A

Diagnosability under Weak Fairness

VASILEIOS GERMANOS, Newcastle University
STEFAN HAAR, INRIA & LSV (ENS Cachan & CNRS)
VICTOR KHOMENKO, Newcastle University
STEFAN SCHWOON, LSV (ENS Cachan & CNRS) & INRIA

In partially observed Petri nets, diagnosis is the task of detecting whether or not the given sequence of observed labels indicates that some unobservable fault has occurred. Diagnosability is an associated property of the Petri net, stating that in any possible execution an occurrence of a fault can eventually be diagnosed.

In this paper we consider diagnosability under the weak fairness (WF) assumption, which intuitively states that no transition from a given set can stay enabled forever — it must eventually either fire or be disabled. We show that a previous approach to WF-diagnosability in the literature has a major flaw, and present a corrected notion. Moreover, we present an efficient method for verifying WF-diagnosability based on a reduction to LTL-X model checking. An important advantage of this method is that the LTL-X formula is fixed — in particular, the WF assumption does not have to be expressed as a part of it (which would make the formula length proportional to the size of the specification), but rather the ability of existing model checkers to handle weak fairness directly is exploited.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Software/Program Verification

General Terms: Reliability, Verification

Additional Key Words and Phrases: Diagnosability, weak fairness, model checking, LTL-X, formal verification, Petri nets

1. INTRODUCTION

The *diagnosability* of systems has recently drawn the attention of many researchers in both artificial intelligence and control theory communities. *Diagnosis* is the process of explaining abnormal behaviours of a physical system, and diagnosability is an important property that determines the possibility of detecting faults given a set of observations. If a system is diagnosable, it is always possible to determine whether the fault has occurred by observing the system's behaviour for sufficiently long time, and then the diagnosis can find possible explanations for the given sequence of observations. Otherwise there are scenarios in which it is impossible to tell whether the fault has occurred or not, no matter for how long the system is observed. Non-diagnosability usually indicates that the system should be re-designed by e.g. changing the sensor map and/or augmenting it with additional sensors to make more events observable or to make the observations more discernable by distinguishing different transitions that had previously been registered under the same observation label.

This research was supported by the EPSRC grants EP/K001698/1 (UNCOVER) and EP/L025507/1 (A4A), and by the project IMPRO ANR-2010-BLAN-0317.
Author's addresses: V. Germanos, V. Khomenko: School of Computing Science, Newcastle University, Newcastle upon Tyne, UK; S. Haar, S. Schwoon: LSV, ENS Cachan, 94230 Cachan, France.

DOI:http://dx.doi.org/10.1145/0000000.0000000

The seminal work [Sampath et al. 1995] introduced a formal language framework for diagnosis and analysis of diagnosability properties of discrete event systems represented by finite automata. The proposed method for diagnosability verification was based on the construction of a *diagnoser* — an automaton with only observable transitions that allows one to estimate states of the system by observing its traces. Improvements based on the *twin plant* method have been introduced in [Jiang et al. 2001; Schumann and Pencolé 2007], where the basic idea was to build a *verifier* by constructing the synchronous product of the system with itself on observable transitions. This allowed to avoid constructing the exponentially big diagnoser, and reduce the complexity of diagnosability checking to polynomial in the number of reachable states [Yoo and Lafortune 2002]. The verifier compares every pair of executions in the system that have the same projection on the observable transitions. If the original system is given as a labelled Petri net, then the verifier can be constructed directly, by synchronising the original net with its replica at the Petri net level, and the problem reduces to model checking of a fixed LTL-X [Pnueli 1977; Lamport 1983] property of the verifier [Madalinski and Khomenko 2010]. Further work on the twin-plant construction for Petri nets include [Cabasino et al. 2012], where both diagnosability and K-diagnosability (the fault must be diagnosed at most K steps after its occurrence) are considered, and [Madalinski et al. 2010], where Petri net unfoldings are used to make diagnosability checking faster.

Recent work [Haar et al. 2013] presented a diagnosis method that encompasses *weak fairness*. There, concurrent systems are modelled by partially observable safe Petri nets, and diagnosis is carried out under the assumption that all executions of the Petri net are weakly fair, that is, the only infinite executions admitted are those in which any transition *enabled* at some stage will be *disabled* at some later stage, i.e. either it will actually fire later in that execution, or else some conflicting transition will fire. Under this assumption, a given finite observation diagnoses a fault if no finite execution yielding this observation can be extended to a weakly fair fault-free execution. The work in [Haar et al. 2013] gave a procedure for deciding this diagnosis problem. It remained open for which systems this procedure reliably diagnoses faults, i.e. how to determine whether a system is *diagnosable* under the weak fairness assumption. In this paper, we address this problem.

Note that a first definition of diagnosability under weak fairness was proposed in [Agarwal et al. 2012]. However, that definition is incompatible with the notion of diagnosis in [Haar et al. 2013] and contains a major flaw, as we shall point out below.

We make the following contributions in this paper:

— We develop a notion of weakly fair (WF) diagnosability, which corrects and supersedes the one in [Agarwal et al. 2012].
— We characterise executions that *witness* violations of WF-diagnosability.
— We further investigate the special case where fault transitions are not WF, i.e. a fault is a *possible* outcome in the system but not one that is *required* to happen. (Our examples in Sect. 5 suggest that this is a reasonable assumption in practice.) Under this assumption, the notion of a witness can be significantly simplified.
— We develop a method for verifying WF-diagnosability in this case, and evaluate it experimentally.

This paper is the full version of the conference paper [Germanos et al. 2014]. The following additional contributions are made in this paper:

— An alternative (and more convenient) notion of a witness of WF-diagnosability violation, together with a proof of its equivalence to the notion in [Germanos et al. 2014].
— A general method for verifying WF-diagnosability, which allows faults to be WF.

The paper is organised as follows: Sect. 2 discusses existing notions of diagnosability and explains why they are problematic for concurrent systems. Sect. 3 develops our notion of WF-diagnosability and witnesses of its violation. Sect. 4 presents the construction of the verifier, which is evaluated in Sect. 5. Sect. 6 presents a generalised method for verifying WF-diagnosability, which allows faults to be WF. We conclude in Sect. 7.

2. PETRI NETS AND DIAGNOSABILITY

This section explains why the standard notion of diagnosability, as well as the notion of WF-diagnosability developed in [Agarwal et al. 2012], are problematic, which motivates our new definition, to be presented later.

Throughout the paper we assume that the system is modelled as a labelled Petri net (LPN) $\mathcal{N}$, where each transition is labelled with the performed action. The actions are partitioned into *observable* and *silent,* i.e. there is a labelling function ℓ mapping the LPN's transitions to $O \cup \{\varepsilon\}$, where O is an alphabet of *observable* actions and $\varepsilon \notin O$ is the empty word denoting the *silent* action. (Intuitively, observable actions correspond to controller commands and sensor readings, while the silent action models some internal activity that is not recorded by sensors.) This labelling function ℓ can be naturally extended to finite and infinite executions of the LPN, projecting them to words in O^* or O^ω. We assume that the LPN is free from deadlocks and divergencies, i.e. every execution of the LPN can be extended to an infinite one, and every infinite execution of the LPN has infinitely many observable transitions. Some of the transitions are designated as *faults;* w.l.o.g., we assume that none of them is observable. An example in Fig. 1 shows an LPN with the observable transitions t_3, t_4 and t_5 with $\ell(t_3) = a$, $\ell(t_4) = b$ and $\ell(t_5) = tick$ (the other transitions are unobservable). Note that we draw faults as black boxes, and the observable transitions are shaded.

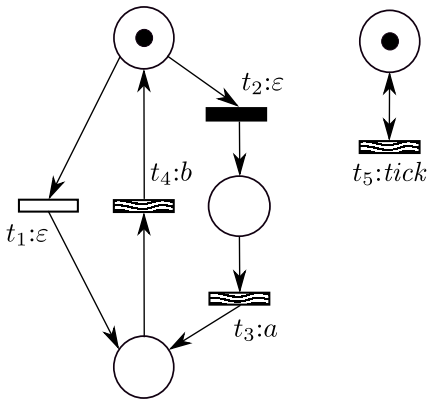

Fig. 1: This LPN without t_5 would be diagnosable, but t_5 makes it undiagnosable. Making t_3 WF makes the LPN diagnosable.

The usual interleaving semantics is used in this paper; in particular, references to time in temporal modalities like 'eventually' and 'always' are w.r.t. the 'internal' clock that progresses when some transition of the LPN fires.

2.1. Standard diagnosability

Given a finite execution σ of the LPN, the observer sees the outputs of the system $\ell(\sigma) \in O^*$, and needs to conclude whether some fault transition t has definitely oc-

curred in σ. In a diagnosable system, once a fault has occurred, the observer is able to *eventually* detect this. That is, provided that the suffix of σ after the first occurrence of a fault in it is sufficiently long, the observer should be able to conclude that each execution with the same projection $\ell(\sigma)$ contains a fault, i.e. a fault has either already occurred or will definitely occur in the future. Let us first recall the definition of standard diagnosability:[1]

Definition 2.1 (*Diagnosability*). An LPN is diagnosable iff for all its infinite traces σ and ρ such that $\ell(\sigma) = \ell(\rho)$, σ contains a fault iff ρ contains a fault.

In other words, a non-diagnosable LPN has two infinite executions having the same projection onto the observable actions and such that one of them contains a fault and the other does not; such a pair of traces constitutes a *witness* of diagnosability violation.

For example, the LPN in Fig. 1 is not diagnosable. Indeed, the diagnoser can only conclude that the fault has occurred after observing a. However, the infinite execution $t_2 t_5^\omega$ contains a fault but never fires t_3. Nevertheless, if t_5 is removed, the LPN becomes diagnosable.

2.2. Weak fairness

The example from Fig. 1 exhibits a pathological property of this notion of diagnosability: a diagnosable system ceases to be such simply because some unrelated concurrent activity is added to the specification. In practice, it is often reasonable to assume that the system is keen to fire its enabled transitions, and *cannot perpetually ignore an enabled transition.* In other words, one can consider the LPN in Fig. 1 diagnosable, by declaring the infinite execution $t_2 t_5^\omega$ impossible.

To capture this idea formally, the notion of *weak fairness* is helpful [Vogler 1995]. Suppose the designer wants to disallow some of the transitions to be perpetually ignored when enabled. We call such transitions *weakly fair* (WF). An infinite execution σ of the LPN is called *weakly fair* (WF) if for each WF transition t, if t is enabled after some prefix of σ then the rest of σ contains some transition in $({}^\bullet t)^\bullet$, see Fig. 2. All finite executions are regarded as WF. We now can use the set of WF executions as the semantics of the LPN, i.e. other executions are considered impossible. Coming back to the example in Fig. 1, if t_3 is WF then the execution $t_2 t_5^\omega$ is not WF and thus impossible, and so the LPN becomes diagnosable.

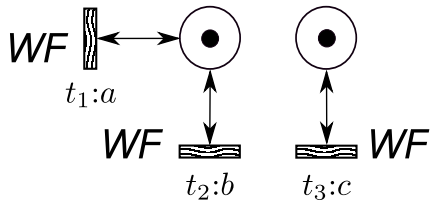

Fig. 2: (i) The execution $(t_1 t_2 t_3)^\omega$ is WF as no enabled transition is perpetually ignored by it. (ii) The execution $(t_1 t_2)^\omega$ is not WF as t_3 is enabled but all the transitions in $({}^\bullet t_3)^\bullet = \{t_3\}$ are perpetually ignored. (iii) The execution $(t_1 t_3)^\omega$ is WF: even though t_2 is perpetually ignored, $t_1 \in ({}^\bullet t_2)^\bullet = \{t_1, t_2\}$ is fired.

[1]This definition is taken from [Madalinski and Khomenko 2010]. It is subtly different from the original definition in [Sampath et al. 1995], but equivalent for finite state systems, and simpler to use in practice. (An LPN has finitely many reachable markings iff it is bounded.)

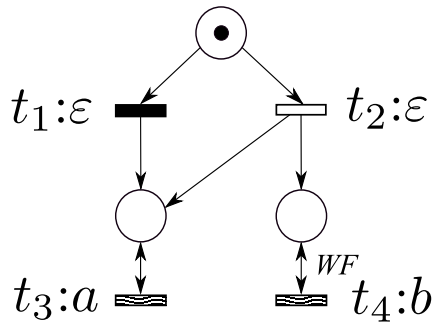

Fig. 3: This LPN is WF-diagnosable according to the definition from [Agarwal et al. 2012], but not according to the corrected definition (Def. 3.1 and Lemma 3.3). Note that the observer cannot detect the fault in finite time.

It is tempting to derive the definition of WF-diagnosability simply by taking Def. 2.1 and restricting it to WF executions. In fact, such an approach was taken in [Agarwal et al. 2012], where an LPN $\mathcal{N}$ was said to be WF-diagnosable iff for all its infinite WF executions σ and ρ such that $\ell(\sigma) = \ell(\rho)$, σ contains a fault iff ρ contains a fault.

Unfortunately, this definition contains a major flaw, demonstrated by the example in Fig. 3. This LPN would be said to be diagnosable, while it is not possible for the observer to detect a fault in finite time, as one would have to observe the infinite trace a^ω to positively conclude that the fault has occurred.

3. WEAKLY FAIR DIAGNOSABILITY

To fix the problems exhibited in Sect. 2, we present a corrected definition of WF-diagnosability, where the possibility of detecting a fault in finite time is imposed. Intuitively, it states that each infinite WF execution containing a fault must have a finite prefix after which it is possible to conclude unambiguously that the fault has either occurred or will inevitably occur in future. Below we denote by '$<$' the prefix relation on sequences.

Definition 3.1 (*WF-diagnosability*). An LPN is WF-diagnosable iff each infinite WF execution σ containing a fault has a finite prefix $\hat{\sigma}$ such that every infinite WF execution ρ with $\ell(\hat{\sigma}) < \ell(\rho)$ contains a fault.

The LPN in Fig. 3 is not WF-diagnosable according to Def 3.1, as for each finite prefix (say, $t_1 t_3^n$ for some $n \in \mathbb{N}$) of the infinite WF execution $t_1 t_3^\omega$ containing a fault, there is a finite execution ($t_2 t_3^n$) with the same projection to observable actions, that can be extended to an infinite WF execution without a fault (e.g. $t_2 t_3^n (t_3 t_4)^\omega$).

In this example one can also identify a fault-free *infinite* execution $t_2 t_3^\omega$ that is in itself not WF, but each of its finite prefixes can be extended to an infinite fault-free WF execution. As we shall see, such an execution can always be found in a *bounded* LPN that is not WF-diagnosable.

Definition 3.2 (*Witness for a bounded LPN*). Let $\mathcal{N}$ be a bounded LPN. A pair of infinite executions (σ, ρ) with $\ell(\sigma) = \ell(\rho)$ is called *witness* (of WF-diagnosability violation) if σ is WF and contains a fault, and every prefix of ρ can be extended to an infinite fault-free WF execution.

LEMMA 3.3 (WF-DIAGNOSABILITY OF A BOUNDED LPN). *A bounded LPN $\mathcal{N}$ is WF-diagnosable iff no witness of its WF-diagnosability violation satisfying the conditions of Def. 3.2 exists.*

PROOF. If a witness satisfying Def. 3.2 exists then the condition of Def. 3.1 is violated, as for any prefix of σ one can choose a prefix of ρ with the same projection, which can be extended to a fault-free WF execution, i.e. $\mathcal{N}$ is not WF-diagnosable.

In the reverse direction: Suppose $\mathcal{N}$ is not WF-diagnosable. Then, according to Def. 3.1, there exists an infinite, WF, faulty execution σ such that for every finite prefix $\hat{\sigma} < \sigma$ there exists some infinite, WF, fault-free execution ρ with $\ell(\hat{\sigma}) < \ell(\rho)$. From σ, we shall construct a pair of executions (σ', ρ') constituting a witness according to Def. 3.2.

Let K be the number of states (i.e. reachable markings) of $\mathcal{N}$. Let $m(\sigma, i)$ denote the marking generated by the i-th observable transition in σ; since $\mathcal{N}$ has no divergencies, it is well-defined for all $i \geq 1$. Moreover, let $s(\sigma, i, j)$ denote the interval of σ starting after the i-th observable transition and ending at the j-th observable transition, for all $0 < i < j$. Furthermore, let k be the number of observable transitions in σ before the first occurrence of a fault.

By the pigeonhole principle, some marking m must satisfy $m = m(\sigma, i)$ for infinitely many i, and thus one can construct an infinite, strictly ascending sequence of indices $(i_j)_{j \geq 0}$ such that $i_0 > k$, and all $j \geq 0$ satisfy (i) $m(\sigma, i_j) = m$, and (ii) $s(\sigma, i_j, i_{j+1}) \cap ({}^\bullet t)^\bullet \neq \emptyset$ for every WF transition t enabled in m (such a subsequence exists since σ is WF and m appears infinitely often). Let $\hat{\sigma}$ be the prefix of σ with $|\ell(\hat{\sigma})| = i_K$.

Now, let ρ be an infinite, WF, fault-free sequence with $\ell(\hat{\sigma}) < \ell(\rho)$. By the pigeonhole principle, there must be two indices $0 \leq j_1 < j_2 \leq K$ with $m(\rho, i_{j_1}) = m(\rho, i_{j_2}) =: m'$.

We are now ready to conclude. Consider the execution σ', identical to σ up to $m(\sigma, i_{j_1})$ and then executing $s(\sigma, i_{j_1}, i_{j_2})^\omega$. This execution is infinite, contains a fault, and is WF by construction. Moreover, let ρ' be an infinite execution identical to ρ up to $m(\rho, i_{j_1})$ and then executing $s(\rho, i_{j_1}, i_{j_2})^\omega$. By construction, $\ell(\sigma') = \ell(\rho')$ but ρ' does not contain a fault. Also, every prefix of ρ' can be extended to a WF fault-free execution by going to the next occurrence of m' and then appending any suffix of ρ starting at an occurrence of m'. Thus, (σ', ρ') constitutes a witness. □

Def. 3.2 is difficult to use due to the necessity to consider every prefix of ρ. Lemma 3.4 provides an alternative characterisation for ρ. Its advantage is that instead of considering infinitely many prefixes of ρ, it considers a single well chosen one.

LEMMA 3.4. *Let $\mathcal{N}$ be a bounded LPN and ρ be an infinite execution. Then the following two statements are equivalent:*

(*1*) *every prefix of ρ can be extended to an infinite fault-free WF execution;*
(*2*) *ρ is fault-free and has a prefix $\hat{\rho}$ such that ρ passes through the marking reached by $\hat{\rho}$ infinitely many times, and $\hat{\rho}$ can be extended to an infinite fault-free WF execution.*

PROOF. $(1) \Rightarrow (2)$ Suppose every prefix of ρ can be extended to an infinite fault-free WF execution $(*)$. Then ρ must be fault-free, as otherwise some prefix of ρ would contain a fault, contradicting $(*)$. Since the LPN is bounded and ρ is infinite, ρ passes through some marking m infinitely many times. Let $\hat{\rho}$ be a prefix of ρ that reaches m first time. By $(*)$, $\hat{\rho}$ can be extended to an infinite WF execution.

$(1) \Leftarrow (2)$ Suppose ρ is fault-free and has a prefix $\hat{\rho}$ such that ρ passes through the marking m reached by $\hat{\rho}$ infinitely many times, and $\hat{\rho}$ can be extended to an infinite fault-free WF execution $\hat{\rho}\rho'$. Since ρ passes through m infinitely many times, every prefix of ρ can be extended to a longer prefix reaching m. In turn, this longer prefix can be extended by ρ' to an infinite fault-free execution, as ρ' is enabled from m. Moreover, this execution is WF, as $\hat{\rho}\rho'$ is WF, and ρ' fully determines whether the execution is WF or not. □

We note that in certain practical cases, the witness definition can be simplified. In particular, we consider the case when no fault transition is WF: Then one can further simplify the requirements imposed on ρ in Lemma 3.4.

LEMMA 3.5. *Let $\mathcal{N}$ be a bounded LPN where no fault transition is WF, and let ρ be an infinite execution. Then the following two statements are equivalent:*

(*1*) *every prefix of ρ can be extended to an infinite fault-free WF execution;*
(*2*) *ρ is fault-free.*

PROOF. The proof of (1) $\Rightarrow$ (2) is as in the proof of Lemma 3.4. As for the other direction, suppose ρ is infinite and fault-free and take any finite prefix $\hat{\rho}$ of ρ such that $\hat{\rho}\rho' = \rho$. We construct an infinite, WF, fault-free continuation of $\hat{\rho}$. If ρ itself is WF then we are done. Otherwise there exists some WF transition t that is enabled at some point in ρ after which ρ contains no more transition from $({}^\bullet t)^\bullet$; note that t is not a fault by the assumption. But this means that firing t cannot disable any transition in the rest of the execution, so we can insert it anywhere into ρ' without disabling the rest of this execution. The repeated application of this insertion process yields the required continuation of $\hat{\rho}$, and it always can be done in such a way that no enabled WF transition is perpetually ignored by the insertion process, and no transition from ρ' is indefinitely delayed by the newly inserted transitions. □

Lemma 3.5 provides another characterisation of ρ in witnesses provided that the net contains no WF fault transitions. This result is central for the WF-diagnosability verification method proposed in Sect. 4. Note also that this characterisation is quite similar to the (flawed) definition from [Agarwal et al. 2012], but with the following important differences: (i) it is limited to bounded LPNs without WF faults, and (ii) ρ is not required to be WF. As an example, a witness of WF-diagnosability violation for the LPN in Fig. 3 would be $(t_1 t_3^\omega, t_2 t_3^\omega)$; note that the latter execution is not WF itself, but that each of its prefixes can be extended to a WF execution.

It should be noted that the assumption that the faults are not WF is essential for Lemma 3.5. Indeed, consider the LPN in Fig. 4, where t_1^ω constitutes an infinite, fault-free execution as required by Lemma 3.5 (2) that has the same observations as the faulty execution $t_2 t_1^\omega$. Nonetheless, this LPN is trivially WF-diagnosable, as all its infinite WF executions contain the WF fault transition. Thus, the assumption about the absence of WF faults cannot be dropped in Lemma 3.5.

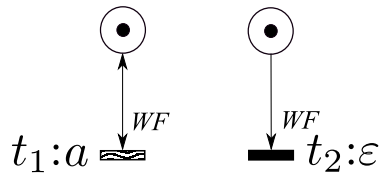

Fig. 4: A bounded LPN illustrating that the assumption about faults being non-WF is essential for Lemma 3.5. Despite the presence of an infinite fault-free execution t_1^ω, this LPN is trivially WF-diagnosable, as the fault must occur in every infinite WF execution.

Remark 3.6. The notion of WF-diagnosability is very different from *fair diagnosability* in the sense of [Biswas 2013b; Biswas et al. 2010; Biswas 2013a]: The fairness notion there concerns the behaviour of sequential systems w.r.t. *choices* — an execution is *fair* iff for every state that is visited infinitely often, every transition leading

out of it is selected infinitely many times. Assuming this property makes diagnosis more powerful, and leads to a notion of diagnosability for automata models that can be shown to coincide with stochastic diagnosability in the sense of [Thorsley and Teneketzis 2005] when adding an appropriate Markov law. In this paper we are dealing with very different issues; in fact, weak fairness is relevant only in *concurrent* systems, since in sequential systems as in [Biswas 2013b; Biswas et al. 2010; Biswas 2013a] every infinite run is WF. ◇

4. CHECKING WF-DIAGNOSABILITY

In this section we show how checking WF-diagnosability can be re-formulated in terms of LTL-X [Pnueli 1977; Lamport 1983] model checking.

Our approach works for a bounded LPN $\mathcal{N}$. We perform various operations on $\mathcal{N}$ to obtain another bounded LPN $\mathcal{V}$, called the *verifier*, which we check against a *fixed* LTL-X formula (in particular, its size does not depend on $\mathcal{N}$). To achieve this, we exploit the ability of many existing model checkers to handle weak fairness directly.[2]

We first introduce the operations on $\mathcal{N}$ needed to obtain $\mathcal{V}$ (Sect. 4.1), then recall the approach for non-WF diagnosability (Sect. 4.2), and finally present the modifications necessary to handle WF-diagnosability for the special case where no fault transition is WF (Sect. 4.3). The general case of bounded nets is considered in Sect. 6.

We use the net in Fig. 5 as a running example.

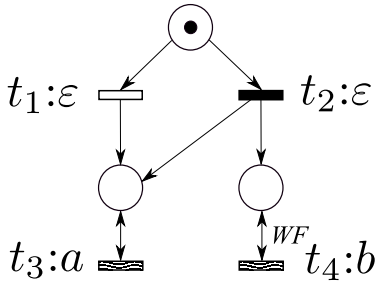

Fig. 5: An LPN similar to that in Fig. 3, but with a different choice of a fault transition. It is not diagnosable but WF-diagnosable, as an occurrence of a fault enables t_4, which can be perpetually ignored under the non-WF semantics, but must eventually fire — thus diagnosing the fault — under the WF semantics.

4.1. Net operations

In this paper we are concerned with the state-based LTL-X verification. However, the definition of diagnosability in Sect. 3 is action-based, and thus has to be re-formulated in terms of states. The first operation is defined for this purpose.

Fault monitor. We will need to keep track whether some execution contains a fault transition. Given $\mathcal{N}$, the net $\mathcal{N}^{ft}$ denotes $\mathcal{N}$ extended with two additional places $\overline{p_f}$ and p_f of which $\overline{p_f}$ is initially marked, indicating that no fault has happened so far. Then we make every fault transition move a token from $\overline{p_f}$ to p_f, indicating that a fault has occurred. Also, since a fault transition may fire several times in $\mathcal{N}$, another transition f' is added for each fault transition f, in order to simulate these subsequent firings in $\mathcal{N}^{ft}$. The construction is illustrated in Fig. 6, where it is applied to Fig. 5.

[2]The algorithm looking for an accepting (lasso-shaped) execution of a Büchi automaton can be modified in such a way as to ignore non-WF executions.

In terms of behaviour, $\mathcal{N}$ and $\mathcal{N}^{ft}$ are equivalent in a strong sense. Suppose that the transitions of $\mathcal{N}$ are injectively labelled, and the transitions of $\mathcal{N}^{ft}$ retain these labels, with the label of f and f' being the same. Then these two nets are strongly bisimilar. Moreover, if $\overline{p_f}$ in $\mathcal{N}^{ft}$ is unmarked then a fault occurred in the past.

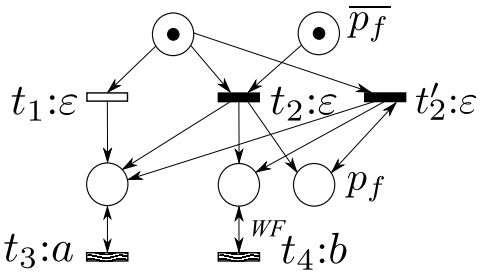

Fig. 6: Fault tracking net $\mathcal{N}^{ft}$ for the LPN in Fig. 5.

Stubs. We will want to know whether an infinite execution perpetually ignores certain enabled WF transitions. Given a subset of $\mathcal{N}$'s WF transitions and a 'fresh' initially marked place $stub\_monitor$, we can turn these transitions into *stubs* by removing all their outgoing arcs and adding $stub\_monitor$ to their presets.

Stubs are not meant to be executed: in fact, our LTL-X formulae will make such executions 'irrelevant' by demanding that $stub\_monitor$ remains always marked. Then, a 'relevant' WF execution that keeps $stub\_monitor$ marked cannot enable a stub forever.

Removing transitions. We can remove a given subset of transitions from an LPN, together with their incoming and outgoing arcs.

Synchronising. Let $\mathcal{N}$ and $\mathcal{N}'$ be two LPNs with disjoint sets of places and transitions. Intuitively, the *synchronisation* of $\mathcal{N}$ and $\mathcal{N}'$ puts $\mathcal{N}$ and $\mathcal{N}'$ side-by-side, and then each observable transition t of $\mathcal{N}$ is fused with each transition t' of $\mathcal{N}'$ that has the same label (each fusion produces a new transition, and t and t' remain in the result). Thus the synchronised net has three types of transitions: those from $\mathcal{N}$, those from $\mathcal{N}'$, and the fused ones.

4.2. Verifying ordinary diagnosability

We recall the verification of (non-WF) diagnosability from [Madalinski and Khomenko 2010] and show that it is unsuitable for WF-diagnosability. Let $\mathcal{N}$ be the original LPN. The construction works in the following steps:

(1) Let $\mathcal{N}^{ft}$ be the fault tracking net corresponding to $\mathcal{N}$.
(2) Let $\mathcal{N}'$ be a copy of $\mathcal{N}$.
(3) Let $\mathcal{N}_s$ be the result of synchronising $\mathcal{N}^{ft}$ and $\mathcal{N}'$.
(4) Remove from $\mathcal{N}_s$ all observable transitions of $\mathcal{N}^{ft}$.
(5) Remove from $\mathcal{N}_s$ all observable and fault transitions of $\mathcal{N}'$.
(6) Call the resulting net $\mathcal{V}$.

Note that after $\mathcal{V}$ has been built, it is no longer necessary to remember which actions are visible and which are not, and so we can disregard all the labelling and treat $\mathcal{V}$ as an unlabelled PN. This construction is illustrated in Fig. 7.

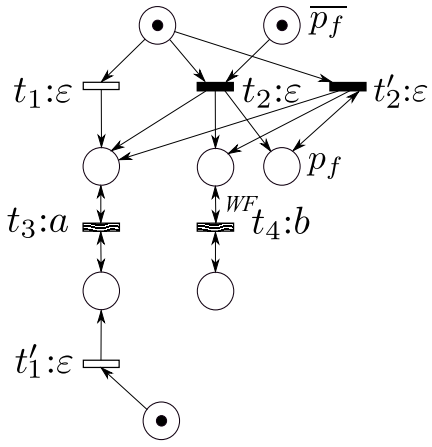

Fig. 7: The (non-WF) verifier for the LPN in Fig. 5.

It turns out [Madalinski and Khomenko 2010] that $\mathcal{N}$ is diagnosable iff the following LTL-X property holds for all traces of $\mathcal{V}$:

$$diag \stackrel{\text{df}}{=} \Box\, \overline{p_f},$$

i.e. one simply requires that no infinite trace of $\mathcal{V}$ contains an occurrence of a fault.

Conversely, a counterexample satisfying $\Diamond\, \neg\overline{p_f}$ is an infinite execution of $\mathcal{V}$ containing a fault; when projected to the parts corresponding to $\mathcal{N}^{ft}$ and $\mathcal{N}'$, it gives a witness of (non-WF) diagnosability violation, i.e. two infinite executions of $\mathcal{N}$ that have the same projection on the set of observable actions but the first contains a fault while the second does not. Similarly, such a pair of executions corresponds to an infinite trace of $\mathcal{V}$, with the first being executed by the part of $\mathcal{V}$ corresponding to $\mathcal{N}^{ft}$, and the second (without occurrences of faults) being executed by the part of $\mathcal{V}$ corresponding to $\mathcal{N}'$.

Unfortunately, this construction is not appropriate for WF-diagnosability, even if the executions of the verifier are restricted to be WF. For example, consider the net in Fig. 5. The verifier proposed in [Madalinski and Khomenko 2010] is shown in Fig. 7. It has an infinite execution containing a fault, $t_2 t_1' t_3^\omega$, which, when projected to $\mathcal{N}^{ft}$ and $\mathcal{N}'$, yields a pair of traces constituting a witness of diagnosability violation. However, this verifier cannot be used for checking WF-diagnosability simply by restricting its executions to be WF, as the same execution $t_2 t_1' t_3^\omega$ is actually WF, since t_4 is not permanently enabled by it (in fact, it is a dead transition in the verifier). Therefore, this execution is a false negative (the original LPN is in fact WF-diagnosable). Note that when this WF execution of the verifier is projected to $\mathcal{N}^{ft}$ and $\mathcal{N}'$, the resulting pair of traces will not constitute a witness of WF-diagnosability, as the former projection will be a non-WF execution of $\mathcal{N}^{ft}$ that perpetually ignores an enabled transition t_4.

Below, we amend $\mathcal{V}$ to fix this problem for bounded LPNs with no WF faults.

4.3. Verifier for non-WF fault transitions

Let $\mathcal{N}$ be a bounded LPN, in which no fault transition is WF. We keep the basic idea of the verifier construction from Sect. 4.2, i.e. our verifier $\mathcal{V}_{WF}$ will be the synchronisation of two nets, and a counterexample to our LTL-X formula will give a faulty execution σ in one net, and a fault-free execution ρ in the other net, such that (σ, ρ) is a witness.

The first important change is to check the formula only against WF executions. As seen in Sect. 4.2, this alone is not enough: The false counterexample obtained for Fig. 5 comes from the fact that $\mathcal{V}_{WF}$ allows σ to perpetually ignore a transition (here: t_4) if ρ

does not enable it. We use stubs to prevent this from happening. More precisely, $\mathcal{V}$ is constructed as follows:

(1) Obtain the net $\mathcal{N}_s$ as in Sect. 4.2; its fused transitions are declared non-WF.
(2) Turn in $\mathcal{N}_s$ the observable WF transitions of $\mathcal{N}^{ft}$ into stubs; they remain WF.
(3) Remove from $\mathcal{N}_s$ all observable and fault transitions of $\mathcal{N}'$.
(4) In $\mathcal{N}_s$, make the remaining transitions of $\mathcal{N}'$ non-WF.
(5) Call the resulting net $\mathcal{V}_{WF}$.

Fig. 8 shows the verifier $\mathcal{V}_{WF}$ for the LPN in Fig. 5.

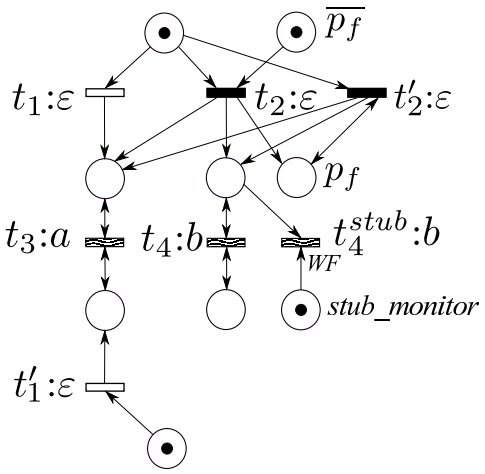

Fig. 8: The WF verifier for the LPN in Fig. 5.

Now we can formulate WF-diagnosability of the original $\mathcal{N}$ as a fixed LTL-X formula on $\mathcal{V}_{WF}$ that has to be checked for infinite WF executions only:

$$diag_{WF} \stackrel{\text{df}}{=} \Box\, \overline{p_f} \;\vee\; \Diamond\, \neg stub\_monitor.$$

Thus a counterexample is an infinite WF execution containing a fault but no stubs.

THEOREM 4.1 (CORRECTNESS OF SPECIALISED WF VERIFIER). *Let $\mathcal{N}$ be a bounded LPN where no fault transition is WF. Then $\mathcal{N}$ is WF-diagnosable iff all infinite WF executions of $\mathcal{V}_{WF}$ satisfy $diag_{WF}$.*

PROOF. According to Lemma 3.3, $\mathcal{N}$ is WF-diagnosable iff no witness (σ, ρ) exists. Since no fault transition is WF, we can employ the simplified condition of Lemma 3.5.

First, suppose $diag_{WF}$ is false, i.e. $\mathcal{V}_{WF}$ has an infinite WF execution τ that contains a fault and no stubs. Let σ and ρ be the projections of τ to $\mathcal{N}^{ft}$ and $\mathcal{N}'$, respectively. We claim that (σ, ρ) is a witness. Indeed, since $\mathcal{N}$ has no divergencies, τ must contain infinitely many observable transitions. Thus, both σ and ρ are infinite, and $\ell(\sigma) = \ell(\rho)$ holds; moreover, σ contains a fault but ρ does not. Finally, σ must be WF because τ is and no stubs are fired.

For the reverse direction, it is fairly straightforward to see that any witness (σ, ρ) gives rise to an execution τ of $\mathcal{V}_{WF}$ violating $diag_{WF}$, even if ρ satisfies only the simplified condition from Lemma 3.5. Moreover, τ is WF because σ is. The fact that ρ is not necessarily WF does not play a role, as ρ is executed in the part of the verifier corresponding to $\mathcal{N}'$ and so contains no WF transitions by construction. □

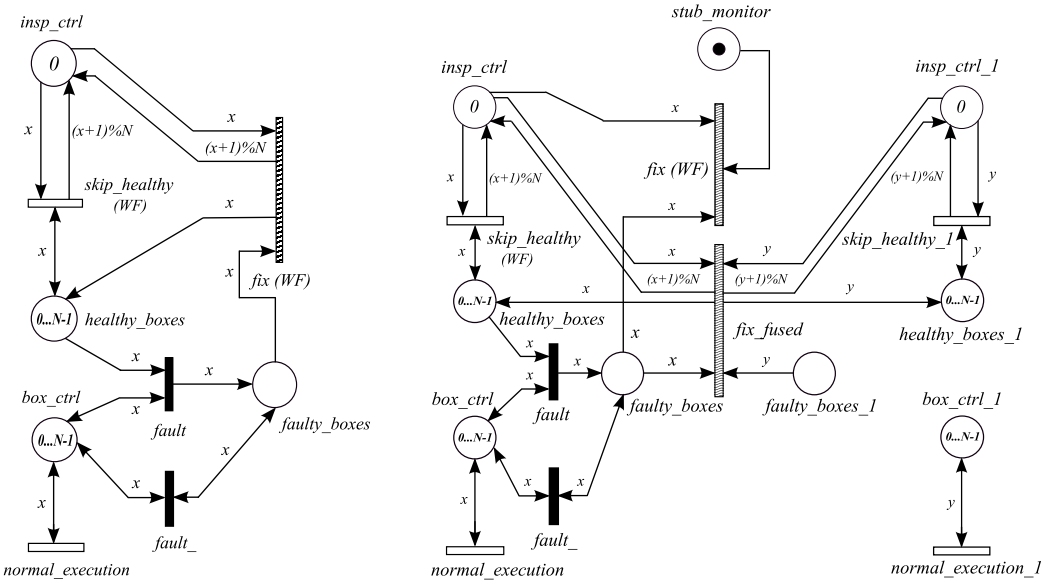

Fig. 9: The COMMBOX (n) benchmark (left) and the corresponding verifier (right).

5. EXPERIMENTAL RESULTS

In this section we present experimental results for the proposed WF-diagnosability approach. Furthermore, we demonstrate that the proposed approach can easily be lifted from low-level Petri nets to high-level ones: both the used benchmarks and the corresponding verifiers were modelled using high-level LPNs.

For the verification, the MARIA (modular reachability analyser) tool [Mäkelä 2005] was used. Since MARIA supports modular verification, it was possible to exploit the modular structure of the verifier (recall that it is built by synchronising two LPNs, see Sect. 4) to significantly speed up the verification.

It should be noted that finding interesting benchmarks was a challenging task: Despite a lot of theoretical work done in the area of diagnosability, rather few practical experiments have been conducted. Moreover, we wanted benchmarks where weak fairness is essential, i.e. removing some transitions from the WF set would make the system undiagnosable. Hence, we designed the following two new families of scalable benchmarks, available from the authors upon request.

COMMBOX *(n).* Fig. 9 shows a high-level LPN modelling the system comprising commutator boxes and an inspector, together with the verifier. It models n boxes commuting telephone calls. Normally, a box just handles telephone calls (the *normal_execution* transition), but occasionally it may register a fault (the *fault* transition) in a telephone line. Such an event, however, does not take the box out of action, and it still continues to commute calls (the *normal_execution* transition) and register further faults (the *fault_* transition). Nevertheless, the registered faults have to be considered and fixed, and so there is an inspector visiting the boxes on a round trip and fixing them if necessary (the *skip_healthy* and *fix* transitions). It is assumed that *fix* is the only observable transition, and one can be sure that a fault has occurred once it fires. Nevertheless, it is possible that the inspector indefinitely postpones visiting the boxes (i.e. its transitions are always preempted by, e.g., *normal_execution* which is always enabled), and so the system is undiagnosable. However, if the transitions modelling the inspector are

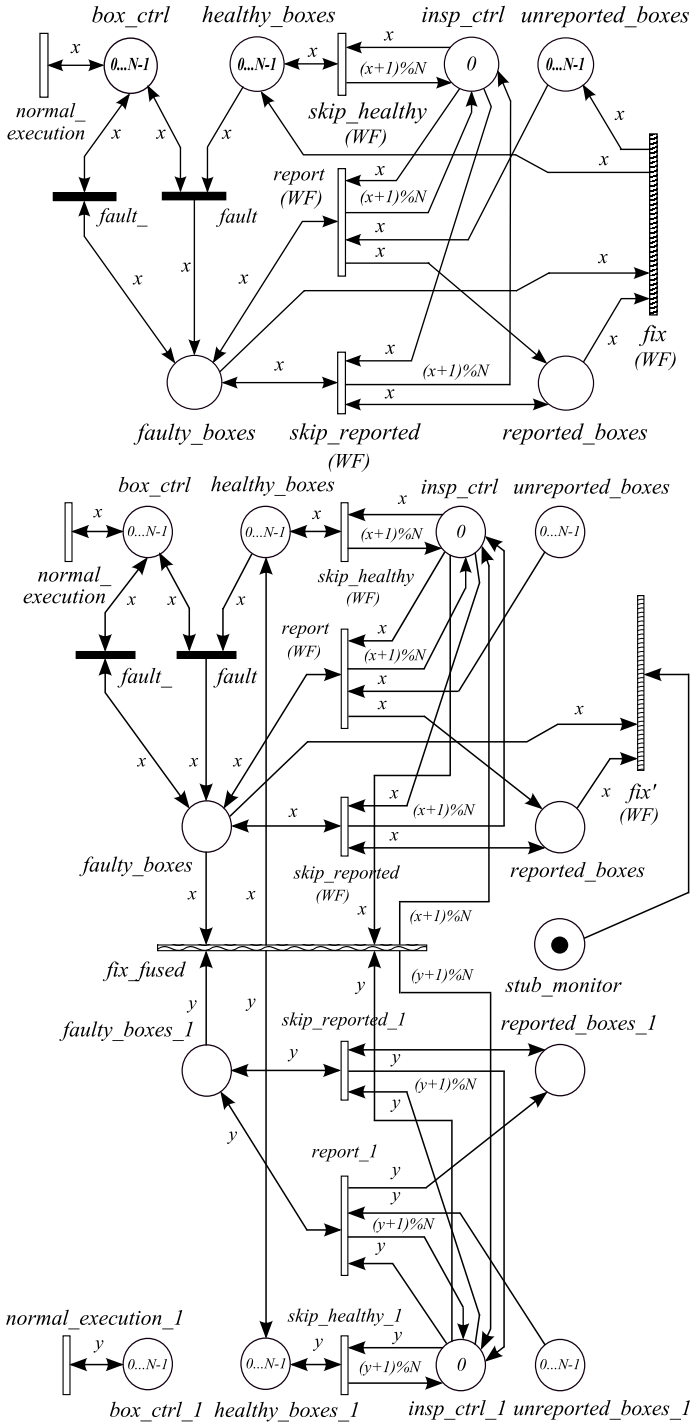

Fig. 10: The COMMBOXTECH (n) benchmark (top) and its verifier (bottom).

WF, the system becomes diagnosable, as after a fault the *fix* transition is eventually executed.

Benchmarks	Vrf Time	Vrf Modular Time
CommBox (4)	<1	<1
CommBox (5)	4	1
CommBox (6)	12	4
CommBox (7)	38	14
CommBoxTech (4)	17	6
CommBoxTech (5)	101	33
CommBoxTech (6)	561	162
CommBoxTech (7)	2995	Bug

Table I: Experimental results for CommBox and CommBoxTech benchmarks (all nets are diagnosable).

CommBoxTech *(n)*. Fig. 10 shows an elaborated version of the above system, together with its verifier: The inspector reports the faults to a technician, who then fixes them. Again, the inspector's and technician's transitions must both be WF to make the system diagnosable.

The experimental results are summarised in Table I, where the meaning of the columns is as follows (from left to right): name of the benchmark, verification time, and verification time using the modular representation of the verifier. (The time is measured in seconds.) All experiments were conducted on a PC with 64-bit Windows 7 operating system, an Intel Core i7 2.8GHz Processor with 8 cores and 4GB RAM (no parallelisation was used for the results in this table). The Maria tool has confirmed that the diagnosability property holds for these benchmarks. We also discovered a bug in Maria: for the CommBoxTech (7) benchmark there is a mismatch between the verification outcomes in the standard and modular modes.

We also wanted to check that the WF constraint is essential for diagnosability, i.e. that if even one transition is removed from the WF set then the system becomes undiagnosable. These results are summarised in Tables II and III. The Maria tool confirmed that this is the case for the transitions *skip_healthy* and *fix* for the CommBox family, and for the transitions *skip_healthy, report* and *fix* for the CommBoxTech

Benchmarks	WF enabled *skip_healthy*	WF enabled *fix*	Vrf Time	Vrf Modular Time	Diagnosable
CommBox (4)	✓	×	1	<1	×
	×	✓	1	<1	×
CommBox (5)	✓	×	1	1	×
	×	✓	2	1	×
CommBox (6)	✓	×	2	1	×
	×	✓	2	2	×
CommBox (7)	✓	×	2	2	×
	×	✓	3	2	×

Table II: Experimental results for CommBox with reduced WF set.

Benchmarks	WF enabled *skip_ healthy*	*report*	*skip_ reported*	fix	Vrf Time	Vrf Modular Time	Diagnosable
CommBox Tech(4)	✓	✓	✓	×	2	1	×
	✓	✓	×	✓	17	6	✓
	✓	×	✓	✓	1	1	×
	×	✓	✓	✓	8	3	×
CommBox Tech(5)	✓	✓	✓	×	3	2	×
	✓	✓	×	✓	102	30	✓
	✓	×	✓	✓	2	1	×
	×	✓	✓	✓	42	14	×
CommBox Tech(6)	✓	✓	✓	×	3	2	×
	✓	✓	×	✓	560	147	✓
	✓	×	✓	✓	2	1	×
	×	✓	✓	✓	6	61	×
CommBox Tech(7)	✓	✓	✓	×	4	3	×
	✓	✓	×	✓	2853	Bug	✓
	✓	×	✓	✓	3	2	×
	×	✓	✓	✓	1099	4	×

Table III: Experimental results for CommBoxTech with reduced WF set.

family. However, to our surprise, the CommBoxTech benchmarks remain diagnosable even when the *skip_reported* transition is removed from the WF set: This is in fact correct, as *skip_reported* can be enabled only after some fault has been reported, i.e. some fault will be diagnosed due to the *fix* transition even if *skip_reported* never fires.

6. GENERAL VERIFIER FOR BOUNDED LPNS

Let $\mathcal{N}$ be a bounded LPN whose fault transitions may or may not be WF. The consequences of having WF faults are exemplified by Fig. 4. Notice that in both $\mathcal{V}$ and $\mathcal{V}_{WF}$, the fault transitions are removed from the component corresponding to $\mathcal{N}'$. However, in Fig. 4, the fault transition is enabled in the initial marking and WF, hence unavoidable, so the LPN is trivially WF-diagnosable. However, both $\mathcal{V}$ and $\mathcal{V}_{WF}$ violate their respective LTL-X formulae in this example since they allow $\mathcal{N}'$ to ignore the permanently enabled WF fault transition.

To fix this problem, we extend the construction of $\mathcal{V}_{WF}$ as explained below. The generalised construction adds several places called *control monitors*. These will be added to the presets and postsets of certain transitions, which allows one to distinguish different phases of an execution. While such a transformation does not change the behaviour of a standard Petri net, the presence of weak fairness requires a more elaborated construction. To illustrate this issue, suppose t is WF and t' is another transition with ${}^\bullet t \cap {}^\bullet t' = \emptyset$. Thus, if t is enabled then firing t' does not discharge an infinite WF execution from the obligation to fire a transition from $({}^\bullet t)^\bullet$. However, if the same control monitor place is added to the presets of t and t', the set of WF executions can change. To solve this problem we introduce the following notion.

Definition 6.1. A *WF-preserving control monitor* mon *w.r.t. a set of transitions* T is a set of fresh places $\{mon_0\} \cup \{\, mon_t \mid t \in T \text{ is WF} \,\}$.

We use the following net operations involving control monitors.

(Simple) control. Let T be a set of transitions and p, p' be two control monitor places (which may be the same). To *control* T by (p, p') means adding p to the preset and p' to the postset of each $t \in T$.

WF-preserving control. Let T be a set of transitions and mon, mon' be two WF-preserving control monitors. We say that T is controlled by (mon, mon') if we control each non-WF transition in T by (mon_0, mon'_0) and each WF transition $t \in T$ by (mon_t, mon'_t).

Activity monitor. Let T be a set of transitions and $T_{WF} \stackrel{\text{df}}{=} \{t \in T \mid t \text{ is WF}\}$. An *activity monitor* is used to make sure that an execution contains infinitely many occurrences of transitions from T. Formally, an activity monitor a is a tuple of WF-preserving control monitors $(idle^a, active^a)$ that controls T, where all places of $idle^a$ are initially marked. Moreover, we add a number of fresh WF transitions: one that transfers a token from $active^a_0$ to $idle^a_0$, and one for each $t \in T_{WF}$ that transfers a token from $active^a_t$ to $idle^a_t$. With a we associate the LTL-X formula

$$\phi_a \stackrel{\text{df}}{=} \Box \Diamond \left(active^a_0 \vee \bigvee_{t \in T_{WF}} active^a_t \right)$$

expressing that an execution contains infinitely many occurrences of T.

The ideas behind the general verifier $\mathcal{V}^{gen}_{WF}$ capable of handling WF faults are as follows. It has three copies of the original LPN $\mathcal{N}$, corresponding to the executions σ, ρ, and the infinite fault-free WF continuation ρ' of $\hat{\rho}$ as in the proof of Lemma 3.4. The first two copies are transformed and synchronised as before and correspond to $\mathcal{N}^{ft}$ and $\mathcal{N}'$, to ensure that σ is WF, ρ is fault-free and $\ell(\sigma) = \ell(\rho)$. The LTL-X formula, as before, ensures that σ contains a fault. The third copy, $\mathcal{N}''$, initially follows $\mathcal{N}'$, in the sense that any transition modifying the marking of $\mathcal{N}'$ also modifies the marking of $\mathcal{N}''$ in the same way, so that the markings of $\mathcal{N}'$ and $\mathcal{N}''$ are the same for some time. Moreover, a separate set of places $\bar{P}$ corresponding to those in $\mathcal{N}'$ is created, and it is ensured in a similar way that the marking of $\bar{P}$ is the same as that of $\mathcal{N}'$ and $\mathcal{N}''$ for the same period of time. However, at some non-deterministically chosen point of time, $\mathcal{N}''$ starts running completely independently from $\mathcal{N}'$ and the marking of $\bar{P}$ stops changing (the projection of the verifier's execution up to this point to $\mathcal{N}'$ corresponds to $\hat{\rho}$). The non-WF fault transitions of $\mathcal{N}''$ are removed, and its WF fault transitions are turned into stubs, and the LTL-X formula will ensure that these stubs do not fire and the projection of the execution onto this LPN is WF. Moreover, the formula will ensure that ρ periodically passes through the marking stored in $\bar{P}$ as required by Lemma 3.4 (2). For this, we employ the LTL-X formula

$$\phi_{mark} \stackrel{\text{df}}{=} \Box \Diamond \bigwedge_{(p', \bar{p})} \#_{tok}(p') = \#_{tok}(\bar{p}),$$

where p' runs through the places of $\mathcal{N}'$ and $\bar{p}$ is the place of $\bar{P}$ corresponding to p'. Note that this formula uses elementary propositions checking whether two places contain the same number of tokens; such comparisons are supported by mainstream model checkers, e.g. MARIA. Lastly, two *activity monitors* are added to this construction, to check whether the projections of any execution of the verifier satisfying the formula to $\mathcal{N}'$ and $\mathcal{N}''$ are infinite (this is unlike the construction of $\mathcal{V}_{WF}$ in Sect. 4.3, where the infinity of projections followed automatically from the assumptions about $\mathcal{N}$). This implies the infinity of the projection to $\mathcal{N}^{ft}$, due to the assumptions about $\mathcal{N}$.

We now describe the verifier's construction in more detail:

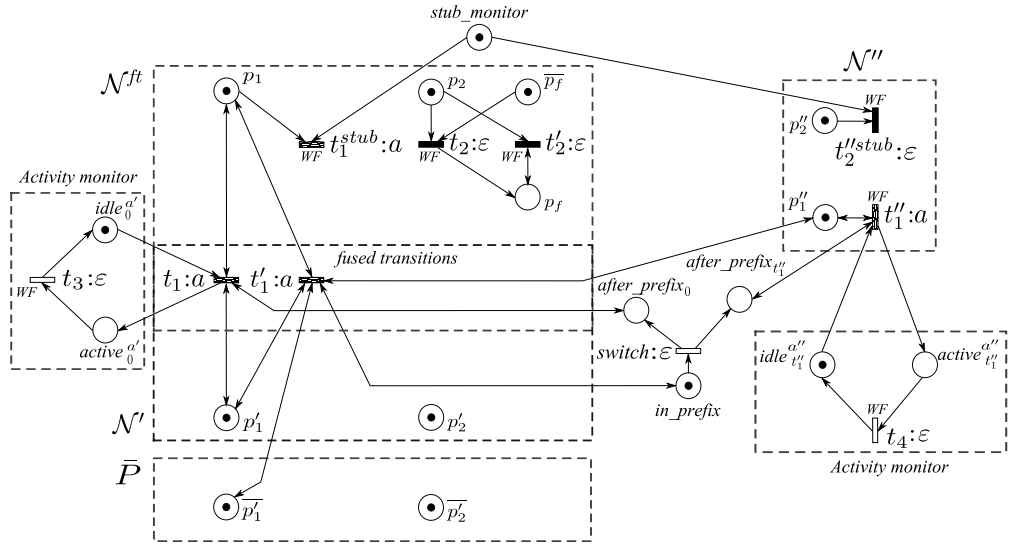

Fig. 11: The general verifier $\mathcal{V}_{WF}^{gen}$ capable of handling WF faults for the LPN in Fig. 4.

(1) Construct the net $\mathcal{V}_{WF}$ as in Sect. 4.3, containing the two copies $\mathcal{N}^{ft}$ and $\mathcal{N}'$ and a place $stub\_monitor$. Let $\mathcal{N}''$ be an additional copy of $\mathcal{N}$. Add a fresh set of places $\bar{P} \stackrel{\text{df}}{=} \{\, \bar{p} \mid p \text{ is a place of } \mathcal{N}' \,\}$.
(2) Remove from $\mathcal{N}''$ all non-WF fault transitions, and turn the WF fault transitions into stubs (re-using $stub\_monitor$); they remain WF.
(3) Add an initially marked control monitor place in_prefix and a WF-preserving monitor $after\_prefix$ w.r.t. the non-stub transitions T of $\mathcal{N}''$. Control the transitions in T by $(after\_prefix, after\_prefix)$. Moreover, add a fresh non-WF transition $switch$ with the preset $\{in\_prefix\}$ and the postset $after\_prefix$.
(4) Let T_{after} be the transitions of $\mathcal{N}'$, including the fused transitions. Add a set T_{in} containing a fresh non-WF copy t' for each $t \in T_{after}$ with the same label. For each place p of $\mathcal{N}'$ in ${}^\bullet t$ (resp. $t^\bullet$), add p and the corresponding places p'' of $\mathcal{N}''$ and $\bar{p} \in \bar{P}$ to the preset (resp. postset) of t'. Furthermore, T_{in} is controlled by (in_prefix, in_prefix) and T_{after} by $(after\_prefix_0, after\_prefix_0)$.
(5) Introduce an activity monitor a' that watches the fused transitions among T_{after}.
(6) Introduce an activity monitor a'' that watches all the visible transitions of $\mathcal{N}''$.
(7) Call the resulting net $\mathcal{V}_{WF}^{gen}$.

This construction is illustrated in Fig. 11.

As before, we formulate diagnosability of $\mathcal{N}$ as an LTL-X formula that needs to hold for the infinite WF executions of $\mathcal{V}_{WF}^{gen}$:

$$diag_{WF}^{gen} \stackrel{\text{df}}{=} \Box\, \overline{p_f} \vee \Diamond\, \neg stub\_monitor \vee \neg \phi_{mark} \vee \neg \phi_{a'} \vee \neg \phi_{a''}.$$

The negation of this formula is

$$\neg diag_{WF}^{gen} = \Diamond\, \neg \overline{p_f} \wedge \Box\, stub\_monitor \wedge \phi_{mark} \wedge \phi_{a'} \wedge \phi_{a''}.$$

A counterexample thus has to commit a fault in $\mathcal{N}^{ft}$ (but not in $\mathcal{N}'$ as fault transitions are removed from there), not execute any stub transitions, contain infinitely many synchronised transitions and infinitely many visible transitions of $\mathcal{N}''$. Additionally, the $\mathcal{N}'$ part of $\mathcal{V}_{WF}^{gen}$ must pass through the marking stored in $\bar{P}$ infinitely often.

THEOREM 6.2 (CORRECTNESS OF GENERAL WF VERIFIER). *A bounded LPN $\mathcal{N}$ is diagnosable iff the corresponding verifier $\mathcal{V}_{WF}^{gen}$ satisfies $diag_{WF}^{gen}$.*

PROOF. ($\Rightarrow$) Suppose $\mathcal{V}_{WF}^{gen}$ does not satisfy $diag_{WF}^{gen}$, i.e. it has a WF execution τ satisfying $\neg diag_{WF}^{gen}$. Hence τ satisfies $\phi_{a'}$ and thus contains infinitely many synchronised transitions. Let σ and ρ be the projections of τ to $\mathcal{N}^{ft}$ and $\mathcal{N}'$, respectively (the latter projection includes all fused transitions, including both T_{in} and T_{after}). We claim that (σ, ρ) is a witness. Indeed, as in the proof of Thm. 4.1 we obtain that σ is an infinite WF faulty execution of $\mathcal{N}$. It remains to prove that ρ satisfies the condition of Lemma 3.4 (2). Clearly, ρ is fault-free by construction. Also, $\phi_{a'}$ implies that τ contains $switch$. Let $\hat{\rho}$ be a finite prefix of ρ before the occurrence of $switch$, and m be the marking of $\mathcal{N}'$ reached by $\hat{\rho}$. (A copy of m is stored in $\bar{P}$ and is never changed after this point in time.) Then ρ goes through m infinitely often because τ satisfies ϕ_{mark}. Let ρ' be the projection of τ to the transitions of $\mathcal{N}''$. These are controlled by $after\_prefix$, hence describe an execution starting at m. That execution is infinite because $\phi_{a''}$ is satisfied; it is WF because τ is, and it is fault-free because no stub has fired. Thus, (σ, ρ) is indeed a witness.

($\Leftarrow$) Let (σ, ρ) be a witness according to Def. 3.2. According to Lemma 3.4, ρ goes infinitely often through some marking m, and a prefix $\hat{\rho}$ reaching m can be extended to an infinite WF fault-free execution $\hat{\rho}\rho'$. Such a witness can be transformed into an execution τ of $\mathcal{V}_{WF}^{gen}$ satisfying $\neg diag_{WF}^{gen}$ as follows. Prefix $\hat{\rho}$ is simulated by the transitions in T_{in}. Upon reaching m, the verifier can fire $switch$. Up to this point, $\mathcal{N}'$ and $\mathcal{N}''$ have the same marking, which is m. Then, transitions from T_{after} are used to simulate the continuation of $\hat{\rho}$ in ρ, and $\mathcal{N}''$ is used to simulate its infinite WF continuation ρ'. By assumption, σ and ρ' are infinite and WF, hence so is τ; thus no stub needs to be fired, and $\phi_{a'} \wedge \phi_{a''}$ are satisfied. Since σ contains a fault, τ satisfies $\Diamond \neg \overline{p_f}$. Also, ρ passes through m infinitely often, thus satisfying ϕ_{mark}. Hence $diag_{WF}^{gen}$ is violated by τ. □

7. CONCLUSIONS

In this paper we have identified a major flaw in the previous definition of WF-diagnosability in the literature, and proposed a corrected notion. Moreover, under a simplifying assumption that the fault transitions are non-WF, we have presented an efficient technique for verifying WF-diagnosability based on a reduction to LTL-X model checking. An important advantage of this method is that the LTL-X formula is fixed — in particular, the WF assumption does not have to be expressed as a part of it (which would make the formula length proportional to the size of the specification), but rather the ability of existing model checkers to handle weak fairness directly is exploited. Furthermore, the construction has been generalised to arbitrary bounded LPNs.

We also created two families of scalable benchmarks, where the weak fairness is essential for diagnosability. The proposed WF-diagnosability verification method has been tested on these benchmarks, and the experimental results demonstrate its feasibility in practice.

REFERENCES

A. Agarwal, A. Madalinski, and S. Haar. 2012. Effective Verification of Weak Diagnosability. In *Proc. SAFEPROCESS'12*. IFAC. DOI:http://dx.doi.org/10.3182/20120829-3-MX-2028.00083

S. Biswas. 2013a. Equivalence of Fair Diagnosability and Stochastic Diagnosability of Discrete Event Systems. In *Proc. IEEE SMC*. 378–383.

S. Biswas. 2013b. Fair Diagnosability in PN-Based DES Models. In *Proc. ICCA*. 378–383.

S. Biswas, D. Sarkar, S. Mudhopadhyay, and A. Patra. 2010. Fairness of transitions in diagnosability analysis of discrete event systems. *Discrete Event Dynamic Systems: theory and applications* 20, 3 (September 2010), 349–376.

M.P. Cabasino, A. Giua, S. Lafortune, and C. Seatzu. 2012. A New Approach for Diagnosability Analysis of Petri Nets Using Verifier Nets. *IEEE Trans. on Automatic Control* 57, 12 (December 2012), 3104–3117.

V. Germanos, S. Haar, V. Khomenko, and S. Schwoon. 2014. Diagnosability under Weak Fairness. In *Proc. ACSD'14*, A. Mokhov and L. Bernardinello (Eds.). IEEE Computing Society Press, 132–141.

S. Haar, C. Rodríguez, and S. Schwoon. 2013. Reveal Your Faults: It's Only Fair! In *Proc. ACSD'13*. IEEE Computer Society Press, 120–129. DOI:http://dx.doi.org/10.1109/ACSD.2013.15

S. Jiang, Z. Huang, V. Chandra, and R. Kumar. 2001. A polynomial algorithm for testing diagnosability of discrete event systems. In *IEEE Trans. on Autom. Control*.

L. Lamport. 1983. What good is temporal logic? In *Proc. IFIP Congr.'83*. Elsevier, 657–668.

A. Madalinski and V. Khomenko. 2010. Diagnosability Verification with Parallel LTL-X Model Checking Based on Petri Net Unfoldings. In *Proc. SysTol'10*. IEEE Computer Society Press, 398–403.

A. Madalinski, F. Nouioua, and P. Dague. 2010. Diagnosability verification with Petri net unfoldings. *KES Journal* 14, 2 (2010), 49–55. DOI:http://dx.doi.org/10.3233/KES-2010-0191

M. Mäkelä. 2005. MARIA: The Modular Reachability Analyzer. (2005). URL: `http://www.tcs.hut.fi/Software/maria/index.en.html`.

A. Pnueli. 1977. The Temporal Logic of Programs. In *Proc. FOCS'77*. 46–57.

M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. 1995. Diagnosability of Discrete Events Systems. *IEEE Trans. on Autom. Control* 40, 9 (1995), 1555–1575.

A. Schumann and Y. Pencolé. 2007. Scalable diagnosability checking of event-driven systems. In *Proc. IJCAI'07*. 575–580.

D. Thorsley and D. Teneketzis. 2005. Diagnosability of Stochastic discrete event systems. *IEEE Trans. Automat. Control* 50, 4 (2005), 476–492.

W. Vogler. 1995. Fairness and Partial Order Semantics. *Inf. Process. Lett.* 55, 1 (1995), 33–39.

T.-S. Yoo and S. Lafortune. 2002. Polynomial-Time Verification of Diagnosability of Partially Observed Discrete-Event Systems. *IEEE Trans. on Autom. Control* 47, 9 (2002), 1491–1495.