$ be a set of automata, then $\boldsymbol{diag}(A_1 \times \cdots \times A_n) \Leftrightarrow \boldsymbol{diag}(\mathcal{N}_{A_1} \times \cdots \times \mathcal{N}_{A_n})$.*

We can now exploit the concurrency of the system and analyze its diagnosability by the verification of Petri nets.

LTL-X model checking for non-diagnosability. The diagnosability property is verified using LTL-X model checking based on Petri net unfoldings [Madalinski and Khomenko, 2010]. The *verifier* $\mathcal{V}$ is built with respect to a fault f by synchronizing two replicas of $\mathcal{N}$ on the observable transitions. Note that for efficiency reasons one replica does not consider the fault.

Intuitively, the two replicas are put side-by-side, and then each observable transition in the first replica is fused with each transition in the second replica that has the same label (each fusion produces a new transition, and the original observable transitions are removed). One can see that there is a one-to-one correspondence between the traces of $\mathcal{V}$ and pairs of traces of $\mathcal{N}$ with the same projections on the set of observable actions.

As explained above, given the verifier $\mathcal{V}$, checking the complement $\overline{\mathbf{diag}}$ of the diagnosability property can be reduced to checking the existence of an infinite trace of $\mathcal{V}$ containing an occurrence of f, in LTL-X it can be expressed as $\overline{\mathbf{diag}} \stackrel{\mathrm{df}}{=} \Diamond f$, where $\Diamond$ is the modality *eventually*.

Example 3. *The verifier $\mathcal{V}_2^D$ of the net $\mathcal{N}_2^D$ (presented in the next section, see Figure 7) is depicted in Figure 5. The superscript is used to distinguish nodes belonging to each copy of $\mathcal{N}_2^D$, e.g. there are two copies of u_2 in $\mathcal{V}_2^D$, u_2^1 and u_2^2; the fusion transitions do not have superscripts: they are considered 'common'. The infinite trace of $\mathcal{V}_2^D : o_2f^1u_2^1u_2^2\widehat{o}_4$ satisfies the $\overline{\mathbf{diag}}$ property. This trace of $\mathcal{V}_2^D$ corresponds to the pair of traces $o_2fu_2\widehat{o}_4$ and $o_2u_2\widehat{o}_4$ of $\mathcal{N}_2^D$, constituting a witness of diagnosability violation.*

5 Distributing the Diagnosability Analysis

In this section we present a method that allows to decide the diagnosability of a distributed system in terms of the diagnosability of each faulty component interacting with fault

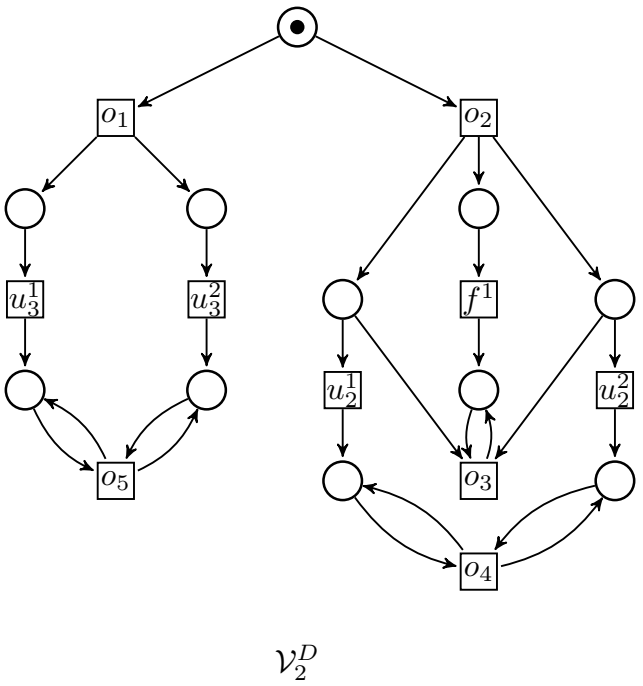


Figure 5: Verifier of $\mathcal{N}_2^D$

free versions of the remaining ones. These diagnosability analyses can be done in parallel.

For testing the diagnosability of a fault $f \in \Sigma_F$ in a net $\mathcal{N} = \mathcal{N}_{A_1} \times \cdots \times \mathcal{N}_{A_n}$, we consider a component i and compose it with fault free versions of the others, we denote such net as $\mathcal{N}^i$. These fault free versions may be taken as the specification of each component, when provided, or can be computed by removing the fault f in the net of such component using Algorithm 1 and considering it as the correct behavior of the system.

Algorithm 1

Require: A Petri net $\mathcal{N} = (P, T, F, M_0, \lambda)$
Ensure: A f-fault free version of $\mathcal{N}$

```
P' := M_0 , T' := ∅ , P := P \ M_0
while ∃t ∈ T\T' : •t ⊆ P' do
    if λ(t) ≠ f then
        P' := P' ∪ •t
        T' := T' ∪ {t}
    end if
    T := T \ {t}
end while
F' := F ∩ ((P' × T') ∪ (T' × P'))
λ' := λ_|T'
return N^f = (P', T', F', M_0, λ')
```

We now consider the net $\mathcal{N}^i$ composed by component $\mathcal{N}_{A_i}$ and the fault free version of $\mathcal{N}_{A_j}$ for $j \neq i$. Figure 6 shows the four components after removing fault f and Figure 7 shows the product nets obtained after these reductions.

Example 4. *Let us consider the nets from Figure 7. System $\mathcal{N}_1^B$ is trivially diagnosable. In the case of $\mathcal{N}_1^A$, it is easy to see that the observable traces are of the form $\widehat{o}_3$, but all traces containing o_3 also contain f and therefore $\mathcal{N}_1^A$ is also diagnosable. Traces $o_2 u_2 \widehat{o}_4$ and $o_2 f u_2 \widehat{o}_4$ of net $\mathcal{N}_2^D$ have the same observability, but one contains a fault and the other does not. We can conclude that $\mathcal{N}_2^D$ is not diagnosable. This result is consistent with the one obtained by the verifier in Example 3.*

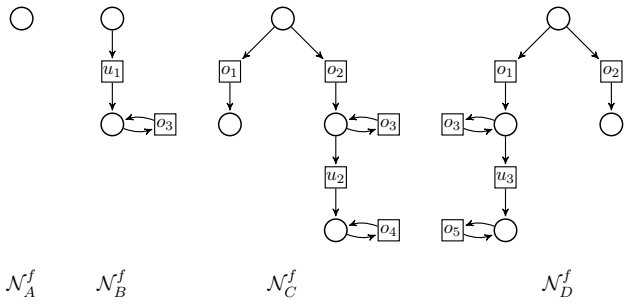


Figure 6: Components after removing their faults

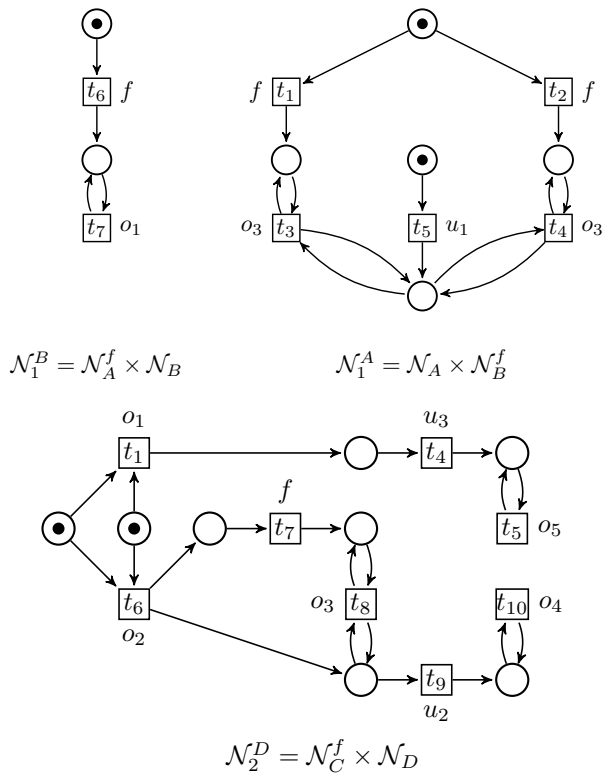


Figure 7: Nets after removing faults in some components

Clearly the traces of $\mathcal{N}^i$ are those of $\mathcal{N}$ such that its projections on every A_j are fault free for $j \neq i$.

Proposition 5. *Let $\mathcal{N}$ be a net, then $\sigma \in \text{traces}^\omega(\mathcal{N}^i)$ iff $\sigma \in \text{traces}^\omega(\mathcal{N}) \wedge \forall j \neq i, \sigma_j : P_j(\sigma) = \sigma_j \Rightarrow f \notin \sigma_j$.*

The following result states necessary conditions for the diagnosability of $\mathcal{N}$, i.e. the non diagnosability of $\mathcal{N}^i$ for some i implies the non diagnosability of $\mathcal{N}$.

Theorem 1. *Consider the net $\mathcal{N}$, then*

$$\textbf{\textit{diag}}(\mathcal{N}) \Rightarrow \bigwedge_{i=1}^{n} \textbf{\textit{diag}}(\mathcal{N}^i)$$

Proof 1. *Lets assume that $\neg$**diag**$(\mathcal{N}^i)$ for some i, then there exist $\sigma, \alpha \in \text{traces}^\omega(\mathcal{N}^i)$ and f such that $obs(\sigma) = obs(\alpha)$ with $f \in \sigma$, but $f \notin \alpha$. We know from Proposition 5 that every trace in $\mathcal{N}^i$ is a trace in $\mathcal{N}$, so we have found two traces of $\mathcal{N}$ with the same observability, one containing a fault and the other one not. Therefore $\mathcal{N}$ is non-diagnosable.*

Example 5. *We see in Example 4 that $\mathcal{N}_2^D$ is non diagnosable. Using Theorem 1 we can conclude that $\mathcal{N}_2$ is non diagnosable, which is consistent with the diagnosability analysis made in Example 2.*

As explained above, the idea is to build a diagnosable component and to test that its interaction with the others fault free component is also diagnosable. We can then decide the diagnosability of $\mathcal{N} = \mathcal{N}_{A_1} \times \cdots \times \mathcal{N}_{A_n}$ in terms of the diagnosability of A_i and $\mathcal{N}^i$.

Theorem 2. *Let $\mathcal{N} = \mathcal{N}_{A_1} \times \cdots \times \mathcal{N}_{A_n}$, then*

$$\bigwedge_{i=1}^{n} (\boldsymbol{diag}(A_i) \wedge \boldsymbol{diag}(\mathcal{N}^i)) \Rightarrow \boldsymbol{diag}(\mathcal{N})$$

Proof 2. *Let assume that we have a fault $f \in \Sigma_F$ and $\sigma, \alpha \in \mathrm{traces}^\omega(\mathcal{N})$ with $f \in \sigma$ and $obs(\sigma) = obs(\alpha)$, we need to prove that $f \in \alpha$. Consider the following cases:*

1. *if $\sigma, \alpha \in \mathrm{traces}^\omega(\mathcal{N}^i)$ we can prove by $\mathcal{N}^i$'s diagnosability that $f \in \alpha$ and then $\mathcal{N}$ is diagnosable,*
2. *if $\alpha \notin \mathrm{traces}^\omega(\mathcal{N}^i)$, using the hypothesis that $\alpha \in \mathrm{traces}^\omega(\mathcal{N})$, we can apply Proposition 5 and obtain that $\exists \alpha_i : P_i(\alpha) = \alpha_i \wedge f \in \alpha_i$. By Proposition 1 we know that every fault belonging to a projection also belongs to the trace in the net, then $f \in \alpha$ and $\mathcal{N}$ is diagnosable,*
3. *if $\alpha \in \mathrm{traces}^\omega(\mathcal{N}^i)$ and $\sigma \notin \mathrm{traces}^\omega(\mathcal{N}^i)$ we know by Proposition 5 that $\forall \alpha_i : P_i(\alpha) = \alpha_i$ and $f \notin \alpha_i$ and also that $\exists \sigma_i : P_i(\sigma) = \sigma_i$ with $f \in \sigma_i$. As $obs(\sigma) = obs(\alpha)$ we have that $obs(\sigma_i) = obs(\alpha_i)$ by Proposition 2. Finally, as A_i is diagnosable and $f \in \sigma_i$, the fault should belong to α_i, leading to a contradiction. We can conclude that $\mathcal{N}$ is diagnosable.*

6 Conclusions

We have presented a framework for the distributed diagnosability analysis of concurrent systems. We remove the assumption that a kind of fault can only occur in a single component (which is usually made in faulty distributed systems) and allow to analyze more general systems. The method presented in this paper is a continuation of [Ponce de León *et al.*, 2013], which to the best of our knowledge, is the first method that allows the diagnosability analysis to be done in a parallelized manner. Thus, a component can do the diagnosability analysis independently of other components, even when the other components are not yet ready. Furthermore, we employ LTL-X model checking based on Petri net unfolding to test diagnosability, which has been proven to be very efficient.

We plan to try to reduce the system in order to obtain minimal components from which we can infer the diagnosability of the original global system. In addition, we intend to relax the assumption that the communicating (synchronizing) events are observable. Moreover, we aim to apply our framework to other diagnosability related properties such as predictability.

References

[Baldan *et al.*, 2010] Paolo Baldan, Thomas Chatain, Stefan Haar, and Barbara König. Unfolding-based diagnosis of systems with an evolving topology. *Inf. Comput.*, 208(10):1169–1192, 2010.

[Debouk *et al.*, 2000] R. Debouk, S. Lafortune, and D. Teneketzis. Coordinated decentralized protocols for failure diagnosis of discrete event systems. *Discrete Event Dynamic Systems*, 10(1-2):33–86, 2000.

[Esparza and Heljanko, 2001] Javier Esparza and Keijo Heljanko. Implementing LTL model checking with net unfoldings. In *Proc. of SPIN'01*, pages 37–56, 2001.

[Esparza *et al.*, 2002] J. Esparza, S. Römer, and W. Vogler. An Improvement of McMillan's Unfolding Algorithm. *Form. Methods Syst. Des.*, 20(3):285–310, 2002.

[Fabre *et al.*, 2005] E. Fabre, A. Benveniste, S. Haar, and C. Jard. Distributed Monitoring of Concurrent and Asynchronous Systems. *J. Discrete Event Systems*, 15:33–84, 2005.

[Gastin and Oddoux, 2001] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In *Proc. of CAV'01*, volume 2102, pages 53–65, 2001.

[Heljanko *et al.*, 2002] Keijo Heljanko, Victor Khomenko, and Maciej Koutny. Parallelisation of the Petri net unfolding algorithm. In *Proc. of TACAS'2002*, volume 2280, pages 371–385, 2002.

[Jiang *et al.*, 2000] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 46:1318–1321, 2000.

[Khomenko, 2012] V. Khomenko. Punf. http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/punf, 2012.

[Madalinski and Khomenko, 2010] A. Madalinski and V. Khomenko. Diagnosability verification with parallel LTL-X model checking based on Petri net unfoldings. In *Proc. of SysTol'10*, pages 398–403, 2010.

[McMillan, 1992] K. L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Proc. of CAV'92*, pages 164–177, 1992.

[Pencolé, 2004] Y. Pencolé. Diagnosability analysis of distributed discrete event systems. In *ECAI*, pages 43–47, 2004.

[Pnueli, 1977] A. Pnueli. The temporal logic of programs. In *Proc. of FOCS'77*, pages 46–57, 1977.

[Ponce de León *et al.*, 2013] Hernán Ponce de León, Gonzalo Bonigo, and Laura Brandán Briones. Distributed analysis of diagnosability in concurrent systems. In *The 24th International Workshop on Principles of Diagnosis (DX'13)*, 2013.

[Sampath *et al.*, 1995] 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, September 1995.

[Schumann and Huang, 2008] A. Schumann and J. Huang. A scalable jointree algorithm for diagnosability. In *AAAI*, pages 535–540, 2008.

[Schumann and Pencolé, 2007] A. Schumann and Y. Pencolé. Scalable diagnosability checking of event-driven system. In *In Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI07*, pages 575–580, 2007.

[Vardi and Wolper, 1986] M. Vardi and P. Wolper. An automata theoretic approach to automatic program verification. In *Proc. of 1st IEEE LICS*, pages 332–345, 1986.

[Winskel, 1985] Glynn Winskel. Petri nets, morphisms and compositionality. In *Applications and Theory in Petri Nets*, pages 453–477, 1985.

[Ye and Dague, 2012] L. Ye and P. Dague. Diagnosability analysis for self-observed distributed discrete event systems. In *VALID*, pages 93–98, 2012.