RAPPORT DE RECHERCHE

L R I

DIAGNOSABILITY VERIFICATION WITH PETRI NET UNFOLDINGS

MADALINSKI A / NOUIOUA F / DAGUE P

Unité Mixte de Recherche 8623
CNRS-Université Paris Sud –LRI

03/2009

Rapport de Recherche N° 1516

CNRS – Université de Paris Sud
Centre d'Orsay
LABORATOIRE DE RECHERCHE EN INFORMATIQUE
Bâtiment 490
91405 ORSAY Cedex (France)

Diagnosability verification with Petri net unfoldings

Agnes Madalinski, Farid Nouioua and Philippe Dague
LRI, Univ. Paris-Sud, CNRS, Parc Club Orsay Université,
4 rue Jacques Monod, bât. G, Orsay, 91893, France
{agnes.madalinski, farid.nouioua, philippe.dague}@lri.fr

***Abstract*—Today's complex systems increasingly require safety and robustness w.r.t faults occurrences, and diagnosability is a key property to ensure this at design stage. Intuitively, a system is diagnosable if its only observable part allows one to determine without ambiguity the occurrence of its failures. In the recent years diagnosability has been extensively studied, especially in the finite state machine (FSM) based models. Among the developed approaches in this domain the *twin plant* method has been proven to be efficient in verifying diagnosability. However, little work has been done in other discrete event systems such as Petri nets (PNs). PNs and their techniques have been proven to overcome some limitations of FSM, especially the state space explosion problem. One promising technique is the PN unfolding, which has been successfully applied to model checking. It offers a compact representation of the state space because runs are represented by means of partial orders rather than sequences. Therefore, we demonstrate in this paper how PN unfoldings can be applied to verify diagnosability by adapting the *twin plant* method.**

I. Introduction

Diagnosability is an important property that determines the ability of a system to detect faults occurrences given only observable sequences (the system has observable events and unobservable events including faults). If a system is diagnosable the diagnosis will find an accurate explanation for any possible set of observations from the system, otherwise the diagnosis will give an ambiguous and useless explanation.

The seminal work in [17] has introduced a formal language framework for diagnosis and analysis of diagnosability properties of discrete event systems (DES) represented by finite automata. The proposed method for diagnosability verification is based on the construction of a *diagnoser*: an automaton with only observable events which allows one to estimate states of the system after observation of sequences. Other methods with polynomial complexity (the previous one is exponential in the number of states) have been proposed and are based on the *twin plant* method [9], [19]. The basic idea is to build a *verifier* from a diagnoser by constructing the synchronous product of the diagnoser with itself on observable events. The verifier compares every pair of paths in the system that have the same observable behaviour. Adaptations of these methods have been also proposed to deal with the distributed case [15], [18], where the modularity of the system is used to compute local twin plants and to verify the diagnosability of the system by gradually combining local twin plants (in the worst case building the global twin plant).

Naturally, the state-based twin plant method suffers from the state space explosion problem. To alleviate this problem Petri net (PN) unfolding techniques appear promising. A finite and complete prefix of a PN unfolding gives a compact representation of all behaviours and reachable markings of this PN in a partial order. Executions are considered as partial ordered set of events rather than sequences, which results in memory savings. Since introducing the unfolding prefixes in [14] they have been improved [5], [12], extended and applied to various practical applications such as distributed diagnosis [6], model checking (see e.g.[10]), synthesis of asynchronous circuits [11] or planning problems [1]. Also the problem of diagnosability has been studied in this context from a purely theoretical point of view: [8] proposes a definition of diagnosability based on observable partial orders and, opposed to such quantitative criteria, a qualitative notion specific to partial orders has been introduced in [7]. However, the main difference between their definition and that proposed in this paper lies on the granularity level of observations. The one proposed in [8] stays at the partial order level, i.e. any execution of a partial order corresponds to the same observation, whereas in this work the different executions of a partial order correspond to different observations.

The objective of this paper is to use PN unfolding prefixes to verify diagnosability by adapting the twin plant method [19]; with the long term objective to develop an approach to verify diagnosability in a distributed way in the framework of modular complete prefixes [13]. The system is modelled as a labelled Petri net, where transitions are labelled with observable and unobservable events. The diagnosability property is tested using a finite and complete prefix of a verifier. The verifier is obtained by the synchronous product of a diagnoser (the system enriched with information about the occurrence of faults). A necessary and sufficient condition for diagnosability is given. In addition, two algorithms are given to test diagnosability; one performs an exhaustive search and reports in the case the system is not diagnosable all ambiguous explanations, and the other is designed to stop at the first ambiguous explanation if it exists and simply report the "binary" result, i.e. if the system is diagnosable or not. Moreover, two improvements applicable for both algorithms are presented, which exploit the symmetry and "interesting" behaviour of the verifier to reduce its size.

The paper is organised as follows. The second section recalls basic theoretical background concerning PNs, their unfoldings and the derivation of finite and complete prefixes. The third section describes the used model and introduces the twin plant method applied to PN unfolding prefixes. The fourth section presents the algorithms used to verify diagnosability and some improvements. The last section draws conclusions

and presents future perspectives.

II. Petri nets and their unfoldings

This section presents basic definitions concerning Petri nets and their unfoldings mainly adapted from [5], [10], [12].

A. Petri nets

A *Petri net* is a quadruple $N = (P, T, \rightarrow, M^0)$ such that P and T are disjoint sets of *places* and *transitions*, respectively, $\rightarrow \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*, and M^0 is the *initial marking*, where a marking is a function $P \rightarrow \mathbb{N} = \{0, 1, 2, ...\}$ which assigns a number of *tokens* to each place. A Petri net can be considered as a bipartite graph with directed edges between places and transitions. For a node $x \in P \cup T$, its *pre-set* ${}^\bullet x$ is defined by ${}^\bullet x = \{y \mid (y, x) \in \rightarrow\}$ and its *post-set* $x^\bullet$ is defined by $x^\bullet = \{y \mid (x, y) \in \rightarrow\}$. A transition t is *enabled* at a marking M if $\forall p \in {}^\bullet t : M(p) \geq 0$, which is denoted by $M[t\rangle$. An enabled transition can *fire* yielding a new marking $M' = M - {}^\bullet t + t^\bullet$, which is denoted by $M[t\rangle M'$. A transitions sequence $\sigma = t_1, ..., t_k \in T$ is a *firing sequence* from M_1 to M_{k+1}, denoted by $M_1[\sigma\rangle M_{k+1}$ or $M_1[\sigma\rangle$, iff a set of markings $M_2, ..., M_{k+1}$ exit such that $M_i[t_i\rangle M_{i+1}, 1 \leq i \leq k$. A net N is *safe* if for every reachable marking M and every place $p \in P$, $M(p) \subseteq \{0, 1\}$. A safe N has a finite number of reachable markings. An example of a safe Petri net with the initial marking $M^0 = \{p_1, p_2\}$ is illustrated in Figure 1(a).

Two nodes of a net N, y and y', are in *structural conflict*, denoted by $y \# y'$, if there exist distinct transitions $t, t' \in T$ such that ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$, and (t, y) and (t', y') are in the reflexive transitive closure of the flow relation $\rightarrow$, denoted by $\preceq$.

B. Occurrence nets

An *occurrence net* is a net $ON = (C, E, \rightarrow, C^0)$, where C is a set of *conditions* (places), E is the set of *events* (transitions) and $C^0 = \{c \in C : {}^\bullet c = \emptyset\}$ is the set of initial conditions satisfying the following: for every $c \in C$, $| {}^\bullet c | \leq 1$; for every $y \in C \cup E$, $\neg(y \# y)$ and there are finitely many y' such that $y' \prec y$, where $\prec$ denotes the *causal relation*, the transitive irreflexive closure of $\rightarrow$. Two nodes are *concurrent*, denoted $y \parallel y'$, if neither $y \# y'$ nor $y \preceq y'$ nor $y' \preceq y$.

Branching processes: A *branching process* (BP) of a net system N is a pair $\beta = (ON, h)$, where morphism $h : ON \rightarrow N$ is a total function on ON, also called a *folding* of ON into N. This folding can be seen as a labelling function on events and conditions of ON, by which configurations of ON represent runs of N. It is further required that β satisfies a parsimony condition: for all $e_1, e_2 \in E$, if ${}^\bullet e_1 = {}^\bullet e_2$ and $h(e_1) = h(e_2)$ then $e_1 = e_2$. To define finite complete prefixes, it will be useful to consider a (virtual) initial event $\perp$ in β, which has an empty preset, C^0 as post-set and no label (i.e. no image by h). An example of a Petri net and one of its branching processes is shown in Figure 1, where the morphism h is indicated by the labels of the nodes.

A branching process $\beta' = (ON', h')$ of N is a *prefix* of a branching process β, denoted by $\beta' \sqsubseteq \beta$, if ON' is a causally closed sub-net of ON containing all initial conditions and such that: $\forall e \in E, e \in ON'$ implies $e^\bullet \in ON'$ and h' is the restriction of h to $C' \cup E'$. For each net system N there exists a unique (up to isomorphism) maximal (w.r.t $\sqsubseteq$) branching process denoted by $Unf(N)$, or for short Unf_N, called the *unfolding* of N.

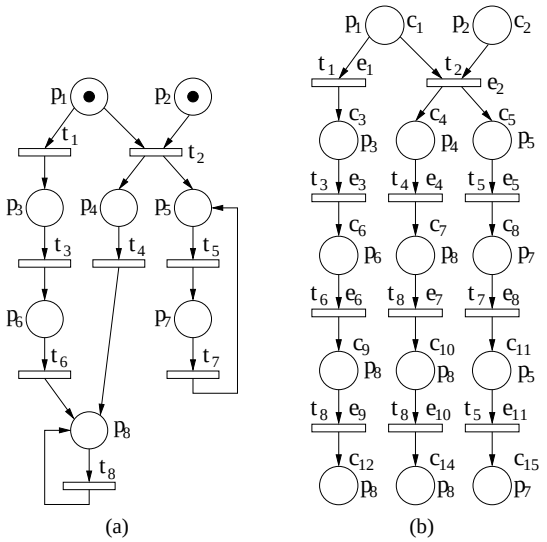


Figure 1: A Petri net (a) and one of its branching processes (b).

Configurations and cuts: A *configuration* of a branching process β is a finite set of events $\kappa \subseteq E$ such that for all $e, e' \in \kappa$, $\neg(e \# e')$ and, for every $e \in \kappa$, $e' \prec e$ implies $e' \in \kappa$; in addition it is required that $\perp \in \kappa$. For every event $e \in E$, the configuration $[e] \stackrel{\text{df}}{=} \{e' | e' \preceq e\}$ is called the *basic configuration*[1] of e, and $\langle e\rangle \stackrel{\text{df}}{=} [e] \setminus \{e\}$ denotes the set of *causal predecessors*. Moreover, for a set of events E' we denote by $\kappa \oplus E'$ the fact that $\kappa \cup E'$ is a configuration and $\kappa \cap E' = \emptyset$. Such an E' is a *suffix* of κ, and $\kappa \oplus E'$ is an extension of κ. The set of all finite (resp. basic) configurations of a branching process β is denoted by $\boldsymbol{\kappa}^\beta_{fin}$ (resp. $\boldsymbol{\kappa}^\beta_{bas}$), and the superscript β is dropped when $\beta = Unf_N$.

A *co-set* is a set of conditions C' such that for all distinct $c, c' \in C'$, $c \parallel c'$, and a *cut* is a maximal co-set for the set inclusion. Let κ be a configuration then $Cut(\kappa) \stackrel{\text{df}}{=} \left(C^0 \cup \kappa^\bullet\right) \setminus {}^\bullet\kappa$ is a cut; furthermore, the set of places $h(Cut(\kappa))$ is a reachable marking of N, which is denoted by $Mark(\kappa)$. A marking M of N is *represented* in β if there is a configuration κ of β such that $M = Mark(\kappa)$. Every marking represented in β is reached in N, and every reachable marking of N is represented in Unf_N. For a branching process β of N a *possible extension* is a pair (t, B), where B is a co-set in β and t is a transition of N such that $h(B) = {}^\bullet t$ and β contains no t-labelled event with preset B.

For example, in Figure 1(b) the basic configuration $[e_4] = \{e_2, e_4\}$ with the cut $Cut([e_4]) = \{c_5, c_7\}$ and the corresponding marking $Mark([e_4]) = \{p_5, p_8\}$. A possible extension of this configuration is $(t_5, \{p_5\})$ or $(t_8, \{p_8\})$. Note that neither

[1] also called *local configuration* in literature.

$\{e_1, e_2\}$ nor $\{e_3, e_6\}$ are configurations since the former includes events in conflicts $e_1 \# e_2$, and the latter does not include e_1 since $e_1 \prec e_6$.

C. Finite and complete prefixes

Although unfoldings are infinite whenever the original net has infinite runs they can be truncated in such a way that the resulting finite prefix contains enough information to decide a certain behavioural property. If this is satisfied a prefix is said to be *complete* for that property. There exist different methods to truncate unfoldings depending which kind of information should be preserved of the unfolding in the prefix and other aspects related to the construction of the prefix.

Definition 1: A *cutting context* for Unf_N is a triple $\Theta = \left(\approx, \lhd, \{\boldsymbol{\kappa}_e\}_{e \in E}\right)$, where:

1) $\approx$ is an equivalence relation on $\boldsymbol{\kappa}_{fin}$.
2) $\lhd$, called an *adequate* order, is a strict well-founded partial order on $\boldsymbol{\kappa}_{fin}$ refining $\subset$, i.e. $\kappa' \subset \kappa''$ implies $\kappa' \lhd \kappa''$.
3) $\approx$ and $\lhd$ are *preserved by finite extensions*, i.e. for every pair of configurations $\kappa' \approx \kappa''$, and for every suffix ϵ' of κ', there exists a finite suffix ϵ'' of κ'' such that
   a) $\kappa'' \oplus \epsilon'' \approx \kappa' \oplus \epsilon'$, and
   b) if $\kappa'' \lhd \kappa'$ then $\kappa'' \oplus \epsilon'' \lhd \kappa' \oplus \epsilon'$.
4) $\{\boldsymbol{\kappa}_e\}_{e \in E}$ is a family of subsets of $\boldsymbol{\kappa}_{fin}$.

The first parameter determines the information intended to be preserved in the complete prefix; the second parameter specifies which configurations are preserved in the complete prefix; the last parameter $\boldsymbol{\kappa}_e$ is needed to specify the set of configurations used to decide whether an event can be designed as a cut-off event (in practise, $\boldsymbol{\kappa}_e$ only contains basic configurations for efficiency reasons). There exist several equivalence relations and adequate orders (e.g. see [10]). The cutting context $\Theta_{ERV} = \left\{\approx_m, \lhd_{tot}, \{\boldsymbol{\kappa}_e = \boldsymbol{\kappa}_{bas}\}_{e \in E}\right\}$ corresponds to the framework in [5], where $\approx_m$ is the equivalence relation on reachable markings of N, i.e. $\kappa' \approx_m \kappa''$ iff $Mark(\kappa') = Mark(\kappa'')$, and $\lhd_{tot}$ is a total adequate order.

Definition 2: The set of *feasible events*, denoted by $fsble^{\Theta}$, and the set of *static cut-off events*, denoted by $Ecut^{\Theta}$, are two sets of events of Unf_N defined inductively, in the following way:

1) An event e is a feasible event if $\langle e \rangle \cap Ecut^{\Theta} = \emptyset$.
2) An event e is a static cut-off event if it is feasible, and if there is a (so called *corresponding*) configuration $\kappa \in \boldsymbol{\kappa}_e$ such that $\kappa \subseteq fsble^{\Theta} \setminus Ecut^{\Theta}$, $\kappa \approx [e]$, and $\kappa \lhd [e]$. (An event e' is referred as *corresponding event* if $\kappa = [e']$.)

The branching process $Pref_N^{\Theta}$ induced by the set of events $fsble_N^{\Theta}$ is called the *canonical prefix* of Unf_N.
Note that $Pref_N^{\Theta}$ is uniquely determined by the cutting context Θ. Several fundamental properties of $Pref_N^{\Theta}$ have been proven in [12]. In particular, $Pref_N^{\Theta}$ is always complete w.r.t. $Ecut_N^{\Theta}$, and it is finite if $\approx$ has finitely many equivalence classes and $\boldsymbol{\kappa}_e \supseteq \boldsymbol{\kappa}_{bas}$. The prefix induced by the events $e_1 - e_9$ in Figure 1(b) is a canonical prefix $Pref_N^{\Theta_{ERV}}$ with the set of cut-off events $Ecut_N^{\Theta_{ERV}} = \{e_7, e_8, e_9\}$.

III. Diagnosability

The system is modelled as a labelled Petri net, where transitions are labelled with observable and unobservable events. The twin plant method [9] is applied to the framework of Petri net unfoldings. This involve to build a finite and complete prefix of a verifier, where the diagnosability property is tested. The verifier is obtained by the synchronous product of a diagnoser (the system enriched with information about the occurrence of faults). A necessary and sufficient condition for diagnosability is given.

A. System model

The system is modelled with a safe *labelled* Petri net

$$\mathcal{N} = (N, O, U, \ell), \tag{1}$$

which is a Petri net N extended with sets of *observable* and *unobservable* transition labels O and U, respectively, and a labelling function $\ell : T \to O \cup U$ on transitions. The observable transitions correspond to controller commands, sensor readings and their changes, and in contrast, unobservable transitions correspond to some internal events that cause changes in the system not recorded by sensors. The set of *fault* transition labels $F \subseteq O \cup U$ and it is assumed that $F \subseteq U$ since it is trivial to diagnose fault transitions that are observable. Moreover, $F = F_1 \cup ... \cup F_n$ is partitioned into disjoint sets, where F_i denotes the set of fault transitions corresponding to a fault type i such that $1 \leq i \leq n$ and n is the number of fault types. This allows one to handle subsets of faults if it is not necessary to detect uniquely every fault transition. An example of a system is illustrated in Figure 2(a) with highlighted set of observable transitions labelled with $O = \{a, b, c\}$, and the set of unobservable transitions labelled with $U = \{u, f_1, f_2\}$ including $F = F_1 \cup F_2$, where $F_1 = \{f_1\}$ and $F_2 = \{f_2\}$.

The labelled Petri net $\mathcal{N}$ inherits the operational semantics of the underlying net N. One has $M\,[\ell(t)\rangle M'$ if $M\,[t\rangle M'$. Moreover, a firing sequence $\sigma \in O \cup U$ is called a *trace* of $\mathcal{N}$ if $M\,[\sigma\rangle$. The language of a labelled Petri net $\mathcal{N}$ is the set of all traces of $\mathcal{N}$ and is denoted by $L(\mathcal{N})$.

The set of markings reachable from M^0 can be represented as a *reachability graph*, an edge-labelled directed graph on reachable markings with M^0 as root and edges from M to M' labelled $\ell(t)$ if $M\,[t\rangle M'$ with $L(\mathcal{N})$. It can be regarded as a finite automaton (where all states are accepting) with the language $L(\mathcal{N})$. It is assumed that $L(\mathcal{N})$ is live. We denote by $|\sigma|$ the length of a trace σ, by $Obs(\sigma)$ the observable trace produced by any trace σ of $L(\mathcal{N})$, and by $Obs^{-1}(\sigma)$ the set of sequences of $L(\mathcal{N})$ sharing with σ the same observable traces.

Definition 3: Let f_i stand for any fault in F_i. Let σ_{f_i} be a trace ending with a transition labelled f_i, i.e. $M^0\,[\sigma_{f_i}\rangle M$ for some reachable marking M, and let σ'_{f_i} be any continuation trace of σ_{f_i}, i.e. $M\left[\sigma'_{f_i}\right\rangle M'$ for some other reachable marking M'. The fault f_i is diagnosable iff:

$$(\exists n \in \mathbb{N})\left(\forall \sigma_{f_i}\sigma'_{f_i}\right)\left[\left|\sigma'_{f_i}\right| \geq n \Rightarrow D\right],$$

where D is: $\forall \omega \in Obs^{-1}\left(\sigma_{f_i}\sigma'_{f_i}\right) \Rightarrow f_i \in \omega$.

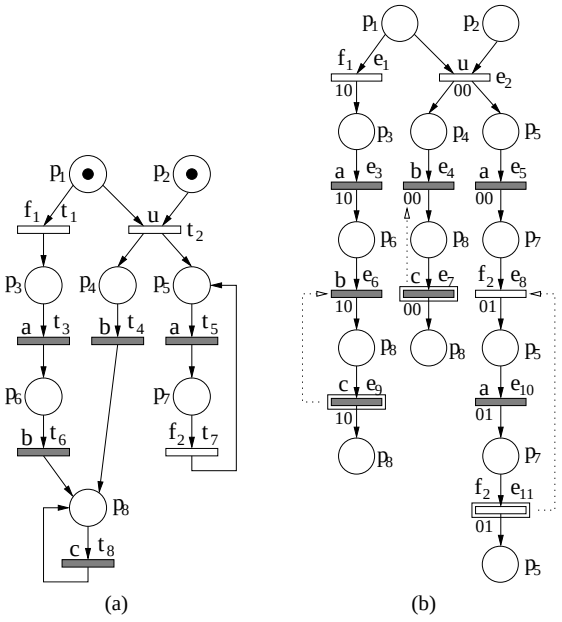


Figure 2: The system $\mathcal{N}$ (a) and the canonical prefix of the diagnoser D (b).

This diagnosability definition is based on the one introduced in [17].

B. Diagnoser

In the diagnoser[2] only observable transition are visible and unobservable transitions can be seen as silent transitions. Thus, in order to keep track of the occurrences of fault types an additional piece of information is needed. Let $\nu : F \to \{0,1\}$ be a *fault label function* then to each marking M of $\mathcal{N}$ is associated a fault label as follows:

- $\nu^0 = (0,...,0)$, where ν^0 is the fault label of M^0
- $M^i\left[\ell\left(t\right)\right\rangle M^{i+1} \Rightarrow \nu_j^{i+1} = \begin{cases} 1 & \text{if } \nu_j^i = 1 \vee \ell\left(t\right) \in F_j \\ 0 & \text{otherwise} \end{cases}$, $1 \leq j \leq n$

Hence, a *diagnoser state* of $\mathcal{N}$ is a pair (M,ν), where M is a marking of $\mathcal{N}$ and ν is the fault label associated with M. Let $Fault((M,\nu)) \stackrel{\text{df}}{=} \nu$. The diagnoser[3] can be represented as

$$D = (\mathcal{N}, \nu^0) \tag{2}$$

[2]The diagnoser used here is not the same as in the original work of [17]. In both cases it describes the behaviour of the system w.r.t. the observable events by introducing information about faults in the states. The diagnoser defined in [17] is always deterministic and a state may involve different states of the system with different fault labels. This means that after the observation of a given trace the system can be in one of these different states with the corresponding fault labels. The diagnoser defined here can be non-deterministic and a state is represented by only one state of the system with the corresponding fault label. This means that after the observation of a given trace the system is in this state with the corresponding fault label.

[3]Note that a diagnoser can be seen as a Signal Transition Graph (STG) [16], a labelled Petri net used to specify asynchronous circuits. Both have labelled transitions, although labelled in different context (the STG's transitions correspond to input and output signals of a circuit), and both extend their states with additional information (the STG state is extended with a signal coding function). Thus, it is natural to adapt definitions and notions of STGs and their canonical prefixes from [10] to diagnosers.

with $(M,\nu)\left[\ell\left(t\right)\right\rangle(M',\nu')$ if $M\left[\ell\left(t\right)\right\rangle M'$ and ν,ν' are the fault labels associated to M and M', respectively. Note that the number of states of a diagnoser grows by 2^n compared with $\mathcal{N}$.

A branching process of a labelled Petri net $\mathcal{N}$ is a branching process of N augmented with additional labelling of its events $\ell \circ h : E \to O \cup U$. The function "Fault" is extended to configuration of the branching process of D with $Fault(\kappa) \stackrel{\text{df}}{=} Fault(Mark(\kappa))$. In order to build a canonical prefix of D the equivalence relation has to be adapted to diagnoser states; $\approx_\nu$ is defined as follows $\kappa \approx_\nu \kappa'$ iff $Mark(\kappa) = Mark(\kappa')$ and $Fault(\kappa) = Fault(\kappa')$. Then, a canonical prefix $Pref_D^{\Theta^{tot}}$ can be obtained with $\Theta^{tot} = \left\{\approx_\nu, \lhd_{tot}, \{\boldsymbol{\kappa}_e = \boldsymbol{\kappa}_{bas}\}_{e\in E}\right\}$, where $\approx_\nu$ is the equivalence relation on diagnoser states, and $\lhd_{tot}$ is a total adequate order [5]. The diagnoser D and its canonical prefix $Pref_D^{\Theta^{tot}}$ are depicted in Figure 2 (in figures the cut-off events are drawn as double boxes, and dotted lines indicate the corresponding events of cut-off events; in addition, to each event e is associated a fault vector of $[e]$, which is shown next to events). Observe that in a canonical prefix of $\mathcal{N}$ the event e_8 would be a cut-off event with the corresponding event e_2 since only the relation $\approx_m$ is considered. However, their codes are different and therefore e_8 cannot be designated as a cut-off event of a diagnoser; the prefix has to be extended until the cut-off event e_{11}, which has a corresponding event e_8 ($[e_8] \approx_\nu [e_{11}]$ and $[e_8] \lhd_{tot} [e_{11}]$).

C. Synchronous product

The synchronous product of two labelled Petri nets on common observable events is presented. From the language theory point of view a language generated by a product is the intersection of the languages generated by the two synchronised nets. In this case it is the intersection of the projection of observable language/words.

Definition 4: Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be two labelled Petri nets their product $\mathcal{N} = \mathcal{N}_1 \times \mathcal{N}_2$ is defined as

$$\begin{aligned}
P &= \{(p_1,\star) : p_1 \in P_1\} \cup \{(\star,p_2) : p_2 \in P_2\} \\
T &= \{(t_1,t_2) : t_1 \in T_1, t_2 \in T_2, \\
&\quad \ell_1\left(t_1\right) = \ell_2\left(t_2\right) \in O_1 \cap O_2\} \\
&\quad \cup \{(t_1,\star) : t_1 \in T_1, \ell_1\left(t_1\right) \notin O_1 \cap O_2\} \\
&\quad \cup \{(\star,t_2) : t_2 \in T_2, \ell_2\left(t_2\right) \notin O_1 \cap O_2\} \\
\to &= \begin{cases} {}^\bullet(t_1,t_2) = {}^\bullet t_1 \cup {}^\bullet t_2 \\ {}^\bullet(t_1,\star) = {}^\bullet t_1 \quad \text{for } t_1 \in T_1, t_2 \in T_2 \\ {}^\bullet(\star,t_2) = {}^\bullet t_2 \end{cases} \\
M^0 &= M_1^0 \cup M_2^0 \\
O &= O_1 \cup O_2 \\
U &= U_1 \cup U_2 \\
\ell\left(t_1,t_2\right) &= \begin{cases} \ell_1\left(t_1\right) & \text{if } t_1 \in T_1 \\ \ell_2\left(t_2\right) & \text{if } t_2 \in T_2 \end{cases}
\end{aligned}$$

where $\star$ is a dummy element. The flow relation is defined symmetrical for post-sets of transitions.

In the product the places of each component are preserved; the transitions which have a common observable label are synchronised by merging common transitions, otherwise the transitions of each component are also preserved. The firing $(M_1,M_2)\left[\ell\left((t_1,t_2)\right)\right\rangle(M_1',M_2')$ of $\mathcal{N}$ corresponds to firings $M_1\left[\ell_1\left(t_1\right)\right\rangle M_1'$ in $\mathcal{N}_1$ and $M_2\left[\ell_2\left(t_2\right)\right\rangle M_2'$ in $\mathcal{N}_2$, where the firing of $\star$ means that an empty transition sequence fires.

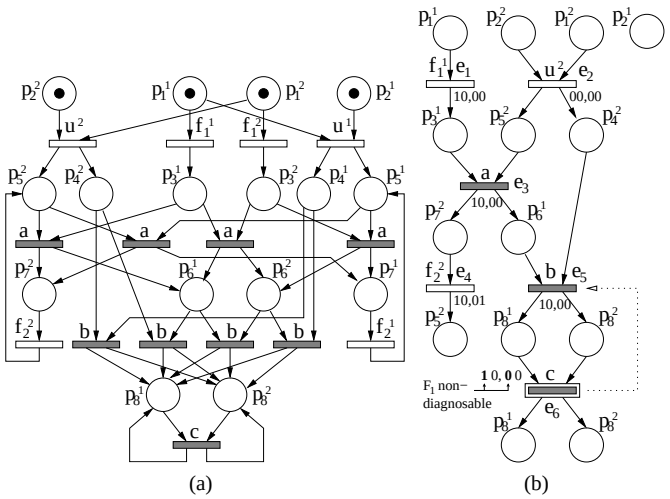


Figure 3: Petri net of the verifier V (a) and part of its canonical prefix showing non-diagnosable traces (b).

Figure 3(a) shows the product of the running example (Figure 2(a)) with itself. For simplicity, a place is denoted with p^1 for $(p, \star)$ and with p^2 for $(\star, p)$, this also applies to non-synchronised transitions, where their labels are depicted. For synchronised transitions only their labels are used. Observe that there are four transitions labelled with a (and also with b). This is due to the fact that t_3 and t_5 are labelled with a in the original net, and in the product they synchronise to $(t_3, t_3), (t_3, t_5), (t_5, t_3)$ and (t_5, t_5) (and similar applies to transitions labelled with b). Note that in general, some transitions in the product may never be enabled, however, in this example this is not the case.

D. Verifier

In order to check the diagnosability property a verifier is build from the synchronous product of the diagnoser with itself. The *verifier*

$$V = D \times D = \left(\mathcal{N} \times \mathcal{N}, \left(\nu^0, \nu^0\right)\right), \tag{3}$$

where $D = (\mathcal{N}, \nu^0)$ is a diagnoser, $\mathcal{N}$ is the system model and $\left(\nu^0, \nu^0\right)$ is the synchronised fault label. In the sequel V is denoted as $\left(\mathcal{N}, \mathcal{N}, \nu^0, \nu^0\right)$ for the sake of simplicity. Thus, V can be regarded as a diagnoser net (it has the same dynamics as a diagnoser net). The canonical prefix of the verifier V, $Pref_V^\Theta$, is used with $\Theta = \left\{\approx_\nu, \subset, \{\boldsymbol{\kappa}_e = \boldsymbol{\kappa}_{bas}\}_{e \in E}\right\}$, the set of feasible events $fsble_V^\Theta$ and the set of static cut-off events $Ecut_V^\Theta$. Note that for simplicity $\subset$ is used as adequate order.

E. Condition for diagnosability

A trace σ in V forms a *cycle* if there exists a trace σ' within σ such that $\left(M^1, M^2, \nu^1, \nu^2\right)[\sigma'\rangle\left(M^1, M^2, \nu^1, \nu^2\right)$ and $\sigma' \neq \emptyset$. Let $e \in Ecut_V^\Theta$ be a cut-off event in $Pref_V^\Theta$ then $\kappa^e \stackrel{\text{df}}{=} \{\kappa : [e] \subseteq \kappa\}$. (Note that in the sequel proofs are given in the appendix.)

Lemma 1: The following holds.

1) for each cycle in the verifier V there is a configuration $\kappa \in \kappa^e$ such that $e \in Ecut_V^\Theta$
2) for each $\kappa \in \kappa^e$, where $e \in Ecut_V^\Theta$, there exists a cycle in the verifier V

It is assumed that there are no cycles of unobservable events; however, if this assumption is dropped it can be easily checked while building the canonical prefix (by checking whether there is an occurrence of an observable event between the cut-off events and their corresponding events; note that the corresponding event of a cut-off event is always in the history of its cut-off event since $\subset$ is used as adequate order).

Lemma 2: Let $q = \left(M^1, M^2, \nu^1, \nu^2\right)$ and $q' = \left(M'^1, M'^2, \nu'^1, \nu'^2\right)$ be two states in a cycle in V then $\nu^1 = \nu'^1$ and $\nu^2 = \nu'^2$.

Recall that the verifier compares every pair of traces in the system which have the same observables. Thus, if a cut-off event e occurs this means that there exists a cycle (between e and its corresponding event). Furthermore, if there exist different local fault labels of F_i, i.e. $\nu_i^1 \neq \nu_i^2$ with $Fault([e]) = \left(\nu^1, \nu^2\right)$, the system is not diagnosable w.r.t. F_i. This is illustrated in Figure 3(b), where a part of the canonical prefix of the verifier in Figure 3(a) is shown. The cut-off event e_6 and its corresponding event e_5 form a cycle since their basic configurations reach the same marking $\left\{p_8^1, p_8^2\right\}$. It is evident that the system is not diagnosable w.r.t. F_1 since their corresponding fault labels are different. Thus, there exist two equivalent observable traces in the system one with an occurrence of F_1 (f_1^1, a, b, c) and the other without any occurrence of F_1 (u^2, a, b, c). They can never be distinguished since they are within a cycle.

However, it is not enough to check the basic configuration of a cut-off event e since its concurrent events can influence the decision. This is also illustrated in Figure 3(b) with the fault type F_2. The fault label of $[e_6]$ indicates that there is no ambiguity, yet the occurrence of the concurrent event e_4, which corresponds to F_2, changes one of the fault label of F_2 and the system becomes not diagnosable w.r.t. F_2. One has f_1^1, a, b, c and u^2, a, b, c with the occurrence of f_2^2 (note that this happens only in the case where the corresponding fault vectors of a fault type F_i are zero, otherwise one can straight conclude about diagnosability without considering concurrent events of cut-offs).

Proposition 1: Let $Pref_V^\Theta$ be the canonical prefix of the verifier V with its set of cut-off events $Ecut_V^\Theta$. Then V is called *F_i-diagnosable* w.r.t. O and F_i if $\forall e \in Ecut_V^\Theta\ \forall \kappa \in \kappa^e, \nu_i^1 = \nu_i^2$, where $Fault(\kappa) = \left(\nu^1, \nu^2\right)$.

Remark 1: The system $\mathcal{N}$ is *diagnosable* w.r.t. O and F if it is F_i-diagnosable for all $F_i \in F$.

Remark 2: To reduce complexity of the verifier one can consider one fault type at a time and perform the diagnosability check n times w.r.t. to the fault type set $F_1, ..., F_n$ as in [9] (by setting other faults as non-fault unobservables) . The complexity is then linear in the number of faults (the state space reduces by 2^{n-1}).

Algorithm 1 General algorithm

input :$V = \left(\mathcal{N}^1 \times \mathcal{N}^2, \left(\nu^{1^0}, \nu^{1^2}\right)\right)$
output :Δ_i

$Pref \leftarrow C^0 = h^{-1}\left(M^0\right)$
$pe \leftarrow PE\left(Pref\right)$ (possible extensions)
$Ecut \leftarrow \emptyset$ (cut-off events)
$\Delta_i \leftarrow \emptyset$ (minimal non-diagnosable traces of F_i)

while $pe \neq \emptyset$ **do**
  choose $e \in min_{\subset} pe$
  $pe \leftarrow pe \backslash \{e\}$
  if e is a cut-off event **then**
    $Ecut \leftarrow Ecut \cup \{e\}$
  else
    $Pref \leftarrow Pref \oplus \{e\}$
    $pe \leftarrow pe \cup updatePE\left(Pref, e\right)$

$Pref \leftarrow Pref \oplus \{Ecut\}$

forall $e \in Ecut$ ***do***
  forall $i, 0 \leq i \leq n$ ***do***
    forall $\kappa \in \kappa^e$ ***do***
      if κ is non-diagnosable w.r.t. F_i **then**
        $\Delta_i \leftarrow \Delta_i \cup \{\kappa\}$

IV. Verification of diagnosability

A. General algorithm

Algorithm 1 generates the canonical prefix $Pref_V^{\Theta}$ and checks diagnosability. It returns (if the system is not diagnosable) a set of configurations Δ_i for each fault type F_i which explain why the system is not diagnosable. The canonical prefix $Pref_V^{\Theta}$ is build by using the algorithm presented in [12]. The construction of $Pref_V^{\Theta}$ starts with the conditions which correspond to the initial marking of the verifier V and no events. Then, the set of possible extension is computed by the function PE. New non-cut-off events together with their post-sets are added each at a time and the set of possible extension is updated w.r.t. the new added event (by the function $updatePE$). Whereas, cut-off events are stored and added after the prefix generation terminates (when the set of possible extension is empty). In order to test diagnosability the cut-off events and their concurrent events are examined for ambiguous fault labels. The result is then reported in form of a set of configurations showing non-diagnosable traces if they exists.

B. Improvements

As stated in the previous section one can consider one fault type F_i at a time and perform the diagnosability test n times. The above procedure can also be applied in the same way but with a reduced fault vector i.e. only the i^{th} bit corresponding to F_i is considered, and by setting the remaining fault types $F \setminus F_i$ as non-fault unobservables. This approach is favoured by the diagnosability community.

1) Contracted verifier: It would be advantageous to exploit the symmetry of the verifier. Recall that the verifier V compares every pair of equivalent observable traces corresponding to the diagnosers, D^1 and D^2. Given a fault f in V f^1 corresponds to D^1 and f^2 corresponds to D^2. There are four cases:

1) a fault f^1 occurs but not its counterpart f^2,
2) a fault f^2 occurs but not its counterpart f^1,
3) the occurrence of both f^1 and f^2, and
4) the absence of both f^1 and f^2.

The first two cases show that the system is not diagnosable if occurring in a infinite trace, and the last two cases indicating that the considered trace is diagnosable.

Due to the symmetry it is sufficient to consider either Case 1 or 2 (e.g. let consider Case 1). Moreover, the Case 4 can be made redundant by removing from the verifier V the fault transitions F^2 corresponding to the diagnoser D^2 together with their arcs resulting in a *contracted* verifier V_c. By doing this the traces containing a fault of F^2 are not reachable leaving only Case 1 and 3. Thus, the diagnosability check is reduced as follows. The system is not diagnosable if there exists an infinite trace containing a fault corresponding to D^1. Note that it is enough to check for faults corresponding to D^1 since faults corresponding to D^2 will never occur in a trace because they are not reachable due to the removal.

Algorithm 1 can be applied to check diagnosability with the contracted verifier and the reduced diagnosability check. Consider the verifier in Figure 3(a). Its contracted version is shown in Figure 4(a), where fault transitions $F^2 = \left\{f_1^2, f_2^2\right\}$ and their arcs are removed. The transition marked black correspond to transitions which are not reachable due to the removal. This is evident on the canonical prefix depicted in Figure 4(b). Note that the fault vector corresponding to D^1 is only important for the diagnosability check since the other fault vectors are always zero. There are two cut-off events, e_8 and e_{10}; it is immediately evident from the fault vector of e_{10} that the system is not diagnosable w.r.t. F_1, and it is later evident that there exist a configuration in κ^{e_8} (containing e_8 and its concurrent event e_9, which correspond to f_2^1) that shows that the system is also not diagnosable w.r.t. F_2.

2) Reduced verifier w.r.t a fault: One can go a step further and consider a verifier built out of the product of a diagnoser

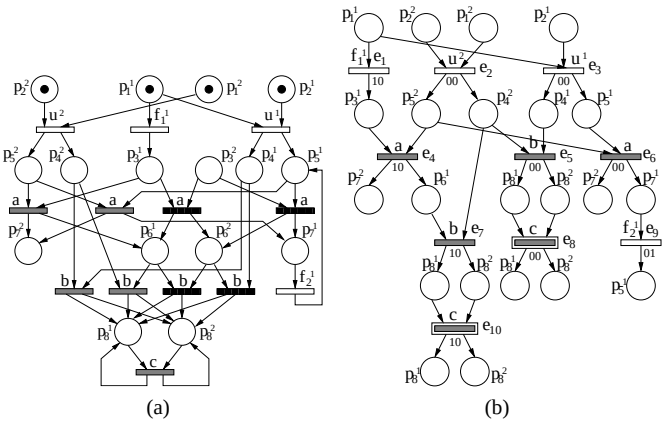
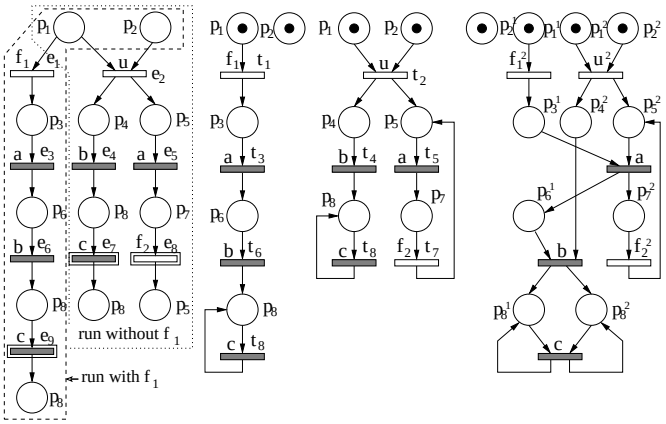


Figure 4: PN of the contracted verifier V_c (a) and its canonical prefix (b).



(a) $Pref^{\Theta}_{\mathcal{N}}$ (b) D_{f_1} (c) $D_{\bar{f}_1}$ (d) $V_{f_1} = D_{f_1} \times D_{\bar{f}_1}$

Figure 5: Reducing the verifier applied to fault f_1.

containing only fault occurrences and a diagnoser containing only non-fault occurrences. This is especially interesting for the case where one fault type at a time is considered due to the size reduction. Given a diagnoser D in which one fault type F_i is considered (i.e. $F \setminus F_i$ is set to non-fault unobservables) one has the *reduced* verifier w.r.t. the fault f_i

$$V_{f_i} = D_{f_i} \times D_{\bar{f}_i}, \tag{4}$$

where f_i is any fault in F_i, $1 \leq i \leq n$, and the reduced diagnoser D_{f_i} ($D_{\bar{f}_i}$), which corresponds to the part of D containing (non-)f_i-fault occurrences. Extracting fault and non-fault occurrences at the level of the diagnoser is not straightforward due to the cyclicity of the net. However, it is possible to extract this information from the canonical prefix of the underlying system of the diagnoser $Pref^{\Theta}_{\mathcal{N}}$ (with $\subset$ as adequate order) by examining maximal configurations w.r.t. set inclusion, called *runs*.

Definition 5: Let Ω be the set of runs in $Pref^{\Theta}_{\mathcal{N}}$. Then,

- $\Omega_{f_i} = \{\omega \in \Omega / \exists e \in \omega : \ell \circ h(e) \in F_i\}$
- $\Omega_{\bar{f}_i} = \{\omega \in \Omega / \forall e \in \omega : \ell \circ h(e) \notin F_i\}$,

where e is an event in $Pref^{\Theta}_{\mathcal{N}}$.

In other words Ω_{f_i} ($\Omega_{\bar{f}_i}$) is the set of runs where faults from type F_i (not) occur. By definition, Ω_{f_i} ($\Omega_{\bar{f}_i}$) corresponds to all the possible executions of the reduced underlying system model representing the reduced diagnoser D_{f_i} ($D_{\bar{f}_i}$). Hence, the projection of Ω_{f_i} and $\Omega_{\bar{f}_i}$ onto the diagnoser D (via its underlying system model $\mathcal{N}$) correspond to D_{f_i} and $D_{\bar{f}_i}$, respectively. For each fault the reduced verifier is constructed from $Pref^{\Theta}_{\mathcal{N}}$, which is unchanged, only the extracted information differs. Thus, it has to be build only once.

Let consider the prefix of the running example $Pref^{\Theta}_{\mathcal{N}}$ in Figure 5(a) and the case that only the fault type F_1 is considered (thus, events of F_2 are set to unobservable non-fault events). There are two runs in $Pref^{\Theta}_{\mathcal{N}}$, one with the fault f_1 and the other without the considered fault type. From the runs the reduced diagnosers D_{f_1} and $D_{\bar{f}_1}$ (depicted in Figure 5(b) and (c), respectively) are obtained by projecting them on D. The reduced verifier $V_{f_1} = D_{f_1} \times D_{\bar{f}_1}$ is shown in Figure 5(d) and its canonical prefix corresponds to the one in Figure 3(b) with the fault vector corresponding to F_1. Incidentally, the reduced verifier w.r.t. F_2, V_{f_2}, is similar to V_{f_1}, they only differ in the fault vectors; D_{f_2} corresponds to the run in Figure 5(c) and $D_{\bar{f}_1}$ correspond to the run in Figure 5(b).

Algorithm 2 Depth-first approach

input :$V = \left(\mathcal{N}^1 \times \mathcal{N}, \left(\nu^{1^0}, \nu^{1^0}\right)\right)$
output : $isDiagnosable$

$Pref \leftarrow C^0 = h^{-1}\left(M^0\right)$
$pe \leftarrow PE\left(Pref\right)$ (possible extensions)
$Ecut \leftarrow \emptyset$ (cut-off events)
$isDiagnosable \leftarrow true$

while $pe \neq \emptyset$ **do**
  choose $e \in pe$: e is the last added event in pe
  $pe \leftarrow pe \backslash \{e\}$
  if e is a cut-off event **then**
    if e is not diagnosable **then**
      return $isDiagnosable \leftarrow false$
    $Ecut \leftarrow Ecut \cup \{e\}$
  else if $\ell \circ h(e) \in F$ and e is not diagnosable **then**
    return $isDiagnosable \leftarrow false$
  else
    $Pref \leftarrow Pref \oplus \{e\}$
    $pe \leftarrow pe \cup updatePE\left(Pref, e\right)$
return $isDiagnosable$

3) Depth first approach : A depth-first approach[4] (see Algorithm 2) as opposed to the breadth-first approach presented above can be employed to test diagnosability. It is advantageous if simply an answer about the system's diagnosability is needed. The depth-first search tries to extend a branch of a prefix before considering other branches. This means that the possible extensions enabled by the last added events are explored (added to the prefix) before any other possible extensions in the prefix. To do that the diagnosability check has to be extended, not only cut-off events are candidates but also their concurrent events are. Recall that the diagnosability check is done w.r.t some cut-off event e by examining fault labels of κ^e. Since e might be added before its concurrent events (the set κ^e is not complete) they must be included in the diagnosability test. However, only events corresponding to faults are of interest since only they can influence the decision about diagnosability. Thus, given a non-cut-off event e' there exists a configuration corresponding to a non-diagnosable trace iff $\ell \circ h(e') \in F_i$ and there exists a cut-off event e such that $e' \parallel e$ and $\nu_i^1 \neq \nu_i'^2$ or $\nu_i'^1 \neq \nu_i^2$, where $Fault([e]) = \left(\nu^1, \nu^2\right)$ and $Fault([e']) = \left(\nu'^1, \nu'^2\right)$. Observe, that the above introduced improvements can be applied to the depth-first approach in a similar way.

C. Complexity

A canonical prefix $Pref^{\Theta}_{V}$ can be exponentially smaller than the reachability graph of V, especially if V exhibits a high

[4][4] shows that the depth-first search is incorrect when applying it to PN unfolding prefixes, however, here the prefix construction uses an adequate order ($\subset$), and thus, is correct.

degree of concurrency combined with a moderate number of branching behaviour. However, in worst case $Pref_V^{\Theta}$ can be exponential in the size of V. In spite of that, the proposed improvements offer a size reduction of $Pref_V^{\Theta}$. In particular, when taking into account the symmetry of the verifier or considering one fault type at a time the size reduction can be significant when building $Pref_V^{\Theta}$ from the contracted and/or reduced verifier. Moreover, the depth-first approach together with the improvements may offer a more efficient way that the breath-first one.

V. Conclusion and future work

This paper proposes an approach to verify diagnosability in the framework of PN unfoldings based on the twin plant method. The approach consists in constructing a verifier, which compares pairs of paths from the initial model sharing the same observable behaviour. In the canonical prefix of the verifier the diagnosability test is reduced to the comparison of binary vector pairs of configurations associated with cut-off events. Each configuration is linked with a pair of binary vectors containing information about fault occurrences in two executions sharing the same observables. This is further reduced when considering the symmetry of the verifier, then only binary vectors of one configuration instead of the pair is examined while considering a reduced reachable space. Moreover, other proposed improvements can be applied to reduce the complexity.

The advantage of using PN and its unfolding prefixes lies in the compact knowledge representation. This compactness manifests, on the one hand, in the model itself and its extensions (the diagnoser and the verifier) and, on the other hand, in the formalism used to test diagnosability (the canonical prefix of the verifier).

In the future we plan to improve the efficiency of the proposed approach as follows:

- using other adequate orders to build canonical prefixes in order to obtain smaller prefixes with fewer test cases.
- constructing complete prefixes of reduced verifiers directly from the extracted runs of the system's canonical prefix; this would reduced the construction complexity of the canonical prefix of the reduced verifier.
- applying a directed search in the depth-first algorithm as it has been done in [1] for the reachability analysis. The unfolding generation could be guided in such a way that only "interesting" branches are explored first rather than "blindly" choosing branches.

In addition, we also plan to extend this approach to verify diagnosability in distributed systems as an application of modular complete prefixes [13]. Moreover, we currently investigate the application of LTL model checking to diagnosability verification using existing approaches to LTL model checking with net unfoldings [2], [3]. This idea has arisen after having defined the diagnosability problem in this paper.

Acknowledgements

Many thanks to Stefan Haar and Victor Khomenko for interesting discussions.

Appendices

Lemma 1: The following holds.

1) for each cycle in the verifier V there is a configuration $\kappa \in \kappa^e$ such that $e \in Ecut_V^{\Theta}$
2) for each $\kappa \in \kappa^e$, where $e \in Ecut_V^{\Theta}$, there exists a cycle in the verifier V

Proof:

1) Let us consider a cycle in the verifier V $cyc = \left(M_1^1, M_1^2, \nu_1^1, \nu_1^2\right)[a_1, ..., a_{k-1}\rangle\left(M_k^1, M_k^2, \nu_k^1, \nu_k^2\right)$ with $\left(M_1^1, M_1^2, \nu_1^1, \nu_1^2\right) = \left(M_k^1, M_k^2, \nu_k^1, \nu_k^2\right)$. By definition of cut-off events it follows that some a_i $(1 \leq i \leq k-1)$ corresponds to a cut-off event e_i in $Pref_V^{\Theta}$. Without loss of generality let $a_i = a_{k-1}$ and $e_i = e_{k-1}$. We consider two cases.
   a) The trace $a_1, ..., a_{k-1}$ correspond to the configuration $[e_{k-1}]$, $a_j \in \ell \circ h\left([e_{k-1}]\right)$ for $j = 1...k-1$. Thus, $[e_{k-1}]$ is a configuration of $\kappa^{e_{k-1}}$ which corresponds to the cycle cyc.
   b) There is a subset $B = \{b_1, ..., b_m\}$ $(m \geq 1)$ of $A = \{a_1, ..., a_{k-1}\}$ such that $b_m = a_{k-1}$ and $b_j \in \ell \circ h\left([e_{k-1}]\right)$. Let $D = A \setminus B = \{d_1, ..., d_r\}$ $(r \geq 1)$ be the remaining transition labels of A. Then, each element d_j does not correspond to a predecessor of e_{k-1} (otherwise it would be in $[e_{k-1}]$). It does not correspond to successor of e_{k-1} (since e_{k-1} corresponds to a_{k-1} the last transition label in the trace). Furthermore, it does not correspond to an event in conflict with elements of e_{k-1} since the trace $a_1, ..., a_{k-1}$ includes d_j and a_{k-1} such that $a_{k-1} = \ell \circ h\left(e_{k-1}\right)$. This means that each e_j $(d_j = \ell \circ h\left(e_j\right))$ is concurrent to the cut-off event e_{k-1}. Therefore, the configuration κ corresponding to $a_1, ..., a_{k-1}$ contains $[e_{k-1}]$, $\kappa \in \kappa^{e_{k-1}}$.
2) Suppose $\kappa \in \kappa^e$, where $e \in Ecut_V^{\Theta}$. We consider two cases.
   a) If $\kappa = [e]$ then by definition of cut-off events κ corresponds to a cycle.
   b) If $\kappa = \kappa^e$ and $\kappa \neq [e]$. Let us consider the cycle cyc_{min}, which corresponds to $[e]$ and to the trace $b_1, ..., b_m$ (where $b_m = \ell \circ h\left(e\right)$). Since $\kappa^e = \{\kappa : [e] \subseteq \kappa\}$ then κ includes in addition to events of $[e]$ (corresponding to $b_1, ..., b_m$) a concurrent set of events corresponding to $d_1...d_r$. Let $a_1...a_{k-1}$ be a trace involving elements of both $b_1, ..., b_m$ and $d_1...d_r$. Let $q = \left(M^1, M^2, \nu^1, \nu^2\right)$ be the state of V reached after executing a trace involving all the events $a_1...a_{k-1}$. We can then show that q belongs to a cycle. Indeed, this cycle is obtained by executing from q the trace $b_1, ..., b_m$, i.e. $\left(M^1, M^2, \nu^1, \nu^2\right)[b_1\rangle...[b_m\rangle\left(M^1, M^2, \nu^1, \nu^2\right)$ since the concurrent events are not executed the same marking is reached.

■

Lemma 2: Let $q = \left(M^1, M^2, \nu^1, \nu^2\right)$ and $q' =$

$\left(M'^1, M'^2, \nu'^1, \nu'^2\right)$ be two states in a cycle in V then $\nu^1 = \nu'^1$ and $\nu^2 = \nu'^2$.

Proof: Let us consider for the sake of contradiction that $\exists j : \nu_j^1 \neq \nu_j'^1$, e.g. $\nu_j^1 = 1$ and $\nu_j'^1 = 0$. Since q and q' are in a cycle of V then the states $s_1 = \left(M^1, \nu^1\right)$ (respectively $s_2 = \left(M^2, \nu^2\right)$) and $s_1' = \left(M'^1, \nu'^1\right)$(respectively $s_2' = \left(M'^2, \nu'^2\right)$) are in a cycle of the diagnoser D. For some trace $\sigma = b_1, ..., b_m$ we have $\left(M^1, \nu^1\right)[\sigma\rangle\left(M'^1, \nu'^1\right)$. Since $\nu_j^1 = 1$ and by definition of the fault propagation we must have $\nu_j'^1 = 1$ (faults are not reversible!). This contradicts the above assumption. It can be proven that the other case (if we suppose that $\nu_j^1 = 0$ and $\nu_j'^1 = 1$) leads also to a contradiction by taking a path from $\left(M'^1, \nu'^1\right)$ to $\left(M^1, \nu^1\right)$ and using the same arguments. It can also be proven in an analogous manner that $\nu_j^2 = \nu_j'^2$. ■

Proposition 1: Let $Pref_V^{\Theta}$ be the canonical prefix of the verifier V with its set of cut-off events $Ecut_V^{\Theta}$. Then V is called *F_i-diagnosable* w.r.t. O and F_i if $\forall e \in Ecut_V^{\Theta}\ \forall \kappa \in \kappa^e, \nu_i^1 = \nu_i^2$, where $Fault(\kappa) = \left(\nu^1, \nu^2\right)$.

Proof: The following proof is inspired from that of Theorem 2 in [19].

($\Rightarrow$) Suppose that $L(\mathcal{N})$ is diagnosable. For the sake of contradiction suppose that $\exists e \in Ecut_V^{\Theta}, \exists \kappa \in \kappa^e, \nu_i^1 \neq \nu_i^2$, where $Fault(\kappa) = \left(\nu^1, \nu^2\right)$. From Lemma 1 it follows that there exists a cycle in V which corresponds to κ. Hence, there exist two corresponding cycles in D: there are two traces s_0 and s_0' such that $\left(M^0, \nu^0\right)[s_0\rangle\left(M_1^1, \nu^1\right)$ and $\left(M^0, \nu^0\right)[s_0'\rangle\left(M_1^2, \nu^2\right)$ with $Obs(s_0) = Obs(s_0')$ and $\nu_i^1 \neq \nu_i^2$. For example, $\nu_i^1 = 1$ and $\nu_i^2 = 0$, i.e. $f_i \in s_0$ and $f_i \notin s_0'$. The cycle of the first machine is $\left(M_1^1, \nu^1\right)[a_1\rangle\left(M_2^1, \nu^1\right) ... [a_{n-1}\rangle\left(M_1^1, \nu^1\right)$, and the cycle of the second machine is $\left(M_1^2, \nu^2\right)[b_1\rangle\left(M_2^2, \nu^2\right) ... [b_{m-1}\rangle\left(M_1^2, \nu^2\right)$.

Let us take for $k > 1$, $Z(k) = s_0 (a_1, ..., a_{n-1})^k$ and $Z'(k) = s_0' (b_1, ..., b_{n-1})^k$. Then, we have $Z(k), Z'(k) \in L(\mathcal{N})$, $Obs(Z(k)) = Obs(Z'(k))$, and $f_i \in Z(k)$ and $f_i \notin Z'(k)$. Moreover, we have $|Obs(Z(k))| \geq 1$ and $|Obs(Z'(k))| \geq 1$ due to the assumption that there are no unobservable cycles.

Since $f_i \in Z(k)$ let us consider $Z(k) = \sigma\sigma'$ such that σ finishes with f_i and σ' is the continuation of σ w.r.t. $Z(k)$. Then, by choosing a large k we will find that $|\sigma'| > n$ for each $n > 1$ and $Obs(Z(k)) = Obs(Z'(k)) = Obs(\sigma\sigma')$, i.e. $Z'(k) \in Obs^{-1}(\sigma\sigma')$ but $f_1 \notin Z'(k)$, which represent a counter example to the diagnosability of the system. This is a contradiction.

($\Leftarrow$) Suppose that $\forall e \in Ecut_V^{\Theta}, \forall \kappa \in \kappa^e, \nu_i^1 = \nu_i^2$ with $Fault(\kappa) = \left(\nu^1, \nu^2\right)$. For the sake of contradiction suppose $L(\mathcal{N})$ is non-diagnosable: $(\forall n \geq 1)\, \exists \sigma\sigma'$ (σ is a trace finishing with f_i and σ' is one of its continuation in $L(\mathcal{N})$) such that $|\sigma'| \geq n$ and $\exists \omega \in Obs^{-1}(\sigma\sigma')$ and $f_i \notin \omega$. Let Max be the maximum number of states in the state graph of the V (denoted by G^V), and let $n \geq Max$. Let $\omega = s\omega'$ such that $s \in Obs^{-1}(\sigma)$; since $f_i \notin \omega$ it is obvious that $f_i \notin s$. In the diagnoser we have: $\left(M^0, \nu^0\right)[s\rangle(M^\sigma, \nu^\sigma)$ and $\left(M^0, \nu^0\right)[s\rangle(M^s, \nu^s)$ with $\nu_i^\sigma = 1$ and $\nu_i^s = 0$. The state $q = (M^\sigma, M^s, \nu^\sigma, \nu^s)$ is accessible in G^V.

We have also in the diagnoser: $\left(M^0, \nu^0\right)[\sigma\sigma'\rangle\left(M^{\sigma\sigma'}, \nu^{\sigma\sigma'}\right)$ and $\left(M^0, \nu^0\right)[\omega\rangle(M^\omega, \nu^\omega)$ with $\nu_i^{\sigma\sigma'} = 1$ and $\nu_i^\omega = 0$. The state $q' = \left(M^{\sigma\sigma'}, M^\omega, \nu^{\sigma\sigma'}, \nu^\omega\right)$ is accessible in G^V.

Since we take $|\sigma'| \geq n$ there is some $r \geq n$ and some trace $a_1 ... a_h$ such that $\left(M_0^1, M_0^2, \nu_0^1, \nu_0^2\right)[a_1\rangle\left(M_1^1, M_1^2, \nu_1^1, \nu_1^2\right) ... [a_r\rangle\left(M_r^1, M_r^2, \nu_r^1, \nu_r^2\right)$ with $q = \left(M_0^1, M_0^2, \nu_0^1, \nu_0^2\right)$ and $q' = \left(M_r^1, M_r^2, \nu_r^1, \nu_r^2\right)$. Since $r \geq n \geq Max$ there must be a cycle inside the path between q and q'. This means that for some x and y with $1 \leq x \leq y \leq r$ we have $\left(M_x^1, M_x^2, \nu_x^1, \nu_x^2\right) = \left(M_y^1, M_y^2, \nu_y^1, \nu_y^2\right)$ with $\left(\nu_x^1\right)_i = \left(\nu_y^1\right)_i = 1$ and $\left(\nu_x^2\right)_i = \left(\nu_y^2\right)_i = 0$. According to the Lemma 1 this cycle corresponds to a cut-off event e and a configuration $\kappa \in \kappa^e$ such that $\nu_i^1 \neq \nu_i^2$, where $Fault(\kappa) = \left(\nu^1, \nu^2\right)$. This is in a contradiction with the supposition. ■

REFERENCES

[1] Blai Bonet, Patrik Haslum, Sarah Hickmott, and Sylvie Thiébaux. Directed unfolding of petri nets. In *Workshop on Unfolding and Partial Order Techniques (UFO) in 28th Int. Conf. on Application and Theory of Petri Nets and Other Models of Concurrency*, Siedlce, Poland, 2007.

[2] Javier Esparza and Keijo Heljanko. A New Unfolding Approach to LTL Model Checking. In *ICALP '00: Proceedings of the 27th International Colloquium on Automata, Languages and Programming*, pages 475–486, London, UK, 2000. Springer-Verlag.

[3] Javier Esparza and Keijo Heljanko. Implementing LTL model checking with net unfoldings. In *SPIN '01: Proceedings of the 8th international SPIN workshop on Model checking of software*, pages 37–56, New York, NY, USA, 2001. Springer-Verlag New York, Inc.

[4] Javier Esparza, Pradeep Kanade, and Stefan Schwoon. A negative result on depth-first net unfoldings. *Int. J. Softw. Tools Technol. Transf.*, 10(2):161–166, 2008.

[5] Javier Esparza, Stefan Römer, and Walter Vogler. An Improvement of McMillan's Unfolding Algorithm. *Form. Methods Syst. Des.*, 20(3):285–310, 2002.

[6] E. Fabre, A. Benveniste, S. Haar, and C. Jard. Distributed Monitoring of Concurrent and Asynchronous Systems. *Journal of Discrete Event Systems, special issue*, pages 33–84, May 2005.

[7] S. Haar. Unfold and Cover: Qualitative Diagnosability for Petri Nets. In *Proceedings CDC*, 2007.

[8] Stefan Haar, Albert Benveniste, Eric Fabre, and Claude Jard. Partial Order Diagnosability of Discrete Event Systems using Petri Nets Unfoldings. In *42nd IEEE Conference on Decision and Control (CDC)*, 2003.

[9] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete event systems. In *IEEE Transactions on Automatic Control*, 2001.

[10] V Khomenko. *Model Checking Based on Petri Net Unfolding Prefixes*. PhD thesis, University of Newcastle upon Tyne, 2002.

[11] V. Khomenko, M. Koutny, and A. Yakovlev. Logic Synthesis for Asynchronous Circuits Based on Petri Net Unfoldings and Incremental SAT. In *Int. Conf. on Application of Concurrency to System Design*, pages 16–25. IEEE Computer Society Press, June 2004.

[12] Victor Khomenko, Maciej Koutny, and Walter Vogler. Canonical prefixes of petri net unfoldings. *Acta Informatica, Volume 40, Number 2*, pages 95–118, October 2003.

[13] A. Madalinski and E. Fabre. Modular construction of finite and complete prefixes. In *Int. Conf. on Application of Concurrency to System Design*, 2008. to appear.

[14] K. L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Proc. International Workshop on Computer Aided Verification*, pages 164–177, July 1992.

[15] Yannick Pencolé. Diagnosability analysis of distributed discrete event systems. In *European Conference on Artificial Intelligence*, pages 173–178, Valencia, Spain, 2004. IEEE Computer Society Press.

[16] L. Y. Rosenblum and A. V. Yakovlev. Signal Graphs: from Self-Timed to Timed Ones. In *Proceedings of International Workshop on Timed Petri Nets*, pages 199–207, Torino, Italy, July 1985. IEEE Computer Society Press.

[17] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of Discrete Events Systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995.

[18] Anika Schumann and Yannick Pencolé. Scalable diagnosability checking of event-driven systems. In *20th International Joint Conference on Artificial Intelligence*, pages 575–580, Hyderabad, India, 2007.

[19] Tae-Sic Yoo and S Lafortune. Polynomial-Time Verification of Diagnosability of Partially Observed Discrete-Event Systems. *IEEE Transactions on Automatic Control*, 47(9):1491–1495, 2002.