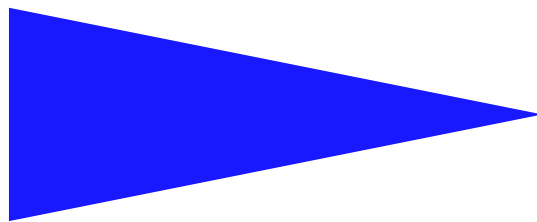# DIAGNOSIS OF ASYNCHRONOUS DISCRETE EVENT SYSTEMS, A NET UNFOLDING APPROACH

ALBERT BENVENISTE, ERIC FABRE, CLAUDE JARD, AND STEFAN HAAR

IRISA

Diagnosis of asynchronous discrete event systems, a net unfolding approach [*] [**]

Albert Benveniste, Eric Fabre, Claude Jard, and Stefan Haar [***]

Abstract: In this paper we formulate asynchronous diagnosis by means of hidden state history reconstruction, from alarm observations. We follow a so-called true concurrency approach, in which no global state and no global time is available. Instead, we use only local states in combination with a partial order model of time, in which local events are ordered if they are either generated on the same site, or related via some causality relation. Our basic mathematical tool is that of *net unfoldings* originating from the Petri net research area. This study was motivated by the problem of event correlation in telecommunications network management.

Key-words: decentralized diagnosis, asynchronous diagnosis, discrete event systems, Petri nets, unfoldings, alarm correlation.

(Résumé : tsvp)

Diagnostic de systèmes à événements discrets asynchrones, une approche par dépliages de réseaux

Résumé :   Dans cet article nous étudions le problème du diagnostic pour des systèmes asynchrones. Ce problème est formulé comme un problème de reconstruction de trajactoire d'état à partir des alarmes observées. Nous adoptons un point de vue dit de la "concurrence vraie", ce qui signifie que nous ne manipulons jamais d'états globaux, et que nous utilisons un temps qui a la structure d'un ordre partiel. Notre outil principal est le concept de *dépliage* de réseau de Petri. Nous étudions un certain nombre de variantes de ce problème. Cette étude est motivée par le cas de la corrélation d'alarmes en gestion de réseaux.

Mots clés :   Diagnostic asynchrone, systèmes discrets, réseaux de Petri, dépliage, corrélation d'alarmes.

Contents

1 Introduction

In this paper we study the diagnosis of truly asynchronous systems. Typical examples are networked systems, such as shown in Fig. 1. In this figure, the sensor system is distributed,
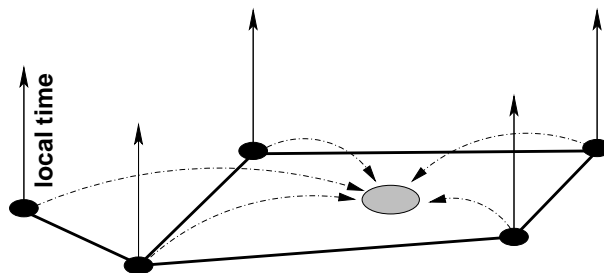

Figure 1: Supervision of a networked system.

it involves several local sensors, attached to some nodes of the network (shown in black). Each sensor has only a partial view of the overall system. The different sensors possess their own local time, but they are not synchronized. Alarms are reported to the global supervisor (depicted in grey) asynchronously, and this supervisor performs diagnosis. This is the typical architecture in telecommunications network management systems today, our motivating application[1]. Events may be correctly ordered by each individual sensor, but communicating alarm events via the network causes a loss of synchronization, and results in a *nondeterministic* and possibly *unbounded* interleaving at the supervisor. Hence, the right picture, for what the supervisor collects, is not a sequence of alarms, but rather a *partially ordered* set of alarms.

Fault diagnosis in discrete event systems has attracted a significant attention, see the work of Lafortune and co-authors [33][9] for an overview of the literature and introduction to the subject. Decentralized diagnosis is analyzed in [9], including both algorithms and their diagnosability properties; the solution is formulated in terms of a precomputed decentralized *diagnoser*, consisting of a set of communicating machines that have their states labeled by sets of faults and react to alarm observations and communications; the language oriented framework of Wonham and Ramadge (see [8]) is used, and the systems architecture is that of communicating automata, with a synchronous communication based on a global time, as revealed by the assumption "A6" in [9]. The work [9] has been extended by the same authors in [10] toward considering the effect of (bounded) communication delays in decentralized diagnosis. Difficulties resulting from communications in diagnosis are also investigated by Sengupta in [34]. Finally, the recent work of Tripakis [35] discusses issues of undecidability for a certain type of decentralized observability, this issue has again some relation with asynchrony. Baroni et al. [3] propose a different approach, more in the form of a simulation

---

[1]See [26] and the url http://magda.elibel.tm.fr/ for a presentation of the MAGDA project on fault management in telecommunications networks.

guided by the observed alarms, for a model of communicating automata. The solution proposed offers a first attempt to handle the problem of state explosion which results from the interleaving of events involving different components.

Diagnosis in the framework of Petri net models has also been investigated by some authors. Hadjicostis and Verghese [20] consider faults in Petri nets in the form of losses or duplications of tokens; this is different from using Petri nets as an asynchronous machine model, for diagnosis. Valette and co-authors [32] use Petri nets to model the normal behavior of systems, and consider as faults the occurrence of events that do not match firing conditions properly. The work closest to ours is that of Giua and co-authors [21][22], it considers the estimation of the current marking from observations.

Event correlation in network management is the subject of a considerable literature, and a number of commercial products are available. We refer the reader to Gardner [19] for a survey. There are two main frameworks for most methods developed in this area. The first one relates to rule-based or case-based reasoning, an approach very different from the one we study here. The second one uses a causal model, in which the relation between faulty states and alarm events is modelled. The articles by Bouloutas et al. [6][7][25] belong to this family, as well as Rozé and Cordier [30] which relies on the diagnoser approach of [33]. The case of event correlation in network management also motivated the series of papers by Fabre et al. [5][1][2], on which the present paper relies.

As said before, our present approach was motivated by the problem of fault management in telecommunications networks, so it is worth discussing how this context motivated some of our choices. As seen from our bibliographical discussion, two classes of approaches were available, to handle the diagnosis of asynchronous systems.

A possible approach would consist in constructing a diagnoser in the form of a Petri net, having certain places labeled by faults, and transitions labeled by alarms. Received alarms trigger the net, and visiting a faulty place would indicate that some fault occurred in the original net for monitoring. Another approach would consist in estimating the current marking of the Petri net for monitoring, as in [21][22].

For our application, we needed to support distributed faults and event propagation and distributed sensor setups, from which wrong interleaving can result. Hence we feel it important, that robustness against a wrong interleaving should be addressed. However, the available approaches typically assume that alarms are received in sequence and that this sequence is an actual firing sequence of the net, an assumption not acceptable in our context.

Also, for our application in fault management in telecommunications networks (where faults are typically transient), providing explanations in the form of *scenarios,* not just snapshots, was essential. Finally, returning all scenarios compatible with the observations, was the requirement from operators in network management. They did not ask for a more elaborated information such as fault identification, or isolation.

In this paper, we propose an approach to handle unbounded asynchrony in discrete event systems diagnosis by using net *unfoldings,* originally proposed by M. Nielsen, G. Plotkin, and G. Winskel [28]. Unfoldings were used by Mc Millan [27] for model checking in verification. They were subsequently developed by Engelfriet [12], Rozenberg and Engelfriet [31], Esparza,

and Römer [13][14][15]. Net unfoldings are branching structures suitable to represent the set of executions of a Petri net using an asynchronous semantics with local states and partially ordered time. In this structure, common prefixes of executions are shared, and executions differing only in the interleaving of their transition firings are represented only once. Our motivation, for using Petri nets and their unfoldings, is to have an elegant model of asynchronous finite state machines, therefore we restrict ourselves to safe Petri nets throughout this paper. Net unfoldings are not well-known in the control community, they have been however used for supervisory control in [23][24].

The paper is organized as follows. Section 2 is devoted to the problem setting. Subsection 2.1 collects the needed background material on Petri nets and their unfoldings. Subsection 2.2 introduces our first example. And our problem setting for asynchronous diagnosis is formalized in subsection 2.3, which constitutes per se our first contribution.

In asynchronous diagnosis, some recorded alarm sequences differ only via the interleaving of concurrent alarms, hence it is desirable not to distinguish such alarm sequences. Similarly, it is desirable not to distinguish diagnoses which only differ in the interleaving of concurrent faults. *Diagnosis nets* are introduced to this end in section 3, they express the solution of asynchronous diagnosis by using suitable unfoldings, and constitute the main contribution of this paper. Corresponding algorithms are given in section 4. These algorithms have the form of pattern matching rules and apply asynchronously.

2   Asynchronous diagnosis: problem setting

In this section we first introduce the background we need on Petri nets and their unfoldings. Then we introduce informally asynchronous diagnosis on an example. And finally we formally define asynchronous diagnosis.

2.1   Background notions on Petri nets and their unfoldings

Basic references are [29][8][11]. Homomorphisms, conflict, concurrency, and unfoldings, are the essential concepts on which a true concurrency and fully asynchronous view of Petri nets is based. In order to introduce these notions, it will be convenient to consider general "nets" in the sequel.

Nets and homomorphisms.   A *net* is a triple $\mathcal{P} = (P, T, \rightarrow)$, where P and T are disjoint sets of *places* and *transitions*, and $\rightarrow \subset (P \times T) \cup (T \times P)$ is the *flow relation*. The reflexive transitive closure of the flow relation $\rightarrow$ is denoted by $\preceq$, and its irreflexive transitive closure is denoted by $\prec$. Places and transitions are called *nodes*, generically denoted by x. For $x \in P \cup T$, we denote by $^\bullet x = \{y : y \rightarrow x\}$ the *preset* of node x, and by $x^\bullet = \{y : x \rightarrow y\}$ its *postset*. For $X \subset P \cup T$, we write $^\bullet X = \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet = \bigcup_{x \in X} x^\bullet$. An *homomorphism* from a net $\mathcal{P}$ to a net $\mathcal{P}'$ is a map $\varphi : P \cup T \longmapsto P' \cup T'$ such that: 1/ $\varphi(P) \subseteq P'$, $\varphi(T) \subseteq T'$, and 2/ for every node x of $\mathcal{P}$, the restriction of φ to $^\bullet x$ is a bijection between $^\bullet x$ and $^\bullet \varphi(x)$, and the restriction of φ to $x^\bullet$ is a bijection between $x^\bullet$ and $\varphi(x)^\bullet$.

Occurrence nets. Two nodes x, x' of a net $\mathcal{P}$ are *in conflict,* written $x \# x'$, if there exist distinct transitions $t, t' \in T$, such that ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$ and $t \preceq x$, $t' \preceq x'$. A node x is in *self-conflict* if $x \# x$. An *occurrence net* is a net $\mathcal{O} = (B, E, \rightarrow)$, satisfying the following additional properties:

$$\forall x \in B \cup E : \neg[x \# x] \qquad \text{no node is in self-conflict}$$
$$\forall x \in B \cup E : \neg[x \prec x] \qquad \preceq \text{ is a partial order}$$
$$\forall x \in B \cup E : |\{y : y \prec x\}| < \infty \qquad \preceq \text{ is well founded}$$
$$\forall b \in B : |{}^\bullet b| \leq 1 \qquad \begin{array}{l} \text{each place has at most} \\ \text{one input transition} \end{array}$$

We will assume that the set of minimal nodes of $\mathcal{O}$ is contained in B, and we denote by $\min(B)$ or $\min(\mathcal{O})$ this minimal set. Specific terms are used to distinguish occurrence nets from general nets. B is the set of *conditions,* E is the set of *events,* $\prec$ is the *causality* relation.

Nodes x and x' are *concurrent,* written $x \perp\!\!\!\perp x'$, if neither $x \preceq x'$, nor $x \preceq x'$, nor $x \# x'$ hold. A *co-set* is a set X of concurrent conditions. A maximal (for set inclusion) co-set is called a *cut.* A *configuration* κ is a sub-net of $\mathcal{O}$, which is *conflict-free* (no two nodes are in conflict), and *causally closed* (if $x' \preceq x$ and $x \in \kappa$, then $x' \in \kappa$).

Occurrence nets are useful to represent executions of Petri nets. They are a subclass of nets, in which essential properties are visible via the topological structure of the bipartite graph.

Petri nets. For $\mathcal{P}$ a net, a *marking* of $\mathcal{P}$ is a multiset M of places, i.e., a map $M : P \longmapsto \{0, 1, 2, \ldots\}$. A *Petri net* is a pair $\mathcal{P} = (\mathcal{P}, M_0)$, where $\mathcal{P}$ is a net having finite sets of places and transitions, and M_0 is an *initial* marking. A transition $t \in T$ is *enabled* at marking M if $M(p) > 0$ for every $p \in {}^\bullet t$. Such a transition can *fire,* leading to a new marking $M' = M - {}^\bullet t + t^\bullet$, we denote this by $M[t\rangle M'$. The set of *reachable* markings of $\mathcal{P}$ is the smallest (w.r.t. set inclusion) set $M_0[\rangle$ containing M_0 and such that $M \in M_0[\rangle$ and $M[t\rangle M'$ together imply $M' \in M_0[\rangle$. Petri net $\mathcal{P}$ is *safe* if $M(P) \subseteq \{0, 1\}$ for every reachable marking M. Throughout this paper, we consider only safe Petri nets, hence marking M can be regarded as a subset of places. A finite occurence net $\mathcal{B}$ can be regarded as a Petri net, where the initial marking is $M_0 = \min(\mathcal{B})$.

Branching processes and unfoldings. A *branching process* of Petri net $\mathcal{P}$ is a pair $\mathcal{B} = (\mathcal{O}, \varphi)$, where $\mathcal{O}$ is an occurrence net, and φ is an homomorphism from $\mathcal{O}$ to $\mathcal{P}$ regarded as nets, such that: 1/ the restriction of φ to $\min(\mathcal{O})$ is a bijection between $\min(\mathcal{O})$ and M_0 (the set of initially marked places), and 2/ for all $e, e' \in E$, ${}^\bullet e = {}^\bullet e'$ and $\varphi(e) = \varphi(e')$ together imply $e = e'$. By abuse of notation, we shall sometimes write $\min(\mathcal{B})$ instead of $\min(\mathcal{O})$.

The set of all branching processes of Petri net $\mathcal{P}$ is uniquely defined, up to an isomorphism (i.e., a renaming of the conditions and events), and we shall not distinguish isomorphic

branching processes. For $\mathcal{B}, \mathcal{B}'$ two branching processes, $\mathcal{B}'$ is a *prefix* of $\mathcal{B}$, written $\mathcal{B}' \sqsubseteq \mathcal{B}$, if there exists an injective homomorphism ψ from $\mathcal{B}'$ into $\mathcal{B}$, such that $\psi(\min(\mathcal{B}')) = \min(\mathcal{B})$, and the composition $\varphi \circ \psi$ coincides with φ', where $\circ$ denotes the composition of maps. By theorem 23 of [12], there exists (up to an isomorphism) a unique maximum branching process according to $\sqsubseteq$, we call it the *unfolding* of $\mathcal{P}$, and denote it by $\mathcal{U}_\mathcal{P}$. The unfolding of $\mathcal{P}$ possesses the following universal property: for every occurrence net $\mathcal{O}$, and every homomorphism $\phi : \mathcal{O} \mapsto \mathcal{P}$, there exists an *injective* homomorphism $\iota : \mathcal{O} \mapsto \mathcal{U}_\mathcal{P}$, such that:

$$\phi \;\; = \;\; \varphi \circ \iota, \tag{1}$$

where φ denotes the homomorphism associated to $\mathcal{U}_\mathcal{P}$. Decomposition (1) expresses that $\mathcal{U}_\mathcal{P}$ "maximally unfolds" $\mathcal{P}$. If $\mathcal{P}$ is itself an occurrence net and $M_0 = \min(\mathcal{P})$ holds, then $\mathcal{U}_\mathcal{P}$ identifies with $\mathcal{P}$.

As announced, configurations are adequate representations of firing sequences of Petri net $\mathcal{P}$. Let $M_0, M_1, M_2, \dots$ be a maximal firing sequence of $\mathcal{P}$, and let $M_{k-1}[t_k\rangle M_k$ be the associated sequence of fired transitions. Then there exists a unique maximal (for set inclusion) configuration κ of $\mathcal{P}$ having the following properties: κ is the union of a sequence $e_1, e_2, \dots$ of events and a sequence $\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \dots$ of cuts, such that, for each $k > 0$, $\varphi(\mathbf{c}_k) = M_k$, $\varphi(e_k) = t_k$, and $\mathbf{c}_{k-1} \supseteq {}^\bullet e_k, e_k^\bullet \subseteq \mathbf{c}_k$. Conversely, each maximal configuration of $\mathcal{P}$ defines a maximal firing sequence, which is unique up to the interleaving of structurally concurrent transitions—transitions t and t' are structurally concurrent iff ${}^\bullet t' \cap ({}^\bullet t \cup t^\bullet) = \emptyset$ and ${}^\bullet t \cap ({}^\bullet t' \cup t'^\bullet) = \emptyset$.

Example 1.   Fig. 2 shows the example we will use throughout this paper. A Petri net $\mathcal{P}$ is shown on the left. Its places are $1, 2, 3, 4, 5, 6, 7$, and its transitions are i, ii, iii, iv, v, vi. Places constituting the initial marking are encircled in thick.

A branching process $\mathcal{B} = (\mathcal{O}, \varphi)$ of $\mathcal{P}$ is shown on the right. Its conditions are depicted by circles, and its events are figured by boxes. Each condition b of $\mathcal{B}$ is labeled by $\varphi(b)$, a place of $\mathcal{P}$. Each event e of $\mathcal{B}$ is labeled by $\varphi(e)$, a transition of $\mathcal{P}$. A configuration of Petri net $\mathcal{P}$ is shown in grey. Note that the minimal condition labeled by 7 is branching in $\mathcal{B}$, although it is not branching in $\mathcal{P}$ itself. The reason is that, in $\mathcal{P}$, the token can freely move along the circuit $1 \to ii \to 2 \to iii \to 1$, and resynchronize afterwards with the token sitting in 7.

The mechanism for constructing the unfolding of Petri net $\mathcal{P}$ is illustrated in the middle, it is informally explained as follows. Put the three conditions labeled by the initial marking of $\mathcal{P}$, this is the minimal branching process of $\mathcal{P}$. Then, for each constructed branching process $\mathcal{B}$, select a co-set X of $\mathcal{B}$, which is labeled by the preset ${}^\bullet t$ of some transition t of $\mathcal{P}$, and has no event labeled by t in its postset within $\mathcal{B}$. Append to X a net isomorphic to ${}^\bullet t \to t \to t^\bullet$ (recall that $\varphi({}^\bullet t) = X$), and label its additional nodes by t and $t^\bullet$, respectively. Performing this recursively yields all possible finite branching processes of $\mathcal{P}$. Their union is the unfolding $\mathcal{U}_\mathcal{P}$.
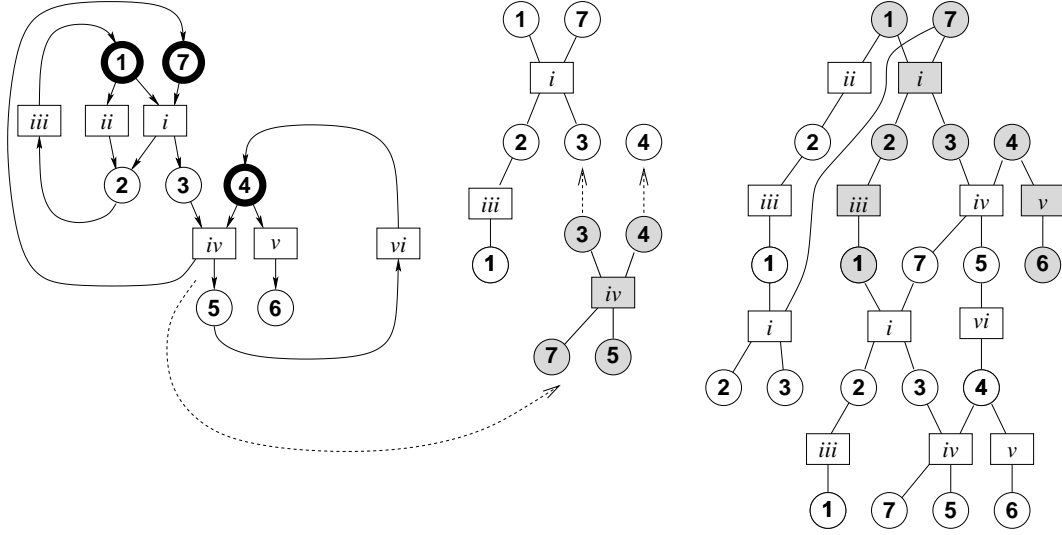
Figure 2: Example 1 (left), a configuration (middle), and a branching process (right). For this and subsequent examples, we take the following convention for drawing Petri nets and occurrence nets. In Petri nets, the flow relation is depicted using *directed arrows.* In occurrence nets, since no cycle occurs, the flow relation *progresses downwards,* and therefore there is no need to figure them via directed arrows, standard solid lines are used instead.

Labeled nets and their products. For $\mathcal{P} = (P, T, \rightarrow)$ a net, a *labeling* is a map $\lambda : T \longmapsto A$, where A is some finite alphabet. A net $\mathcal{P} = (P, T, \rightarrow, \lambda)$ equipped with a labeling λ is called a *labeled net.* For $\mathcal{P}_i = \{P_i, T_i, \rightarrow_i, \lambda_i\}$, $i = 1, 2$, two labeled nets, their product $\mathcal{P}_1 \times \mathcal{P}_2$ is the labeled net defined as follows:

$$\mathcal{P}_1 \times \mathcal{P}_2 = (P, T, \rightarrow, \lambda). \tag{2}$$

In (2), $P = P_1 \uplus P_2$, where $\uplus$ denotes the disjoint union, and:

$$
T = \begin{array}{ll}
\{t_1 \in T_1 : \lambda_1(t_1) \in A_1 \setminus A_2\} & \text{(i)} \\
\cup & \{(t_1, t_2) \in T_1 \times T_2 : \lambda_1(t_1) = \lambda_2(t_2)\} & \text{(ii)} \\
\cup & \{t_2 \in T_2 : \lambda_2(t_2) \in A_2 \setminus A_1\}, & \text{(iii)}
\end{array}
$$

$$
p \rightarrow t \quad \text{iff} \quad \begin{array}{ll}
p \in P_1 \text{ and } p \rightarrow_1 t_1 & \text{for case (ii) or (i)} \\
p \in P_2 \text{ and } p \rightarrow_2 t_2 & \text{for case (ii) or (iii)}
\end{array}
$$

$$
t \rightarrow p \quad \text{iff} \quad \begin{array}{ll}
p \in P_1 \text{ and } t_1 \rightarrow_1 p & \text{for case (ii) or (i)} \\
p \in P_2 \text{ and } t_2 \rightarrow_2 p & \text{for case (ii) or (iii)}
\end{array}
$$

In cases (i,iii) only one net fires a transition and this transition has a private label, while the two nets synchronize on transitions with identical labels in case (ii). Petri nets and occurrence nets inherit the above notions of labeling and product.

2.2 Discussing asynchronous diagnosis on example 1

A labeled Petri net model.   Our first example of Fig. 2 is redrawn slightly differently in Fig. 3, in the form of a labeled Petri net. The example is now intepreted as two interacting

Figure 3: Example 1, two interacting components modelled as a labeled Petri net.

components, numbered 1 and 2. Component 2 uses the services of component 1 for its functioning, and therefore it may fail to deliver its service when component 1 is faulty.

Component 1 has two states: nominal, figured by place 1, and faulty, figured by place 2. When getting in faulty state, the component 1 emits an alarm β, which is associated to transition (*i*) and (*ii*) (cf. Fig. 2) as a label. The fault of component 1 is temporary, and therefore self-repair can occur, this is figured by the label ρ associated to transition (*iii*) (cf. Fig. 2).

Component 2 has three states, figured by places $4, 5, 6$. State 4 is nominal, state 6 indicates that component 2 is faulty, and state 5 indicates that component 2 fails to deliver its service, due to the failure of component 1. Fault 6 is permanent and cannot be repaired.

The fact that component 2 may fail to deliver its service due to a fault of component 1, is modelled by the shared place 3. The monitoring system of component 2 only detects that this component fails to deliver its service, it does not distinguish between the different reasons for this. Hence the *same* alarm α is attached to the two transitions (*iv,v*) as a label (cf. Fig. 2). Since fault 2 of component 1 is temporary, self-repair can also occur for component 2, when in faulty state 5. This self-repair is not synchronized with that of component 1, but is still assumed to be manifested by the same label ρ. Finally, place 7 guarantees that fault propagation, from component 1 to component 2, occurs only when the latter is in nominal state.

The grey area indicates where interaction occurs between the two components. The initial marking consists of the two nominal states $1, 4$, for each component. Labels (alarms α, β or self-repair ρ) attached to the different transitions or events, are generically referred to as *alarms* in the sequel.

The different setups considered, for diagnosis.   Three different setups can be considered:

$\mathbf{S}_1$: The successive alarms are recorded *in sequence* by a single supervisor, in charge of fault monitoring. The sensor and communication infrastructure guarantees that *causality is respected:* for any two alarms that are causally related (α causes α'), then α is recorded before α'.

$\mathbf{S}_2$: Each sensor records its local alarms in sequence, by respecting causality. The different sensors perform independently and asynchronously, and there is a single supervisor which collects the records from the different sensors. *No assumption is made on the communication infrastructure.* Thus any interleaving of the records from different sensors can occur, and possible causalities relating alarms collected at different sensors are lost.

$\mathbf{S}_3$: The fault monitoring is performed in a *distributed* way, by different supervisors cooperating asynchronously. Each supervisor is attached to a component, it records its local alarms in sequence, and can exchange supervision messages with the other supervisor, asynchronously. No assumption is made on the communication infrastructure.

In this paper we consider $\mathbf{S}_1$ and $\mathbf{S}_2$ (and generalizations of them), *but not distributed diagnosis $\mathbf{S}_3$* (for distributed diagnosis, the reader is referred to [16][17]). Note that Internet cannot be used as a communication infrastructure for setup $\mathbf{S}_1$, but it can be used for setup $\mathbf{S}_2$.

The different setups are illustrated in Fig. 4, which is a combination of Fig. 2 and Fig. 3. The labeled Petri net of Fig. 3 is redrawn, on the left, with the topology used in Fig. 2. In the middle, we redraw the configuration shown in grey in Fig. 2-right, call it κ, and we relabel its events by their associated alarms. Configuration κ expresses that component 1 went into its faulty state 2, and then was repaired; concurrently, component 2 moved to its faulty state 6, where self-repair cannot occur. Note that the transmission of the fault of component 1 to component 2, via place 3, is preempted, due to the fatal failure of component 2.

How alarms are recorded is modelled by the two occurrence nets shown in the third and fourth diagrams, we call them *alarm patterns.* In the third diagram, we assume the first setup, in which a single sensor is available to collect the alarms. Hence configuration κ produces the alarms β, α, ρ, recorded in sequence. This record is modelled by the linear alarm pattern shown in the third diagram. This alarm pattern has its events labeled by alarms, but its conditions are "blind", i.e., they have no label. This manifests the fact that
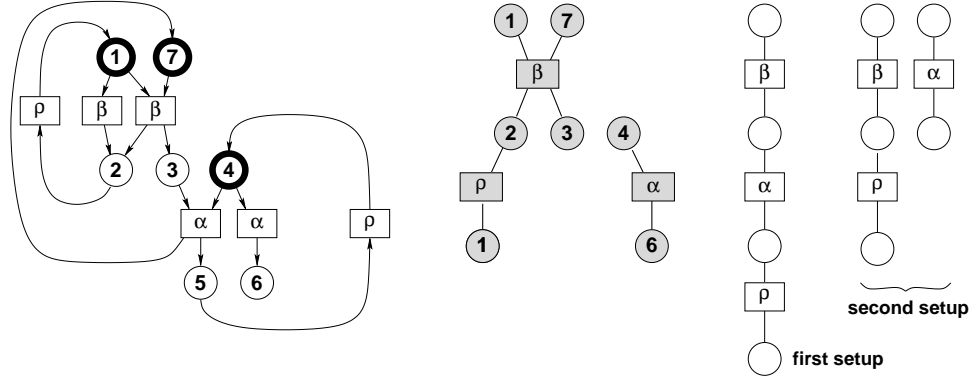
Figure 4: Example 1, a scenario involving a single sensor, and two independent sensors.

the different places of the Petri net, which are traversed while producing the alarms β, α, or ρ, are not observed.

Now, in the last diagram, we show the case of the second setup, in which β, ρ are collected by the first sensor, and α is collected by the second one, independently. The result is an alarm pattern composed of two concurrent parts, corresponding to the records collected by each sensor. When collected by the supervisor, these concurrent parts can interleave arbitrarily—this manifests asynchrony.

Asynchronous diagnosis. Alarm patterns are generically denoted by the symbol $\mathcal{A}$. Note that each sensor delivers, as an alarm pattern, some linear extension of the partial order of events it sees. But the causality relations involving pairs of events seen by different sensors, are lost. In general, *observations may add some causalities, may lose other ones, but they never reverse any of them.* Therefore, the only valid invariant between alarm pattern $\mathcal{A}$ and the configuration κ that produced it, is that $\mathcal{A}$ and κ possess a common linear extension. With this definition, we encompass setups $\mathbf{S}_1$ and $\mathbf{S}_2$ in a common framework. From the above discussion, we must accept as plausible explanations of an alarm pattern $\mathcal{A}$ any configuration κ such that $\mathcal{A}$ and κ possess a common linear extension. Such κ are said to *explain* $\mathcal{A}$. We are now ready to formalize our problem setting.

2.3 Asynchronous diagnosis: formal problem setting

Now, we formalize what an alarm pattern $\mathcal{A}$ is, and what it means, for $\mathcal{A}$, to be associated with some configuration κ. We are given the following objects, where the different notions have been introduced in subsection 2.1:

- A labeled Petri net $\mathcal{P} = (P, T, \rightarrow, M_0, \lambda)$, where the range of the labeling map λ is the alphabet of possible *alarms,* denoted by A, and

- its unfolding $\mathcal{U}_\mathcal{P} = (B, E, \rightarrow, \varphi)$.

Note the following chain of labeling maps:

$$\underbrace{E}_{\text{events}} \xrightarrow{\varphi} \underbrace{T}_{\text{transitions}} \xrightarrow{\lambda} \underbrace{A}_{\text{alarms}} \quad : \quad e \longmapsto \varphi(e) \longmapsto \lambda(\varphi(e)) \stackrel{\triangle}{=} \Lambda(e)\,, \qquad (3)$$

which defines the *alarm label* of event e, we denote it by $\Lambda(e)$—we call it also "alarm", for short, when no confusion can occur.

An *extension* of a net $\mathcal{P} = (P, T, \rightarrow)$ is any net obtained by adding places and flow relations but not transitions. Occurrence nets inherit this notion. An occurrence net induces a labeled partial order on the set of its events, extending this occurrence net induces an extension of this labeled partial order [2].

Two labeled occurrence nets $\mathcal{O} = (B, E, \rightarrow, \Lambda)$ and $\mathcal{O}' = (B', E', \rightarrow', \Lambda')$ are called *alarm-isomorphic* if there exists an isomorphism ψ, from $(B, E, \rightarrow)$ onto $(B', E', \rightarrow')$, seen as directed graphs, which preserves the alarm labels, i.e., such that $\forall e \in E : \Lambda'(\psi(e)) = \Lambda(e)$. Thus, two alarm-isomorphic occurrence nets can be regarded as identical if we are interested only in causalities and alarm labels.

Definition 1 (alarm pattern) *Consider $\mathcal{P}$, $\mathcal{U}_\mathcal{P}$, and Λ, as in (3). A labeled occurrence net $\mathcal{A} = (B_\mathcal{A}, E_\mathcal{A}, \rightarrow_\mathcal{A}, \lambda_\mathcal{A})$ is an* alarm pattern *of $\mathcal{P}$ iff:*

1. *Its labeling map $\lambda_\mathcal{A}$ takes its value in the alphabet A of alarms,*

2. *$\mathcal{A}$ is itself a configuration (it is conflict free), its set of conditions $B_\mathcal{A}$ is disjoint from that of $\mathcal{U}_\mathcal{P}$, and*

3. *There exists a configuration κ of $\mathcal{U}_\mathcal{P}$, such that $\mathcal{A}$ and κ possess extensions that are alarm-isomorphic.*

Assuming, for $\mathcal{A}$, a set of places disjoint from that of $\mathcal{U}_\mathcal{P}$, aims at reflecting that alarm patterns vehicle no information regarding hidden states of the original net. This justifies condition 2. Concerning condition 3 the allowed discrepancy between κ and $\mathcal{A}$ formalizes the possible loss of some causalities (e.g., due to independent and non synchronized sensors), and the possible adding of other ones (e.g., when sensors record their alarms in sequence). The key fact is that the information about the concurrency of events produced by the system cannot be observed by the supervisor. For instance, if the supervisor receives two alarm events α, β that are not causally related, then the net $\mathcal{P}$ may have produced $\alpha \perp\!\!\!\perp \beta$, or $\alpha \preceq \beta$, or $\alpha \succeq \beta$.

To refer to our context of diagnosis, we say that the configuration κ *can explain* $\mathcal{A}$. For $\mathcal{A}$ a given alarm pattern of $\mathcal{P}$, we denote by

$$\mathbf{diag}(\mathcal{A}) \qquad (4)$$

---

[2] Recall that the labeled partial order $(X, \preceq)$ is an *extension* of labeled partial order $(X', \preceq')$ if labeled sets X and X' are isomorphic, and $\preceq \; \supseteq \; \preceq'$ holds. When $(X, \preceq)$ is a total order, we call it a *linear* extension of $(X', \preceq')$.

the set of configurations κ of $\mathcal{U}_\mathcal{P}$ satisfying conditions 1,2,3 of definition 1. Due to asynchrony, ambiguity frequently occurs so that the set $\mathbf{diag}(\mathcal{A})$ is far from being a singleton. Therefore the issue of how to compute and represent this solution set efficiently is of great importance when large scale applications are considered. In the next subsection, we propose an adequate data structure to represent and manipulate the set $\mathbf{diag}(\mathcal{A})$ efficiently, we call it a *diagnosis net*.

3   Diagnosis nets: expressing asynchronous diagnosis by means of unfoldings

In this section, we provide explicit formulas for the solution of asynchronous diagnosis, in the form of suitable unfoldings.

A first natural idea is to represent $\mathbf{diag}(\mathcal{A})$ by the minimal subnet of unfolding $\mathcal{U}_\mathcal{P}$ which contains all configurations belonging to $\mathbf{diag}(\mathcal{A})$, we denote it by $\mathcal{U}_\mathcal{P}(\mathcal{A})$. Subnet $\mathcal{U}_\mathcal{P}(\mathcal{A})$ inherits canonically by restriction, the causality, conflict, and concurrence relations defined on $\mathcal{U}_\mathcal{P}$. Net $\mathcal{U}_\mathcal{P}(\mathcal{A})$ contains all configurations belonging to $\mathbf{diag}(\mathcal{A})$, but unfortunately it also contains undesirable maximal configurations *not* belonging to $\mathbf{diag}(\mathcal{A})$, as Fig. 5 reveals.
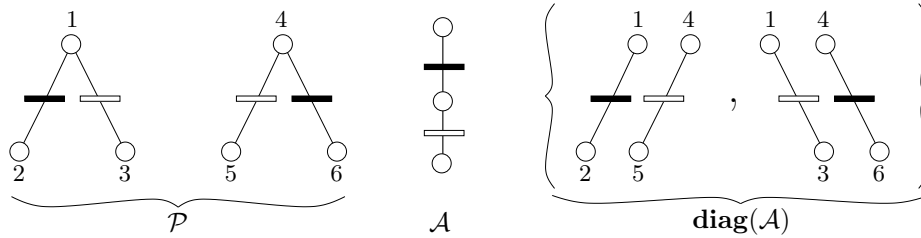

Figure 5: Example 2. Showing $\mathcal{P}, \mathcal{A}$, and $\mathbf{diag}(\mathcal{A})$. Note that $\mathcal{U}_\mathcal{P}(\mathcal{A}) = \mathcal{P}$.

In this figure, we show, on the left hand side, a Petri net $\mathcal{P}$ having the set of places $\{1, 4\}$ as initial marking, note that $\mathcal{P}$ is an occurrence net. In the middle, we show a possible associated alarm pattern $\mathcal{A}$. Alarm labels are figured by colors (black and white). The set $\mathbf{diag}(\mathcal{A})$ is shown on the right hand side, it comprises two configurations. Unfortunately the minimal subnet $\mathcal{U}_\mathcal{P}(\mathcal{A})$ of the original unfolding $\mathcal{P}$ which contains $\mathbf{diag}(\mathcal{A})$, is indeed identical to $\mathcal{P}$! Undesirable configurations are $\{(1, t_{12}, 2), (4, t_{46}, 6)\}$ and $\{(1, t_{13}, 3), (4, t_{45}, 5)\}$ (in these statements, t_{12} denotes the transition separating states 1 and 2). But configuration $\{(1, t_{12}, 2), (4, t_{46}, 6)\}$ is such that its two transitions t_{12}, t_{46} explain the same alarm event in $\mathcal{A}$. And the same holds for the other undesirable configuration.

Fig. 6 suggests an alternative solution, using the product $\mathcal{P} \times \mathcal{A}$ of $\mathcal{P}$ and $\mathcal{A}$, seen as labeled nets with respective labels λ and $\lambda_\mathcal{A}$ (see subsection 2.3 for these notations). The unfolding $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ is shown. The projection, on the set of nodes labelled by nodes from
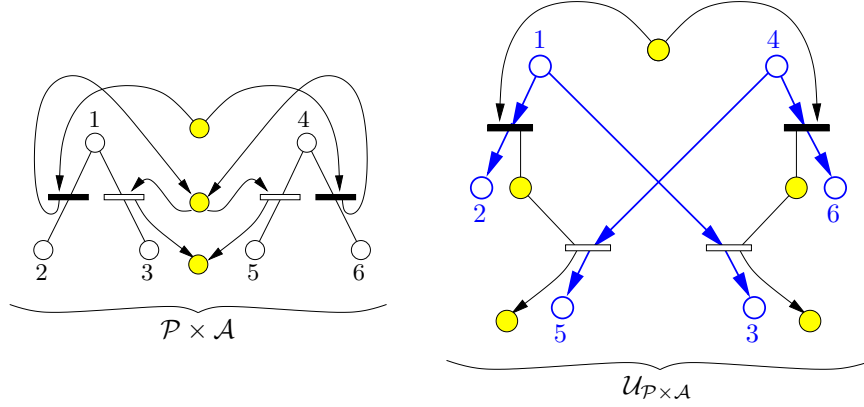
Figure 6: Example 2. Representing **diag**($\mathcal{A}$) by $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$.

$\mathcal{P}$, is depicted using larger arrows. The reader can verify that the corresponding set of maximal configurations coincides with **diag**($\mathcal{A}$). This suggests that $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ is an appropriate representation of **diag**($\mathcal{A}$). We formalize this in the theorem to follow. We use the notations from subsections 2.1 and 2.3, and we need a few more notations.

For $\mathcal{P} = (P, T, \rightarrow)$ a net and X a subset of its nodes, $\mathcal{P}_{|X}$ denotes the *restriction* of $\mathcal{P}$ to X, defined as

$$\mathcal{P}_{|X} \quad \triangleq \quad (P \cap X, T \cap X, \rightarrow_{|X}),$$

where the flow relation $\rightarrow_{|X}$ is defined as the restriction, to $X \times X$, of the flow relation $\rightarrow \subseteq (P \times T) \cup (T \times P)$ given on $\mathcal{P}$. Be careful that we restrict the flow relation, not its transitive closure.

Let $\mathcal{P} = (P, T, \rightarrow, M_0, \lambda)$ and $\mathcal{Q} = (Q, S, \rightarrow, N_0, \mu)$ be two labeled Petri nets, and $\mathcal{U} = (B, E, \rightarrow, \varphi)$ a sub-net of the unfolding $\mathcal{U}_{\mathcal{P} \times \mathcal{Q}}$. Define the labeled occurrence net **proj**$_{\mathcal{P}}(\mathcal{U})$, the *projection* of $\mathcal{U}$ on $\mathcal{P}$, as follows: 1/ restrict $\mathcal{U}$ to its subset of nodes labelled by nodes from $\mathcal{P}$, and 2/ project, onto T, the labels consisting of synchronized pairs of transitions belonging to $T \times S$. Let us formalize this construction. The set E of events of $\mathcal{U}$ decomposes as $E = E_{\mathcal{P}} \cup E_{\mathcal{P}, \mathcal{Q}} \cup E_{\mathcal{Q}}$, where $E_{\mathcal{P}}$ is the set of events labeled by transitions $t \in T$, $E_{\mathcal{Q}}$ is the set of events labeled by transitions $s \in S$, and $E_{\mathcal{P}, \mathcal{Q}}$ is the set of events labelled by pairs of synchronized transitions $(t, s) \in T \times S$. Then we define:

$$\mathbf{proj}_{\mathcal{P}}(\mathcal{U}) \quad \triangleq \quad \left(\mathcal{U}_{| \varphi^{-1}(P) \cup E_{\mathcal{P}} \cup E_{\mathcal{P}, \mathcal{Q}}}, \phi \right) \tag{5}$$

where the labeling map ϕ is defined as follows: if $b \in B$, then $\phi(b) = \varphi(b)$; if $e \in E_{\mathcal{P}}$, then $\phi(e) = \varphi(e)$; if $e \in E_{\mathcal{P}, \mathcal{Q}}$ is such that $\varphi(e) = (t, s)$, then $\phi(e) = t$. Hence **proj**$_{\mathcal{P}}(\mathcal{U})$ has $P \cup T$, the set of nodes of $\mathcal{P}$, as its label set.

Finally, for $\mathcal{O}$ an occurrence net, we denote by **config** ($\mathcal{O}$) the set of all its configurations.

Theorem 1 *Let $\mathcal{U}_{\mathcal{P}}$ be the unfolding of some Petri net $\mathcal{P}$, $\mathcal{A}$ an associated alarm pattern, and let $\mathbf{diag}(\mathcal{A})$ be defined as in (4). Consider the unfolding $\mathcal{U}_{\mathcal{P} \times \mathcal{A}} \triangleq (\bar{B}, \bar{E}, \rightarrow, \bar{\varphi})$, and its associated projections $\mathbf{proj}_{\mathcal{P}}(.)$ and $\mathbf{proj}_{\mathcal{A}}(.)$. Then, $\kappa \in \mathbf{diag}(\mathcal{A})$ iff:*

$$\exists \bar{\kappa} \in \mathbf{config}\,(\mathcal{U}_{\mathcal{P} \times \mathcal{A}}) \quad : \quad \mathbf{proj}_{\mathcal{P}}(\bar{\kappa}) = \kappa \text{ and } \mathbf{proj}_{\mathcal{A}}(\bar{\kappa}) = \mathcal{A}. \tag{6}$$

Note that the product $\mathcal{P} \times \mathcal{A}$ involves only synchronized transitions. Note also that every $\bar{\kappa}$ satisfying (6) must be a maximal configuration of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$. Theorem 1 expresses that $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ is an adequate representation of $\mathbf{diag}(\mathcal{A})$, we call it a *diagnosis net*.

Proof: We first prove the *if* part. Let $\bar{\kappa}$ be a configuration of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ such that $\mathbf{proj}_{\mathcal{A}}(\bar{\kappa}) = \mathcal{A}$, and define $\kappa = \mathbf{proj}_{\mathcal{P}}(\bar{\kappa})$. By definition of net extensions (cf. definition 1 and above), $\bar{\kappa}$ is an extension of both κ and $\mathcal{A}$. Hence, by definition 1, $\kappa \in \mathbf{diag}(\mathcal{A})$. This was the easy part.

We then prove the *only if* part. Select an arbitrary $\kappa \in \mathbf{diag}(\mathcal{A})$. We need to show the existence of a $\bar{\kappa}$ satisfying (6). Since $\kappa \in \mathbf{diag}(\mathcal{A})$, then κ and $\mathcal{A}$ possess two respective extensions, κ^e and $\mathcal{A}^e$, that are alarm isomorphic, let ψ be the corresponding isomorphism, from $E_{\mathcal{A}}$ (the set of events of $\mathcal{A}$), onto the set of events of κ. Note that κ^e possesses additional dummy conditions that are not labeled by places from $\mathcal{P}$, and $\mathcal{A}^e$ possesses conditions that do not belong to $\mathcal{A}$.

Consider the following configuration $\bar{\kappa}^e$, obtained as follows. Its set of events is the set of pairs $(\psi(e), e)$, where e ranges over $E_{\mathcal{A}}$. Then its set of conditions as well as its flow relation is defined by:

$$\text{flow relation of } \bar{\kappa}^e \quad : \quad \left\{ \begin{array}{rcl} {}^{\bullet}(\psi(e), e) & = & {}^{\bullet}\psi(e) \cup {}^{\bullet}e \\ (\psi(e), e)^{\bullet} & = & \psi(e)^{\bullet} \cup e^{\bullet} \end{array} \right. , \tag{7}$$

where the pre- and postset operations occurring on the right hand sides of the two equalities are taken from the *extensions* κ^e and $\mathcal{A}^e$. Informally speaking, $\bar{\kappa}^e$ is obtained by glueing together κ^e and $\mathcal{A}^e$ at their events associated via ψ. Note that $\bar{\kappa}^e$ is circuit free.

Now, erase, in $\bar{\kappa}^e$, the conditions that are neither labeled by places from $\mathcal{P}$, nor belong to $\mathcal{A}$ (such places originate from having extended κ and $\mathcal{A}$ into κ^e and $\mathcal{A}^e$, respectively). Call $\bar{\kappa}$ the so obtained configuration. By construction:

$$\mathbf{proj}_{\mathcal{P}}(\bar{\kappa}) = \kappa \quad \text{and} \quad \mathbf{proj}_{\mathcal{A}}(\bar{\kappa}) = \mathcal{A}.$$

Thus it remains to show that $\bar{\kappa} \in \mathbf{config}\,(\mathcal{U}_{\mathcal{P} \times \mathcal{A}})$. On the one hand, $\bar{\kappa}^e$ was circuit/conflict free and causally closed, then so is $\bar{\kappa}$, thus $\bar{\kappa}$ is a configuration. On the other hand, the flow relation and nodes of $\bar{\kappa}$ are also defined by formula (7), provided that the pre- and postset operations occurring on the right hand sides of the two equalities are taken from the *original* configurations κ and $\mathcal{A}$. By keeping in mind (7), we define the following labeling map $\bar{\phi}$ on $\bar{\kappa}$:

$$\bar{\phi}(\psi(e), e) = (\varphi(\psi(e)), e) \tag{8}$$

$$\forall \bar{b} \in {}^{\bullet}(\psi(e), e) \cup (\psi(e), e)^{\bullet} \quad : \quad \left\{ \begin{array}{llll} \text{for} & \bar{b} \in {}^{\bullet}e \cup e^{\bullet} & : & \bar{\phi}(\bar{b}) = \bar{b} \\ \text{for} & \bar{b} \in {}^{\bullet}\psi(e) \cup \psi(e)^{\bullet} & : & \bar{\phi}(\bar{b}) = \varphi(\bar{b}) \end{array} \right.$$

Hence, $\bar{\phi}$ is an homorphism from $\bar{\kappa}$ into $\mathcal{P} \times \mathcal{A}$. By using the universal property (1), there exists an injective homomorphism from $\bar{\kappa}$ into $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$. This, and the fact that $\bar{\kappa}$ was already proved to be a configuration, shows that $\bar{\kappa} \in \mathbf{config}(\mathcal{U}_{\mathcal{P} \times \mathcal{A}})$, this finishes the proof of the theorem. $\diamond$

Remark. Theorem 1 assumes the knowledge of the initial marking M_0 for Petri net $\mathcal{P}$. When only a set $\mathcal{M}_0$ of possible initial markings is known instead, simply augment $(P, T, \rightarrow)$ as follows. Add some additional place p_0 not belonging to P, for each possible initial marking $M_0 \in \mathcal{M}_0$ add one transition t_{M_0} to T with label α_0 not belonging to A, and add the branches $p_o \rightarrow t_{M_0} \rightarrow M_0$ to the flow relation. To account for this additional places and transitions, add to $\mathcal{A}$ a dummy prefix of the form $b_0 \rightarrow e_0 \rightarrow \min(\mathcal{A})$, where event e_0 has label α_0. Then, apply theorem 1 to the so augmented Petri net.

Example 1, illustration of diagnosis nets, and comparison with the use of the marking graph. Fig. 7 shows an illustration of theorem 1. In this figure we show the Petri
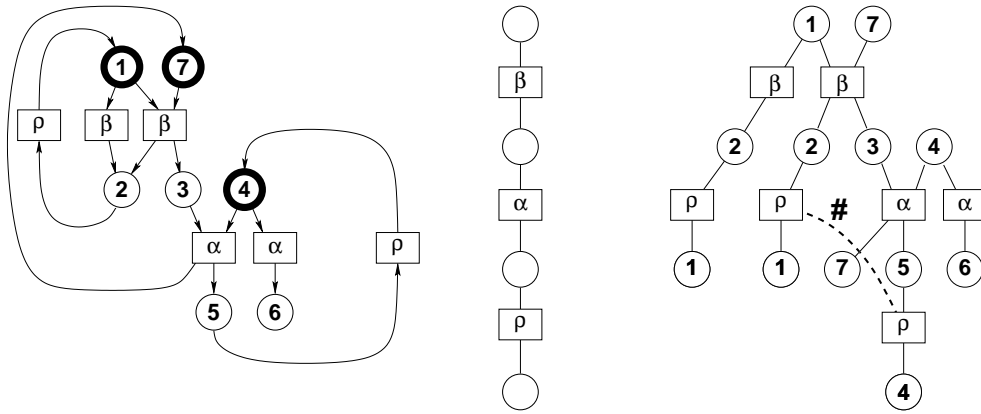

Figure 7: Example 1, diagnosis net, an illustration of theorem 1.

net $\mathcal{P}$ of Fig. 3 (left), an associated alarm pattern $\mathcal{A}$ (middle), and the net $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, restricted to its nodes labeled by nodes from $\mathcal{P}$ (right). We show in dashed-thick the additional conflict relation between the two otherwise concurrent events labeled by the same alarm ρ. This conflict is inherited from the sharing of a common condition, not shown, belonging to $\mathcal{A}$. It is easily checked that $\mathbf{diag}(\mathcal{A})$ is adequately represented by this diagram[3].

Four alternative explanations are delivered by this diagnosis, this reflects the ambiguity resulting from asynchrony in this example. Explanation 1: component 1 gets in its faulty state without causing damage to component 2, and then gets self-repaired; independently,

---

[3]The restriction, to its events, of this data structure, is an *event structure* according to Winskel's definition [28][36].

component 2 gets into its fatal faulty state 6; thus, for this scenario, $(\beta \prec \rho) \perp\!\!\!\perp \alpha$ holds. Explanation 2: component 1 gets in its faulty state while causing damage to component 2, and then gets self-repaired; independently, component 2 gets into its fatal faulty state 6; again, for this scenario, $(\beta \prec \rho) \perp\!\!\!\perp \alpha$ holds. Explanation 3: component 1 gets in its faulty state while causing damage to component 2; consequently, component 2 fails to delivers its service and gets into its state 5, where it subsequently gets self-repaired; for this scenario, we have $\beta \prec \rho \prec \alpha$. Explanation 4: component 1 gets in its faulty state while causing damage to component 2; consequently, component 2 fails to delivers its service; independently component 1 gets self-repaired; thus $\beta \prec (\rho \perp\!\!\!\perp \alpha)$ holds for this scenario.

Fig. 8 compares diagnosis nets with the use of marking graphs. The reader is referred again to our running example 1 (shown in Fig. 7), call it $\mathcal{P}$. In the first diagram we show the
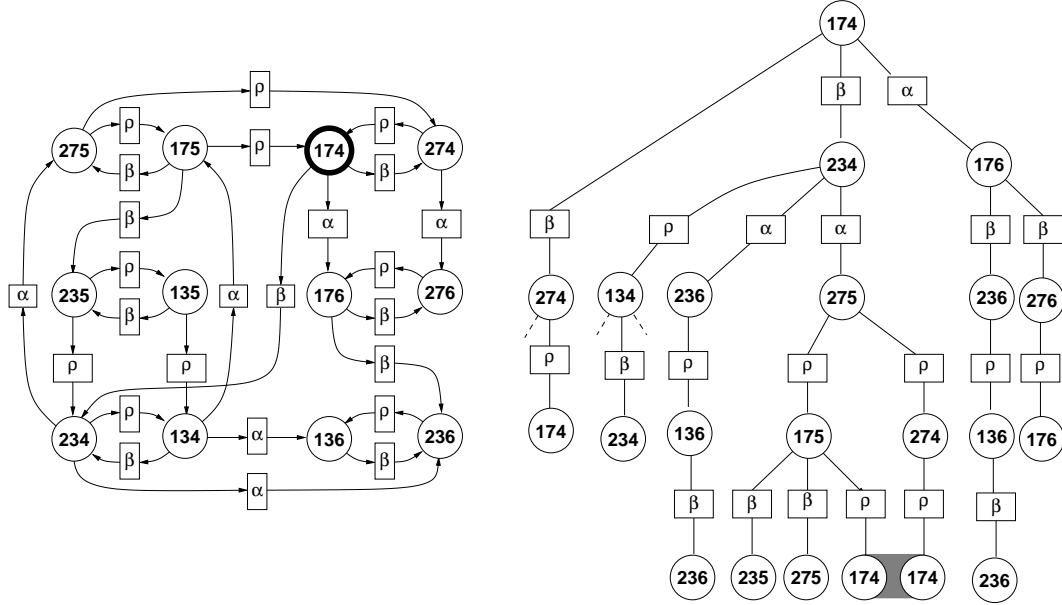

Figure 8: Marking graph of example 1 (left), and unfolding (right).

marking graph of $\mathcal{P}$, denoted by $\mathcal{M}(\mathcal{P})$. It is a labeled Petri net whose places are labeled by the reachable markings of $\mathcal{P}$, shown by the combination of the places composing the different markings. We identify the places of the marking graph $\mathcal{M}(\mathcal{P})$ with the markings of $\mathcal{P}$. Then, $M[t\rangle M'$ in $\mathcal{P}$ iff $M \to \tau \to M'$ in $\mathcal{M}(\mathcal{P})$. Transition τ of $\mathcal{M}(\mathcal{P})$ is then labeled by transition t of $\mathcal{P}$. In Fig. 8 we have labeled instead the transitions τ of $\mathcal{M}(\mathcal{P})$ by the alarm labels $\lambda(t) = \alpha, \beta, \rho$ of the associated transitions t from $\mathcal{P}$.

The pre/postset of each transition of $\mathcal{M}(\mathcal{P})$ is a singleton, hence there is no concurrency, and $\mathcal{M}(\mathcal{P})$ represents an automaton. Note the diamond composed of the two branches

$275 \to \rho \to 175 \to \rho \to 174$ and $275 \to \rho \to 274 \to \rho \to 174$, it represents the two possible interleavings of the concurrent transitions labeled by ρ in $\mathcal{P}$.

We can still regard $\mathcal{M}(\mathcal{P})$ as a Petri net, and consider its unfolding $\mathcal{U}_{\mathcal{M}(\mathcal{P})}$, shown in part in the second diagram (some flow relations are sketched in dashed, to save space). Now, we can safely merge the two conditions labeled by 174 in the bottom of this diagram. The reasons for this are the following: 1/ they are both labeled by the same state (namely 174), hence they possess identical continuations, and, 2/ their causal closures are labeled by the same alarm sequence $\beta, \alpha, \rho, \rho$, i.e., explain the same sequences of alarms. Merging the two conditions labeled by 174 in the bottom of the second diagram yields a *lattice,* i.e., a labeled net with branching and joining conditions but no circuit, we denote it by $\mathcal{L}_{\mathcal{M}(\mathcal{P})}$. Lattices are not new, they are the data structures used when applying the Viterbi algorithm for maximum likelihood estimation of hidden state sequences in stochastic automata.

Being linear and not branching any more, $\mathcal{L}_{\mathcal{M}(\mathcal{P})}$ is a more compact data structure than the unfolding $\mathcal{U}_{\mathcal{M}(\mathcal{P})}$. The reason for merging the two places labeled by 174 in $\mathcal{U}_{\mathcal{M}(\mathcal{P})}$ is the diamond occuring in $\mathcal{M}(\mathcal{P})$. But this diamond manifests the concurrency of the two self-repairing transitions, and the unfolding $\mathcal{U}_{\mathcal{P}}$ of $\mathcal{P}$, shown in Fig. 2, already handles this properly: the marking 174 is not duplicated in $\mathcal{U}_{\mathcal{P}}$, unlike in $\mathcal{U}_{\mathcal{M}(\mathcal{P})}$. In fact, this lattice corresponds to a prefix of the unfolding shown in Fig. 2. The unfolding of Fig. 2 is more compact, but in turn, building co-sets requires some processing, whereas this requires no processing for the unfolding of Fig. 8, since co-sets are just places. Therefore, for applications in which memory constraints prevail over processing speed, unfoldings should be preferred. Still, the generalization of lattices to Petri nets is of interest, and their definition and use for diagnosis is investigated in section 5.

4 Algorithms

In this section, we detail the algorithms for the construction of diagnosis nets. In subsection 4.2 we consider the general case considered in theorem 1, this encompasses setups $\mathbf{S}_1$ and $\mathbf{S}_2$ of subsection 2.2. Then, in subsection 4.3, we focus on $\mathbf{S}_1$, for which we give an improved algorithm. In subsection 4.1 we first describe the framework we need for the description of these algorithms.

4.1 Algorithms as pattern matching rules

As a prerequisite, we formally state an inductive construction of the unfolding $\mathcal{U}_{\mathcal{P}}$ of a Petri net $\mathcal{P}$ in the form of a nested family of branching processes $\mathcal{B}$—this construction is borrowed from [15], it was illustrated in Fig. 2-middle.

We use the following notations. The conditions of $\mathcal{U}_{\mathcal{P}}$ have the form (e, p), where e is an event of $\mathcal{U}_{\mathcal{P}}$ and p a place of $\mathcal{P}$. Similarly, events of $\mathcal{U}_{\mathcal{P}}$ have the form (X, t), where X is a co-set of conditions belonging to $\mathcal{U}_{\mathcal{P}}$, and t is a transition of $\mathcal{P}$. The homomorphism φ, from $\mathcal{U}_{\mathcal{P}}$ to $\mathcal{P}$, is given by $\varphi(e, p) = p$, and $\varphi(X, t) = t$. And the flow relation on $\mathcal{U}_{\mathcal{P}}$ is given by ${}^{\bullet}(e, p) = e$, and ${}^{\bullet}(X, t) = X$. Conditions (nil, p) are those having no input event, i.e., the

distinguished symbol *nil* is used for the minimal conditions of $\mathcal{U}_\mathcal{P}$. Hence, we represent a branching process as a pair $\mathcal{B} = (B, E)$ of conditions and events, and the flow relation and homomorphism are derived implicitly, using the above convention. The term $\mathbf{co}_B$ denotes the set of co-sets of B.

The set of *branching processes* of $\mathcal{P} = (P, T, \rightarrow, M_0)$ can be inductively constructed as follows:

- $(\{(nil, p), p \in M_0\}, \emptyset)$ is a branching process of $\mathcal{P}$.

- For (B, E) a branching process, $t \in T$, and $X \in \mathbf{co}_B$ such that $\varphi(X) = {}^\bullet t$, then the following term is also a branching process of $\mathcal{P}$:

$$(B \cup \{ (e, p) \mid p \in t^\bullet \}, E \cup \{e\}), \quad \text{where} \quad e = (X, t). \tag{9}$$

If $e \notin E$ we call e a *possible extension* of (B, E), and we call the corresponding extended branching process a *continuation* of (B, E) by e. The inductive construction (9) can be expressed in the form of a pattern matching rule:

$$\frac{\text{if precondition } \{\dots\} \text{ holds,}}{\begin{array}{c}\text{then, possible extension } \{\dots\} \\ \text{and its postset } \{\dots\} \text{ result.}\end{array}} \quad : \quad \frac{X \in \mathbf{co}_B, \ \varphi(X) = {}^\bullet t}{\begin{array}{c} e = (X, t) \\ e^\bullet = \{ (e, p) \mid p \in t^\bullet \}\end{array}} \tag{10}$$

where the three $\{\dots\}$ denote the corresponding three statements shown on the right hand side. In rule (10), *it is understood that $e = (X, t)$ is a possible extension of the current branching process, meaning that $e \notin E$ (the current event set),* this will not be repeated in the sequel. Most importantly, rule (10) applies *asynchronously,* meaning that the continuation can be performed in any order, from the different co-sets which have possible extensions in the current branching process.

4.2   Asynchronous diagnosis

The raw rules: computing $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$.   We first provide the rules for the computation of the unfolding $\mathcal{U}_{\mathcal{P}_1 \times \mathcal{P}_2}$, for two nets $\mathcal{P}_1$ and $\mathcal{P}_2$. For two nets $\mathcal{P}_1$ and $\mathcal{P}_2$, p_i (resp. t_i) denotes generically a place (resp. transition) of net $\mathcal{P}_i$, and the labeling map is denoted by λ_i. The homomorphism of the unfolding under construction is denoted by φ. Using these notations, we have the following rules for inductively constructing the branching processes (B, E) of $\mathcal{P}_1 \times \mathcal{P}_2$, whose union forms the unfolding $\mathcal{U}_{\mathcal{P}_1 \times \mathcal{P}_2}$, cf. (2):

$$\text{for } i = 1, 2 : \frac{X_i \in \mathbf{co}_{B_i}, \ \lambda_i(t_i) \text{ is private, and } \varphi(X_i) = {}^\bullet t_i}{\begin{array}{c} e_i = (X_i, t_i) \\ e_i^\bullet = \{ (e_i, p_i) \mid p_i \in t_i^\bullet \}\end{array}} \tag{11}$$

$$\frac{X_1 \cup X_2 \in \mathbf{co}_B, \ \lambda_1(t_1) = \lambda_2(t_2), \ \forall i = 1, 2 : \ \varphi(X_i) = {}^\bullet t_i}{\begin{array}{c} e = (X_1 \cup X_2, (t_1, t_2)) \\ e^\bullet = \{ (e, p) \mid p \in t_1^\bullet \cup t_2^\bullet \}\end{array}} \tag{12}$$

Rule (11) performs a local continuation involving a single component, whereas rule (12) performs a synchronized continuation. Thanks to theorem 1, the above rules (11,12) can be specialized to implement the inductive computation of the branching processes (B, E) of $\mathcal{P} \times \mathcal{A}$: simply discard rule (11) since no private label is involved. This yields:

$$\frac{X \cup X_{\mathcal{A}} \in \mathbf{co}_B \,,\, \lambda(t) = \lambda_{\mathcal{A}}(e^{\mathcal{A}}) \,,\, \varphi(X) = {}^{\bullet}t \,,\, \varphi(X_{\mathcal{A}}) = {}^{\bullet}e^{\mathcal{A}}}{\begin{array}{c} e = (X \cup X_{\mathcal{A}}, (t, e^{\mathcal{A}})) \\ e^{\bullet} = \left\{ (e, p) \,\Big|\, p \in t^{\bullet} \cup e^{\mathcal{A}\bullet} \right\} \end{array}}, \tag{13}$$

where $e^{\mathcal{A}}$ denotes a generic event of $\mathcal{A}$. Since the presence of the term $X \cup X_{\mathcal{A}}$ in the preset of the extension e always requires the corresponding precondition $X \cup X_{\mathcal{A}} \in \mathbf{co}_B$, *we shall omit the term $X \cup X_{\mathcal{A}} \in \mathbf{co}_B$ in the precondition of the rules in the sequel.* Thus rule (13) will be simply written:

$$\frac{\lambda(t) = \lambda_{\mathcal{A}}(e^{\mathcal{A}}) \,,\, \varphi(X) = {}^{\bullet}t \,,\, \varphi(X_{\mathcal{A}}) = {}^{\bullet}e^{\mathcal{A}}}{\begin{array}{c} e = (X \cup X_{\mathcal{A}}, (t, e^{\mathcal{A}})) \\ e^{\bullet} = \left\{ (e, p) \,\Big|\, p \in t^{\bullet} \cup e^{\mathcal{A}\bullet} \right\} \end{array}}$$

Since continuations can occur from any co-set of the current branching process, the whole branching process must be continuously maintained, for possible continuation, along the construction of the unfolding. Of course this data structure is of rapidly increasing complexity, and this makes the general algorithm based on rule (13) quite cumbersome. Also, in this general case, explanations of an alarm can occur with an arbitrary long delay, but this is the unavoidable price to pay for handling asynchrony with no restriction on the allowed sensor setup.

Refining the rules. Let us investigate how to refine rule (13). Theorem 1 states that $\mathbf{diag}(\mathcal{A})$ is in one-to-one correspondence with the set of $\bar{\kappa}$'s satisfying (6). Thus, only a subnet of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ has to be computed, not all of it. We investigate this issue next. Consider the unfolding $\mathcal{U}_{\mathcal{P} \times \mathcal{A}} = (B, E)$. For $\mathcal{A}$ fixed, and $\mathcal{A}'$ a subnet of $\mathcal{A}$, denote by

$$\mathbf{explain}\ \mathcal{A}' \tag{14}$$

the maximal subnet $\mathcal{U}$ of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, such that: 1/ all events of $\mathcal{U}$ are labeled by events of $\mathcal{A}'$, and 2/ $\min(\mathcal{U})$ and $\max(\mathcal{U})$ are conditions. The subnet $\mathbf{explain}\ \mathcal{A}'$ collects all events of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ which can explain some alarm belonging to $\mathcal{A}'$. For $\mathcal{V} \sqsubseteq \mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ a branching process of $\mathcal{P} \times \mathcal{A}$, set:

$$\mathcal{A}_{\mathcal{V}}^{term} \quad = \quad \max \left\{ \mathcal{A}' \sqsubseteq \mathcal{A} \mid (\mathbf{explain}\ \mathcal{A}') \subseteq \mathcal{V} \right\}. \tag{15}$$

Then, $\mathcal{A}_{\mathcal{V}}^{term}$ is the *terminated* prefix of $\mathcal{A}$, i.e., no further continuation of $\mathcal{V}$ will provide a new explanation for $\mathcal{A}_{\mathcal{V}}^{term}$. Symmetrically, keeping in mind that $\mathcal{U}_{\mathcal{P} \times \mathcal{A}} = (B, E)$, set:

$$\mathcal{A}_{\mathcal{V}}^{fut} \quad = \quad \max \left\{ \mathcal{A}' \subseteq \mathcal{A} \mid (\mathbf{explain}\ \mathcal{A}') \cap \mathcal{V} \cap E = \emptyset \right\}, \tag{16}$$

it collects the *future* alarms, that have not yet been considered at all in $\mathcal{V}$. Note the subnet symbol $\subseteq$ in (16), indicating that $\mathcal{A}_{\mathcal{V}}^{fut}$ is not a prefix of $\mathcal{A}$ (it is in fact a postfix of $\mathcal{A}$). Also, the two "max" in (15) and (16) are well defined, since the corresponding sets of $\mathcal{A}'$'s are stable under union. In general, the set $\mathcal{A}_{\mathcal{V}}^{act}$ of *active alarms* satisfies:

$$\mathcal{A}_{\mathcal{V}}^{act} \triangleq \mathcal{A} \setminus \left(\mathcal{A}_{\mathcal{V}}^{term} \cup \mathcal{A}_{\mathcal{V}}^{fut} \right) \neq \emptyset, \tag{17}$$

and $\mathcal{A}_{\mathcal{V}}^{act}$ can even have cardinality greater than 1. This means that alarms cannot be processed in sequence, i.e., there is no on-line algorithm. We shall see, however, that $\mathcal{A}_{\mathcal{V}}^{act} = \emptyset$ holds for a certain increasing chain of $\mathcal{V}$'s, for setup $\mathbf{S}_1$. In general, refined rules must maintain the triple $(\mathcal{A}_{\mathcal{V}}^{term}, \mathcal{A}_{\mathcal{V}}^{act}, \mathcal{A}_{\mathcal{V}}^{fut})$.

For $\mathcal{A}'$ a prefix of $\mathcal{A}$ such that no node of $\mathcal{A}'$ is maximal in $\mathcal{A}$, denote by

$$\mathbf{stop}\,(\mathcal{A}') \tag{18}$$

the maximal subnet of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}'}$ possessing no continuation in $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, note that $\mathbf{stop}\,(\mathcal{A}')$ is a postfix of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}'}$, and no continuation of it will explain alarms belonging to $\mathcal{A} \setminus \mathcal{A}'$. Hence $\mathbf{stop}\,(\mathcal{A}')$ *should be pruned prior to further performing continuations of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}'}$*.

For an arbitrary branching process $\mathcal{V}$ of $\mathcal{P} \times \mathcal{A}$, we must prune $\mathcal{V}^{term} \triangleq \mathbf{stop}\,(\mathcal{A}_{\mathcal{V}}^{term})$, where $\mathcal{A}_{\mathcal{V}}^{term}$ is defined in (15), and keep only $\mathcal{V}^{act} \triangleq \mathcal{V} \setminus \mathcal{V}^{term}$, from which continuation can proceed.

By maintaining the above objects along the steps of the algorithm, refined versions of rule (13) can be derived. Using this technique, in the next subsection we focus on setup $\mathbf{S}_1$ and provide for it an improved algorithm.

4.3 An improved algorithm for the case in which causality is respected

Here we investigate the computation of the diagnosis net $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, and then of $\mathbf{diag}(\mathcal{A})$, for the case of $\mathcal{A}$ being a *totally ordered alarm pattern* of $\mathcal{P}$—this corresponds to setup $\mathbf{S}_1$. We start with some lemmas. The first lemma establishes that on-line algorithms exist for setup $\mathbf{S}_1$.

Lemma 1 *Let $\mathcal{A}' \sqsubseteq \mathcal{A}$ be a prefix of $\mathcal{A}$. Then:*

$$\mathcal{U}_{\mathcal{P} \times \mathcal{A}'} \quad \sqsubseteq \quad \mathbf{explain}\,\mathcal{A}'. \tag{19}$$

If $\mathcal{A}$ is totally ordered, *then we have:*

$$\mathcal{U}_{\mathcal{P} \times \mathcal{A}'} \quad = \quad \mathbf{explain}\,\mathcal{A}'. \tag{20}$$

Formula (20) says that, if $\mathcal{A}$ is totally ordered, then $\mathcal{U}_{\mathcal{P} \times \mathcal{A}'}$ contains *all* explanations of $\mathcal{A}'$. In this case, as soon as $\mathcal{U}_{\mathcal{P} \times \mathcal{A}'}$ has been constructed, we can forget $\mathcal{A}'$, this justifies the consideration of on-line algorithms—we insist that this does not hold in general, cf. (17)!

Proof: Inclusion (19) is obvious, so we need only to prove equality (20) under the assumption that $\mathcal{A}$ is totally ordered. This is the result of interest. It is trivial if $\mathcal{A}' = \emptyset$, so we can assume that this is not the case.

In the sequel of the proof, symbols $\bar{e}, \bar{f}$ denote events of the unfolding $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, hence, using the notations of (13), $\bar{e}$ has the form:

$$\bar{e} = (X \cup X_{\mathcal{A}}, (t, e^{\mathcal{A}})),$$

and $\phi_{\mathcal{A}}(\mathbf{proj}_{\mathcal{A}}(\bar{e})) = e^{\mathcal{A}}$, where $\phi_{\mathcal{A}}$ denotes the labeling map of $\mathbf{proj}_{\mathcal{A}}(\mathcal{U}_{\mathcal{P} \times \mathcal{A}})$ (cf. (5)). Also, $\preceq(\bar{e})$ denotes the configuration spaned by the causal closure of $\bar{e}$ in $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$. Similar notations and remarks hold for $\bar{f}$.

Pick $\bar{e} \in \mathbf{explain}\,\mathcal{A}'$, and set $\bar{\kappa} = \preceq(\bar{e})$. Pick $\bar{f} \in \bar{\kappa}$. Since $\bar{f} \preceq \bar{e}$, then either $e^{\mathcal{A}} \perp\!\!\!\perp f^{\mathcal{A}}$ or $f^{\mathcal{A}} \preceq e^{\mathcal{A}}$ holds ($e^{\mathcal{A}} \preceq f^{\mathcal{A}}$ is impossible, by definition of alarm patterns). But, since $\mathcal{A}$ is totally ordered, then only $f^{\mathcal{A}} \preceq e^{\mathcal{A}}$ can hold, thus $f^{\mathcal{A}}$ must belong to $\mathcal{A}'$, since $e^{\mathcal{A}} \in \mathcal{A}'$. Since this holds $\forall \bar{f} \in \bar{\kappa}$, then $\bar{\kappa} \subset \mathcal{U}_{\mathcal{P} \times \mathcal{A}'}$, therefore $\mathbf{explain}\,\mathcal{A}' \subseteq \mathcal{U}_{\mathcal{P} \times \mathcal{A}'}$, this proves the lemma. $\diamond$

For the following results, we assume $\mathcal{A}$ totally ordered. For $\mathcal{U}$ a subnet of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, denote by $\#(\mathcal{U})$ the subnet of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ comprising the nodes that are in conflict with *every* node of $\mathcal{U}$. The following theorem indicates how the pruning introduced in subsection 4.2 should be performed, it refines theorem 1 for the case in which $\mathcal{A}$ is totally ordered:

Theorem 2 *Assume that $\mathcal{A}$ is totally ordered.*

1. *If $\mathcal{A}'$ is the maximal strict prefix of $\mathcal{A}$, then $\mathbf{diag}(\mathcal{A})$ coincides with the set of all maximal configurations of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}} \setminus \mathbf{stop}\,(\mathcal{A}')$.*

2. *Consider a chain $\mathcal{A}' \sqsubset \mathcal{A}'' \sqsubseteq \mathcal{A}$ of prefixes of $\mathcal{A}$. Then:*

$$\mathbf{stop}\,(\mathcal{A}') = \#\,(\mathcal{W})\,, \text{ where } \mathcal{W} \stackrel{\triangle}{=} \mathcal{U}_{\mathcal{P} \times \mathcal{A}''} \setminus \mathcal{U}_{\mathcal{P} \times \mathcal{A}'}.$$

Proof: Point 1 follows from the definition (18) of $\mathbf{stop}\,(\mathcal{A}')$. Then, by lemma 1, we have $\mathcal{W} = (\,\mathbf{explain}\,\mathcal{A}'' \setminus \mathbf{explain}\,\mathcal{A}'\,)$, from which point 2 follows. $\diamond$

The following lemma is of lesser importance, but it will be useful for further optimizing our algorithm, by restricting the set of co-sets than can serve for possible continuation. For $\mathcal{U}$ a subnet of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, denote by $\perp\!\!\!\perp(\mathcal{U})$ the subnet of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ consisting of the nodes that are concurrent with *some* node of $\mathcal{U}$—note the difference with the definition of $\#(\mathcal{U})$.

Lemma 2 *Let $\mathcal{A}', \mathcal{A}''$, and $\mathcal{W}$ be as in point 2 of theorem 2. Set*

$$\mathbf{ext}\,(\mathcal{A}'') \stackrel{\triangle}{=} \mathcal{W} \cup \perp\!\!\!\perp(\mathcal{W}) \tag{21}$$

Then all possible extensions of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}''}$ have their preset contained in $\mathbf{ext}\,(\mathcal{A}'')$.

Proof: Assume this is not the case. Hence there exists a co-set X not contained in the subnet sitting on the right hand side of (21), and some event $e \in (\mathcal{U}_{\mathcal{P} \times \mathcal{A}} \setminus \mathcal{U}_{\mathcal{P} \times \mathcal{A}''})$, such that $^{\bullet}e = X$. Hence we must have:

$$X \cap (\prec(\mathcal{W}) \cup \#(\mathcal{W})) \;\; \neq \;\; \emptyset,$$

which implies that $e \in \#(\mathcal{W})$, a contradiction with theorem 2. $\diamond$

Using the above results, successive optimizations of the generic rule (13) are performed in several steps.

1. On-line computation of the successive branching processes, (B, E), of unfolding $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$.   Write $\mathcal{A} = (B_{\mathcal{A}}, E_{\mathcal{A}}, \to_{\mathcal{A}}, \lambda_{\mathcal{A}})$, with:

$$
\begin{aligned}
B_{\mathcal{A}} &= \{b_0, b_1, b_2, \ldots, b_n, \ldots\}, \\
E_{\mathcal{A}} &= \{e_1, e_2, \ldots, e_n, \ldots\}, \lambda_{\mathcal{A}}(e_k) \in A, \\
\forall n > 0 \quad &: \quad b_{n-1} \to_{\mathcal{A}} e_n \to_{\mathcal{A}} b_n,
\end{aligned}
\tag{22}
$$

where we recall that A is the alphabet of possible alarms—the superscipt $^{\mathcal{A}}$ has been removed from the events and conditions of $\mathcal{A}$, for the sake of clarity. In other words, the flow relation is obtained by interleaving alternatively one condition from $B_{\mathcal{A}}$ and one event from $E_{\mathcal{A}}$. Using these notations, rule (13) rewrites as follows:

$$
\frac{\lambda(t) = \lambda_{\mathcal{A}}(e_n) \;,\; \varphi(X) = {}^{\bullet}t \;,\; \varphi(X_{\mathcal{A}}) = b_{n-1}}{\begin{array}{c} e = (X \cup X_{\mathcal{A}}, (t, e_n)) \\ e^{\bullet} = \{(e, p) \,|\, p \in t^{\bullet}\} \cup \{(e, b_n)\} \end{array}}
\tag{23}
$$

Denote by $\mathcal{A}_n = (b_0 \to_{\mathcal{A}} e_1 \to_{\mathcal{A}} b_1 \to_{\mathcal{A}} \ldots \to_{\mathcal{A}} b_n)$ the prefix of length n of $\mathcal{A}$, and apply lemma 1 with $\mathcal{A}' := \mathcal{A}_n$, we get:

$$(\hat{B}_n, \hat{E}_n) \;\triangleq\; \mathbf{explain}\, \mathcal{A}_n \;=\; \mathcal{U}_{\mathcal{P} \times \mathcal{A}_n}
\tag{24}$$

(recall that we represent branching processes as a pair of condition and event sets). Formula (24) expresses that we can forget about $\mathcal{A}_n$ as soon as $\mathcal{U}_{\mathcal{P} \times \mathcal{A}_n}$ has been computed, and implement an *on-line* computation of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}_n}$:

$$(\hat{B}_n, \hat{E}_n) \;=\; \mathbf{R}_n^{(23)}(\hat{B}_{n-1}, \hat{E}_{n-1}), \;\; \text{where } \mathbf{R}_n^{(23)} \triangleq [\forall t : \mathbf{R}_{n,t}^{(23)}],
\tag{25}$$

and $\mathbf{R}_{n,t}^{(23)}$ denotes rule (23), for n and t seen as parameters. Formula (25) defines our *on-line* algorithm.

2. Computing the set $\{\bar{\kappa} \in \mathbf{config}\,(\mathcal{U}_{\mathcal{P} \times \mathcal{A}}) : \mathbf{proj}_{\mathcal{A}}(\bar{\kappa}) = \mathcal{A}\}$.   Theorem 1 indicates that we need only to compute those configurations $\bar{\kappa}$ of $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, such that $\mathbf{proj}_{\mathcal{A}}(\bar{\kappa}) = \mathcal{A}$. In

$\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, some configurations, while being maximal (for set inclusion) in $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, explain only a strict prefix of $\mathcal{A}$. To represent $\mathbf{diag}(\mathcal{A})$ exactly, such configurations must be removed. For this we use theorem 2. Consider

$$\delta \hat{E}_n \quad \triangleq \quad \hat{E}_n \setminus \hat{E}_{n-1},$$

the set of events added to $\hat{E}_{n-1}$ by applying the rule $\mathbf{R}_n^{(23)}$, and let $\#(\delta \hat{E}_n)$ denote the set of nodes belonging to $(\hat{B}_n, \hat{E}_n)$ that are in conflict with every node of $\delta \hat{E}_n$. Apply theorem 2 with $\mathcal{A}' := \mathcal{A}_{n-1}, \mathcal{A}'' := \mathcal{A}_n$, we deduce that $\mathbf{stop}(\mathcal{A}_{n-1}) = \#(\delta \hat{E}_n)$, i.e.,

$$\text{the nodes belonging to } \#(\delta \hat{E}_n) \text{ cannot belong} \atop \text{to a configuration explaining alarm } n. \tag{26}$$

Thus, for computing the set $\{\bar{\kappa}_n \in \mathbf{config}(\mathcal{U}_{\mathcal{P} \times \mathcal{A}}) : \mathbf{proj}_{\mathcal{A}}(\bar{\kappa}_n) = \mathcal{A}_n\}$ we must prune the nodes belonging to $\#(\delta \hat{E}_n)$.

This pruning can then be interleaved with the successive application of the on-line rule (25). Performing this yields the *desired sequence (B_n, E_n) of branching processes,* and all maximal configurations, $\bar{\kappa}_n$, of (B_n, E_n), are such that $\mathbf{proj}_{\mathcal{A}}(\bar{\kappa}_n) = \mathcal{A}_n$. Therefore, the following post-processing is applied after rule (25):

$$\mathbf{pruneR}_n \quad : \quad \text{remove } \#(\delta E_n) \text{ from } (B_n, E_n), \tag{27}$$

and rule (25) is modified as follows:

$$\mathbf{R}_n^{(23)} ; \mathbf{pruneR}_n. \tag{28}$$

The pruning mechanism (27) is illustrated in Fig. 9, the reader should compare with Fig. 7. In this figure, we have extended the $\mathcal{A}$ shown in Fig. 7 by adding one more alarm labeled α (middle). The branching process shown in the right is a continuation of the one shown in Fig. 7. The result of the pruning mechanism (27) is depicted in dark grey: corresponding nodes are pruned from the updated $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, which is therefore the white part of the diagram on the right. Note that the ambiguity has been removed, since the remaining net is now itself a single configuration. In fact, this figure shows directly this pruning mechanism on the restriction of diagnosis net $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, to the subset of its nodes that are labeled by nodes from $\mathcal{P}$.

3. Optimizing. We can still optimize rule (28) by noting that, in the term (B_n, E_n) resulting from applying this rule, not all places from B_n can serve for future continuations of (B_n, E_n)—compare this situation with the general one discussed at the end of subsection 4.2. For this, we use lemma 2. Denote by $\perp\!\!\!\perp(\delta E_n)$ the set of places belonging to (B_n, E_n) that are concurrent with *some* event belonging to δE_n—note the difference with the former definition of $\#(\delta E_n)$. Then, by lemma 2 we know that

$$\text{only the nodes belonging to } \delta E_n^{\bullet} \cup \perp\!\!\!\perp(\delta E_n) \atop \text{can serve for future continuations of } (B_n, E_n). \tag{29}$$
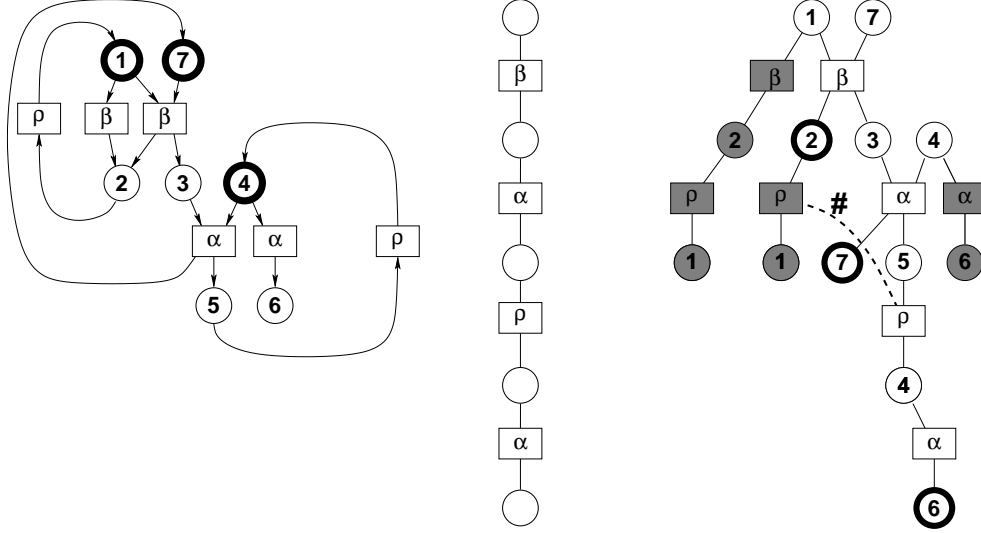
Figure 9: The pruning mechanism (27), and optimization (29).

Using claim (29), rule (23) rewrites as follows, note the modification of the pre-condition:

$$\mathbf{R}_{n,t}^{(30)} : \frac{\begin{array}{c} X \cup X_{\mathcal{A}} \subseteq \left(\delta E_{n-1}^{\bullet} \cup \mathbb{L}(\delta E_{n-1}) \right) \\ \lambda(t) = \lambda_{\mathcal{A}}(e_n) \ , \ \varphi(X) = {}^{\bullet}t \ , \ \varphi(X_{\mathcal{A}}) = b_{n-1} \end{array}}{\begin{array}{c} e = (X \cup X_{\mathcal{A}}, (t, e_n)) \\ e^{\bullet} = \{(e,p) \,|\, p \in t^{\bullet}\} \cup \{(e, b_n)\} \end{array}} \tag{30}$$

The additional precondition is then updated as follows, prior to handling the $n+1$st alarm:

$$\mathbf{optimR}_n \quad : \quad \delta E_n^{\bullet} \cup \mathbb{L}(\delta E_n) \ = \ \delta E_n^{\bullet} \cup \left(\mathbb{L}(\delta E_{n-1}) \setminus {}^{\bullet}\delta E_n \right) \tag{31}$$

Then, post-processing (27) applies after rule (30) as well, hence the optimized rule becomes:

$$\mathbf{R}_n^{(30)} \,;\, \mathbf{pruneR}_n \,;\, \mathbf{optimR}_n. \tag{32}$$

This optimization is illustrated in Fig. 9. In this figure, the set $\delta E_n^{\bullet} \cup \mathbb{L}(\delta E_n)$ consists of the three conditions encircled in thick, in the diagnosis net shown on the right hand side. Hence co-sets not contained in this set need not be tested, for possible continuation. Note that, unlike the pruning, this optimization does not modify the constructed branching process.

4. Maintaining co-sets. So far we have ignored the need for testing the condition $X \in \mathbf{co}_{B_{n-1}}$, see the remark at the end of subsection 4.1. But the optimization $\mathbf{optimR}(n)$ applies, hence only a postfix of the whole branching process is explored for performing the

continuation. Therefore we should avoid exploring backward the entire branching process under construction, in order to check the co-set property. So we need to maintain and update *explicitly* the co-set property, restricted to $\delta E_n^\bullet \cup \perp\!\!\!\perp(\delta E_n)$. We discuss this now.

Focus on rule (30). The following formula holds, to update the co-set property while processing the nth alarm:

$$\mathbf{co}_B \quad := \quad \mathbf{co}_B \, \cup \, \big(\bigcup_{Y \in \mathbf{co}_B \, , \, Y \supseteq X} Y[X \leftarrow e^\bullet] \big), \tag{33}$$

where we denote by $Y[X \leftarrow e^\bullet]$ the co-set Y in which X has been substituted with $e^\bullet$. Formula (33) possesses the following initialization and termination conditions:

$$\begin{aligned}
\text{initialization} &: \mathbf{co}_B &:= \mathbf{co}_{B_{n-1}}, \\
\text{termination} &: \mathbf{co}_{B_n} &:= \mathbf{co}_B.
\end{aligned}$$

Finally, rule $\mathbf{R}_{n,t}^{(30)}$ refines as:

$$\mathbf{R}_{n,t}^{(34)} : \frac{\begin{array}{c} X \cup X_\mathcal{A} \subseteq \big(\delta E_{n-1}^\bullet \cup \perp\!\!\!\perp(\delta E_{n-1})\big) \\ \lambda(t) = \lambda_\mathcal{A}(e_n) \, , \, \varphi(X) = {}^\bullet t \, , \, \varphi(X_\mathcal{A}) = b_{n-1} \end{array}}{e = (X \cup X_\mathcal{A}, (t, e_n))} \atop \begin{array}{c} e^\bullet = \{(e,p) \,|\, p \in t^\bullet\} \cup \{(e, b_n)\} \\ \mathbf{co}_B := \mathbf{co}_B \, \cup \, \big(\bigcup_{Y \in \mathbf{co}_B \, , \, Y \supseteq X} Y[X \leftarrow e^\bullet]\big) \end{array} \tag{34}$$

And the refined on-line algorithm is obtained by substituting $\mathbf{R}_{n,t}^{(34)}$ for $\mathbf{R}_{n,t}^{(30)}$ in (32). Note that maintaining on-line the concurrency relation is of low cost in this case, in contrast to the general case where continuations can be performed from far in the interior of the net under construction.

5 Lattice nets, and their use for asynchronous diagnosis

In this section we discuss how diagnosis nets can be made more compact, by using nets possessing branching *and joining* conditions, but no circuit. Such nets are new, we call them *lattice nets.* We will see that we can avoid unnecessary duplications that can occur in diagnosis nets.

Example 3. The previous examples are not sufficient to exhibit the need for considering lattice nets, hence we consider another one. This example is shown in Fig. 10. It is a simplified, and then symmetrized, version of example 1, Fig. 3, in which fault propagation can occur in both directions, through places 3 and 3′. We assume now that the two "self-repair" events are distinguished, whence the two different labels, ρ and σ. A branching process of example 3 is shown in Fig. 11. While only places $1, 1'$ are branching in the Petri net, conditions labelled by C, C' are infinitely branching in the unfolding, due to
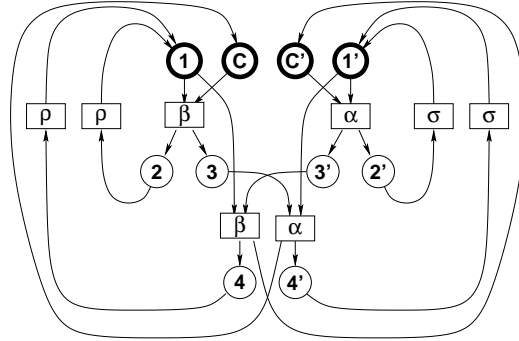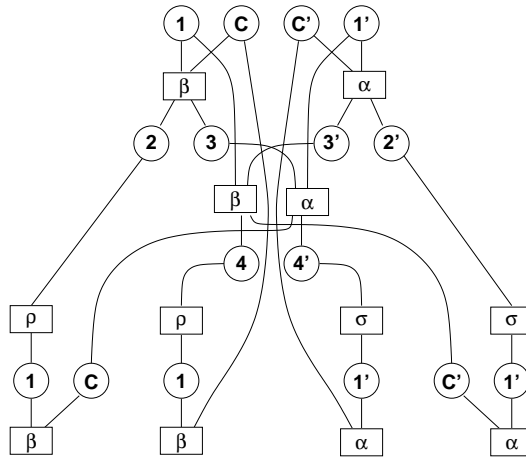
Figure 10: Example 3.

Figure 11: Unfolding example 3.

synchronizations. Unlike example 1, this example has symmetries that are not explained by differing interleavings. How a lattice net can be derived from this unfolding is analyzed in the following section.

5.1   Layers and lattice nets

Finding aggregates of conditions that are suitable for merging relates to finding an appropriate notion of "atomic progress of time", this is analyzed now. Branching processes are partially ordered by the prefix relation. We propose to interpret branching processes of a Petri net as a (partially ordered) time for it, in which the progress of time is materialized by the continuation of branching processes. But not every progress of time is of interest. Pre-

dictable stopping times and layers have been introduced for the first time in [4] to formalize a proper notion of atomic progress of time.

In the following, P_c denotes the set of branching places of Petri net $\mathcal{P}$, i.e., places having at least two different transitions in their postsets. Denote by $B_c \triangleq \{b \in B \ : \ \varphi(b) \in P_c\}$ the set of *branching conditions*.

A branching process $\mathcal{B} = \{B, E, \rightarrow, \varphi\}$ is called a *predictable stopping time* of the unfolding $\mathcal{U}_\mathcal{P}$ if it satisfies the following conditions:

(i) $\forall b \in B_c$, either $b_\mathcal{B}^\bullet = \emptyset$ or $b_\mathcal{B}^\bullet = b^\bullet$, where $b_\mathcal{B}^\bullet$ denotes the postset of condition b in the branching process $\mathcal{B}$, and $b^\bullet$ denotes, as usual, the postset of condition b in the whole unfolding $\mathcal{U}_\mathcal{P}$.

(ii) Each continuation of $\mathcal{B}$ contains at least one additional condition belonging to B_c.

Requirement (i) is the key one. It guarantees that the boundary of $\mathcal{B}$ does not cross conflicting sets of conditions (co-set $X \subset B_c$ is called *conflicting* if $\forall b \in X, \exists b' \in X : b' \neq b$ and $b^\bullet \cap b'^\bullet \neq \emptyset$). Requirement (ii) is stated for convenience, in order to avoid considering trivial progresses of time, in which no choice occurs. Having defined our proper notion of time, it is natural to consider the associated notion of "atomic progress of time".

Let $\mathcal{B}$ and $\mathcal{B}'$ be two predictable stopping times such that 1/ $\mathcal{B}'$ is strictly contained in $\mathcal{B}$, and 2/ there exists no predictable stopping time strictly containing $\mathcal{B}'$ and strictly contained in $\mathcal{B}$ (the set of all predictable stopping times is closed under intersection [4], hence the latter statement is sound). We call a *layer* the following suffix of $\mathcal{B}$:

$$L = (\mathcal{B} \setminus \mathcal{B}') \cup {}^\bullet(\mathcal{B} \setminus \mathcal{B}') \qquad (35)$$

The representation (35) of layer L is not unique. However, we note that, if decompositions $L = (\mathcal{B}_1 \setminus \mathcal{B}'_1) \cup {}^\bullet(\mathcal{B}_1 \setminus \mathcal{B}'_1) = (\mathcal{B}_2 \setminus \mathcal{B}'_2) \cup {}^\bullet(\mathcal{B}_2 \setminus \mathcal{B}'_2)$ hold, then L can also be defined using the pair $(\mathcal{B}_1 \cap \mathcal{B}_2, \mathcal{B}'_1 \cap \mathcal{B}'_2)$. Hence there exists a *minimal* pair $(\mathcal{B}, \mathcal{B}')$ such that representation (35) of layer L holds, we take it as the *canonical* representation of L and write it

$$L = \mathcal{B}/\mathcal{B}' . \qquad (36)$$

We collect below some useful results from [4]. **L** denotes the set of all layers of the considered unfolding.

1. For $L_1, L_2 \in \mathbf{L}$ two layers, either $L_1 \cap B_c^\bullet = L_2 \cap B_c^\bullet$ (L_1 and L_2 possess identical post-branching events), or $L_1 \cap B_c^\bullet \cap L_2 = \emptyset$ (L_1 and L_2 possess disjoint post-branching events). We introduce the following equivalence relation on **L**:

$$L_1 \asymp L_2 \quad \text{iff} \quad L_1 \cap B_c^\bullet = L_2 \cap B_c^\bullet . \qquad (37)$$

2. Consider the following relation on **L**:

$$L' \prec L \quad \text{iff} \quad L = \mathcal{B}/\mathcal{B}', \text{ and } (L' \cap B_c) \subseteq \mathcal{B}' . \qquad (38)$$

Then relation $\prec$ is compatible with the equivalence relation $\asymp$, and the quotient space $\mathbf{L}_{\asymp} = (\mathbf{L}_{/\asymp}, \prec)$ is a partial order. We denote by $\rightarrow$ the transitive reduction of $\prec$, hence we can also regard $\mathbf{L}_{\asymp} = (\mathbf{L}_{/\asymp}, \rightarrow)$ as a DAG. In the sequel, we will freely identify layer L with its equivalence class in $\mathbf{L}_{\asymp}$.

3. Layers can be generally infinite, they are finite for free choice Petri nets, however [4].

We are now ready to extend lattice nets to Petri nets. Consider a labelled Petri net $\mathcal{P} = \{P, T, \rightarrow, M_0, \lambda\}$, where the labelling map takes its values in the alphabet A of alarms, and let $\mathcal{U}_{\mathcal{P}} = \{B, E, \rightarrow, \varphi\}$ be the unfolding of $\mathcal{P}$. We use the notations of (3). Consider the following equivalence relation on the set $\mathbf{L}$:

$$\text{for } L_1, L_2 \in \mathbf{L}, \text{ we write } \quad L_1 \quad \sim \quad L_2 \tag{39}$$

if the following two conditions are satisfied—the reader should compare them with the conditions (ℓ_1) and (ℓ_2) at the beginning of this section (see (36) for the definition of symbol "/"):

(L$_1$) There exist layers $L_i' = \mathcal{B}_i / \mathcal{B}_i'$, such that $L_i' \asymp L_i$, $i = 1, 2$, and L_1' and L_2' are isomorphic, when seen as occurrence nets (i.e., directed graphs labelled by places and transitions from Petri net $\mathcal{P}$); in particular, L_1' and L_2' possess identical continuations;

(L$_2$) the predictable stopping times $\mathcal{B}_1'$ and $\mathcal{B}_2'$ possess extensions that are alarm-isomorphic (cf. definition 1 and above), i.e., they explain the same alarm pattern.

By construction, $\sim \subseteq \asymp$, i.e., the equivalence relation $\sim$ is coarser than $\asymp$. The equivalence class of layer L for $\sim$ is denoted by $L_{\sim}$. Denote the quotient $\mathbf{L}/\sim$ by $\boldsymbol{L}_{\sim}$, we make it a labelled directed circuitfree *hypergraph* as follows:

(a) If $L \in \mathbf{L}$, then

$$L_{\sim} \text{ is a vertex of } \boldsymbol{L}_{\sim}, \text{ and its label is } \quad \psi(L_{\sim}) \overset{\Delta}{=} L_\circ, \tag{40}$$

where $L_\circ$ denotes the equivalence class of layer L modulo an isomorphism of labelled occurrence nets.

(b) For $L \in \mathbf{L}$, denote by ${}^\bullet L$ the preset of L in the DAG $(\mathbf{L}_{\asymp}, \rightarrow)$. The set of equivalence classes of L' for $\sim$, for L' ranging over ${}^\bullet L$, is denoted by $({}^\bullet L)_{\sim}$. We create a *hyperbranch*

$$({}^\bullet L)_{\sim} \rightarrow L_{\sim} \tag{41}$$

and label it with the set

$$\psi\left(({}^\bullet L)_{\sim} \rightarrow L_{\sim}\right) \overset{\Delta}{=} \left\{ (L' \cap L)_\circ \ : \ L' \in {}^\bullet L_{\sim} \right\}, \tag{42}$$

where we regard $(L' \cap L)_\circ$ as a subnet of $L_\circ'$.

Definition 2 *The labelled directed hypergraph* $\mathcal{L}_{\mathcal{P}} \triangleq (\boldsymbol{L}_{\sim}, \rightarrow, \psi)$ *defined in (40,41,42) is called the* lattice net *of the Petri net* $\mathcal{P}$.

Since $\mathcal{L}_{\mathcal{P}}$ is a coding of the unfolding $\mathcal{U}_{\mathcal{P}}$, we can use it to code the set of all configurations of $\mathcal{U}_{\mathcal{P}}$. The unfolding $\mathcal{U}_{\mathcal{P}}$ can be recovered from $\mathcal{L}_{\mathcal{P}}$ by using the following inductive algorithm:

Algorithm 1 Let $\mathcal{L}$ be a prefix of $\mathcal{L}_{\mathcal{P}}$ and $\mathcal{U}$ the corresponding prefix of $\mathcal{U}_{\mathcal{P}}$.

1. Choose $L_{\sim} \in \min(\mathcal{L}_{\mathcal{P}} \setminus \mathcal{L})$.

2. For each hyperbranch $({}^{\bullet}L)_{\sim} \rightarrow L_{\sim}$, construct $X \subset \mathcal{U}$ such that $X_{\circ} = \psi\left(({}^{\bullet}L)_{\sim} \rightarrow L_{\sim}\right)$.

3. Continue $\mathcal{U}$ by gluing $\psi(L_{\sim})$ at X.

This yields an extension of $\mathcal{U}$ by the set of layers belonging to equivalence class $L_{\sim}$, we denote it by $\mathcal{U} \bullet L_{\sim}$, it corresponds to the extension of $\mathcal{L}$ by $L_{\sim}$ in $\mathcal{L}_{\mathcal{P}}$.

5.2 Diagnosis lattices

We are now ready to use lattice nets for diagnosis. We are given an alarm labelled Petri net $\mathcal{P} = \{P, T, \rightarrow, M_0, \lambda\}$, and an associated alarm pattern $\mathcal{A}$. From theorem 1 we need to consider the diagnosis net $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$, and, from the analysis of the preceding section, we can represent it by its lattice net $\mathcal{L}_{\mathcal{P} \times \mathcal{A}}$ instead. Thus, $\mathcal{L}_{\mathcal{P} \times \mathcal{A}}$ is an adequate representation of **diag**$(\mathcal{A})$, we call it a *diagnosis lattice*. Diagnosis lattices provide an alternative data structure for asynchronous diagnosis. This data structure is linear and not branching any more, it is therefore more effective than unfoldings.

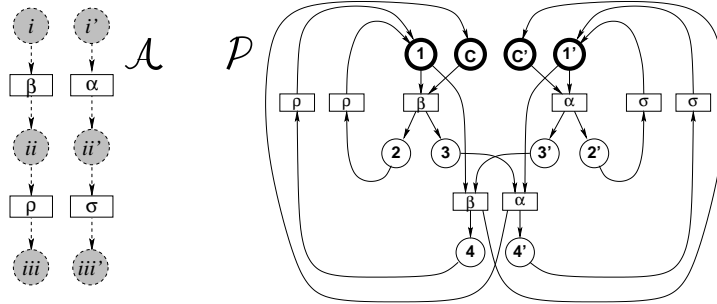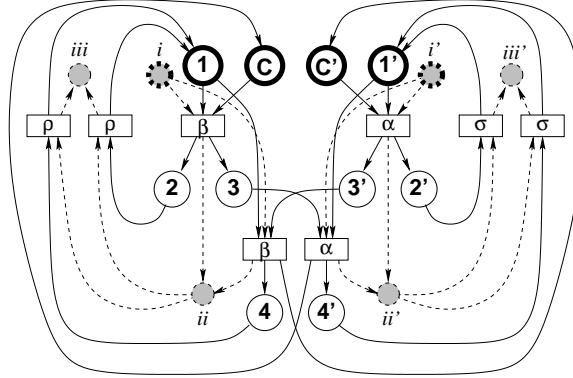**Example 3, continued.** Fig. 12 shows a partially ordered alarm pattern $\mathcal{A}$ for the Petri

Figure 12: Example 3, and a partially ordered alarm pattern for it.

net $\mathcal{P}$ of example 3. It consists of two concurrent alarm sequences $\mathcal{A} = \{\beta \rightarrow \rho, \alpha \rightarrow \sigma\}$, respectively recorded by two local sensors. Then, we show again the net $\mathcal{P}$, we need to compute the product $\mathcal{A} \times \mathcal{P}$, and then its unfolding.

Figure 13: Showing the product $\mathcal{A} \times \mathcal{P}$.

The product $\mathcal{A} \times \mathcal{P}$ is shown in Fig. 13. In this figure, the branches and conditions that originate from the alarm pattern $\mathcal{A}$ are dashed. Also, we have omitted the branches $ii \rightarrow \sigma \rightarrow iii$ and $ii' \rightarrow \rho \rightarrow iii'$, for readability purposes. A branching process $\mathcal{B}$ of the unfolding $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$ is shown in Fig. 14. Again, the branches and conditions that originate from
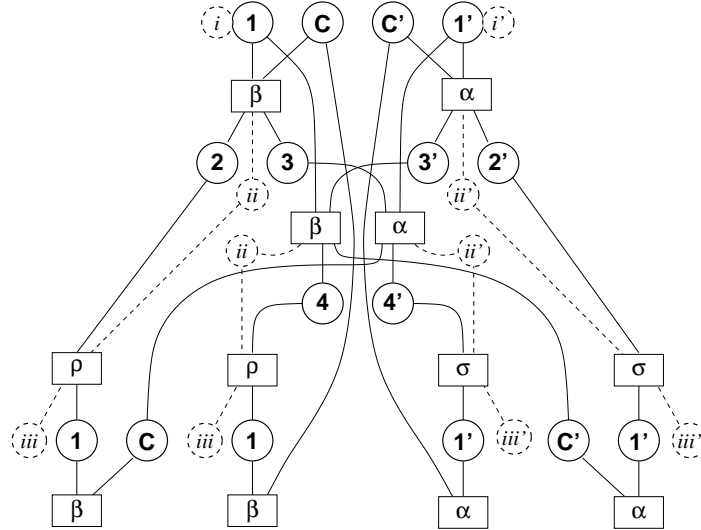
Figure 14: Example 3, the unfolding $\mathcal{U}_{\mathcal{P} \times \mathcal{A}}$.

the alarm pattern $\mathcal{A}$ are dashed. We have glued together the minimal condition labelled by 1 (resp. $1'$) with the condition labelled by i (resp. i'), since they possess identical postsets.

Finally, in Fig. 15-left, we show again this branching process $\mathcal{B}$, in which we have removed
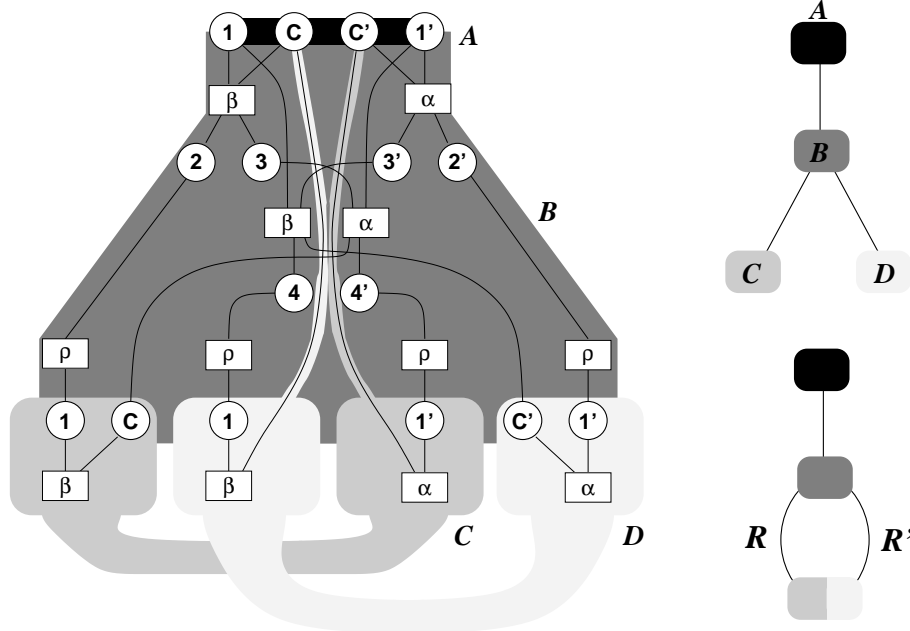
Figure 15: Example 3, diagnosis lattice $\mathcal{L}_{\mathcal{P} \times \mathcal{A}}$.

the dashed part, for the sake of readability. $\mathcal{B}$ is a predictable stopping time, and the grey regions called $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$ are the four layers it contains. Layer $\boldsymbol{A}$ is the set of minimal conditions; the reason is that places $1, 1'$ are branching. The next layer, $\boldsymbol{B}$, continues $\boldsymbol{A}$; hence, by condition (ii) for the definition of predictable stopping times, it must stop at the next conditions labelled by $1, 1'$.

In Fig. 15-top-right, we show the DAG $\mathbf{L}_{\preceq}$. Its vertices are the layers $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$, and its branches reflect the relative position of the corresponding regions, on $\mathcal{B}$. Note that the two layers $\boldsymbol{C}, \boldsymbol{D}$ satisfy $\boldsymbol{C} \sim \boldsymbol{D}$, hence they can be merged when constructing the diagnosis lattice.

In Fig. 15-bottom-right, we show the resulting diagnosis lattice $\mathcal{L}_{\mathcal{P} \times \mathcal{A}}$. The label $\boldsymbol{R}$ is composed of the four conditions labelled by $1, C, C', 1'$ in layer $\boldsymbol{C}$ (shown in mid grey), and the label $\boldsymbol{R'}$ is composed of the four conditions labelled by $1, C, C', 1'$ in layer $\boldsymbol{D}$ (shown in light grey).

In the above example, each layer terminates at a set of cuts—corresponding to a set of states of the marking graph of $\mathcal{P}$. However, in general, for more complex Petri nets, layers terminate at sets of co-sets, not cuts. Therefore, layers provide a finer grain than states of the marking graph, for merging.

6 Discussion

A net unfolding approach to on-line asynchronous diagnosis was presented. This true concurrency approach is suited to distributed and asynchronous systems in which no global state and no global time is available, and therefore a partial order model of time is considered. In the present paper, our basic tool was the net *unfolding,* a branching structure representing the set of configurations of a Petri net, with asynchronous semantics, local states, and partially ordered time. Diagnosis nets were introduced as a way to encode all solutions of a diagnosis problem. They avoid the wellknown state explosion problem, that typically results from having concurrent components in a distributed system interacting asynchronously. Whereas state explosion is kept under control, the computing cost of performing the diagnosis on-line increases (due to the need to compute co-sets); but this is typically a preferred tradeoff for the diagnosis of complex asynchronous systems involving significant concurrency. We have also proposed *lattice nets,* as an improvement of unfoldings, in which both branching and joining conditions are allowed, but no circuit.

It is worth saying what this paper does *not* consider. We do not follow a diagnoser approach. One can view a diagnoser as a "compiled" algorithm for diagnosis. It consists in pre-computing a finite state machine which accepts alarm events, and has states labeled by, e.g., visited faults. In contrast, our approach can be seen as an "interpreted" one, since our diagnosis nets are computed, on-line, by using only the original Petri net structure. Also, we did not investigate issues of diagnosability. Diagnosers for unbounded asynchronous diagnosis and related diagnosability issues have not been considered in the literature, at least to our knowledge. We believe this could be performed by using so-called *complete prefixes* of the unfolding, see [14][15].

Complexity issues have not been addressed. However, the following pragmatic argument can be given to justify the use of unfoldings. Complete prefixes of unfoldings have been used for model checking, an area in which practical complexity is of paramount importance [27][13][14][15].

Various extensions of this work are under progress. The algorithms developed in this paper return all explanations as a diagnosis. Our target application—fault management in telecommunications networks—typically exhibits a great deal of ambiguity. Hence it is of interest to return (the) most likely explanation(s). Probabilistic versions of the present work have been developed for this purpose, see [1][2][18], and [4] for a theory of corresponding stochastic processes.

Since our algorithm is interpreted, not compiled, it can be extended to asynchronous systems subject to *dynamic* changes in their structure—this is typically a situation encountered in network management. This feature favors the use of diagnosis nets instead of the pre-computed diagnosers.

Then, this study is clearly an intermediate step toward *distributed* diagnosis, in which diagnosis is perfomed jointly by a network of supervisors communicating asynchronously. Papers [16][17][18] are a first attempt toward distributed diagnosis. These references concentrate on how to compute the set of configurations $\mathbf{diag}(\mathcal{A})$ in a distributed way, but they do not consider the issue of how to represent $\mathbf{diag}(\mathcal{A})$ efficiently via unfoldings. The techniques

of the present paper therefore should be combined with those of [16][17][18] for getting an efficient solution for the distributed case, this topic will be the subject of a forthcoming paper.

The robustness of algorithms against alarm losses, or more generally the failure to communicate, needs to be investigated. Also, due to the systems complexity, there is little hope indeed, that an exact model can be provided, hence we need to develop diagnosis methods that work based on an incomplete model, i.e., a model not able to explain all observed behaviours.

Last but not least, getting the system model itself is a bottleneck, for complex distributed systems such as, e.g., telecommunications network management systems. The issue of how to partially automatize the model construction is investigated in [26].

References

[1] A. Aghasaryan, E. Fabre, A. Benveniste, R. Boubour, C. Jard. A Petri net approach to fault detection and diagnosis in distributed systems. Part II: extending Viterbi algorithm and HMM techniques to Petri nets. *CDC'97 Proceedings,* San Diego, December 1997.

[2] A. Aghasaryan, E. Fabre, A. Benveniste, R. Boubour, C. Jard. Fault Detection and Diagnosis in Distributed Systems : an Approach by Partially Stochastic Petri nets, *Discrete Event Dynamic Systems: theory and application,* special issue on Hybrid Systems, vol. 8, pp. 203-231, June 98.

[3] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella. Diagnosis of large active systems. *Artificial Intelligence* 110: 135-183, 1999.

[4] A. Benveniste, E. Fabre, and S. Haar. "Markov nets: probabilistic models for distributed and concurrent systems". Irisa Research Report 1415, September 2001, submitted for publication, http://www.irisa.fr/sigma2/benveniste/pub/BlGFH2000.html

[5] R. Boubour, C. Jard, A. Aghasaryan, E. Fabre, A. Benveniste. A Petri net approach to fault detection and diagnosis in distributed systems. Part I: application to telecommunication networks, motivations and modeling. *CDC'97 Proceedings,* San Diego, December 1997.

[6] A.T. Bouloutas, G. Hart, and M. Schwartz. Two extensions of the Viterbi algorithm. *IEEE Trans. on Information Theory,* 37(2):430–436, March 1991.

[7] A.T. Bouloutas, S. Calo, and A. Finkel. Alarm correlation and fault identification in communication networks. *IEEE Trans. on Communications,* 42(2/3/4), 1994.

[8] C. Cassandras and S. Lafortune. *Introduction to discrete event systems.* Kluwer Academic Publishers, 1999.

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

[10] R. Debouk, S. Lafortune, and D. Teneketzis. On the effect of communication delays in failure diagnosis of decentralized discrete event systems. Control group report CGR00-04, Univ. of Michigan at Ann Arbor, submitted for publication, 2001.

[11] J. Desel, and J. Esparza. *Free Choice Petri Nets.* Cambridge University Press, 1995.

[12] J. Engelfriet. *Branching Processes of Petri Nets.* Acta Informatica 28, 1991, pp 575–591.

[13] J. Esparza. Model Checking Using Net Unfoldings. *Sci. of Comp. Prog.* **23**:151–195, 1994.

[14] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In T. Margaria and B. Steffen Eds., *Proc. of TACACS'96,* LNCS 1055, 87-106, 1996. Extended version to appear in *Formal Methods in System Design,* 2000.

[15] J. Esparza, and S. Römer. An unfolding algorithm for synchronous products of transition systems, in *proceedings of CONCUR'99,* LNCS 1664, Springer Verlag, 1999.

[16] E. Fabre, A. Benveniste, C. Jard, L. Ricker, and M. Smith. Distributed state reconstruction for discrete event systems. Proc. of the *2000 IEEE Control and Decision Conference (CDC'2000),* Sydney, Dec. 2000.

[17] E. Fabre, A. Benveniste, C. Jard. Distributed diagnosis for large discrete event dynamic systems. In *Proc of the IFAC congress,* Jul. 2002.

[18] E. Fabre. Distributed diagnosis for large discrete event dynamic systems, *in preparation.*

[19] R.G. Gardner, and D. Harle. Methods and systems for alarm correlation. In *GlobeCom 96,* London, November 1996.

[20] C.N. Hadjicostis, and G.C. Verghese. Monitoring discrete event systems using Petri net embeddings. in *Proc. of Application and theory of Petri nets 1999,* 188–208.

[21] A. Giua. —PN state estimators based on event observation. *Proc. 36th Int. Conf. on Decision and Control,* San Diego, Ca, 4-86–4091, 1997.

[22] A. Giua, and C. Seatzu. Observability of Place/Transition Nets. Preprint, 2001.

[23] K.X. He and M.D. Lemmon. Liveness verification of discrete-event systems modeled by *n*-safe Petri nets. in *Proc. of the 21st Int. Conf. on Application and Theory of Petri Nets,* Danmark, June 2000.

[24] K.X. He and M.D. Lemmon. On the existence of liveness-enforcing supervisory policies of discrete-event systems modeled by *n*-safe Petri nets. in *Proc. of IFAC'2000 Conf. on Control Systems Design,* special session on Petri nets, Slovakia, June 2000.

[25] I. Katsela, A.T. Bouloutas, and S. Calo. Centralized vs distributed fault localisation. *Integrated Network Management IV,* A.S. Sethi, Y. Raynaud, and F. Faure-Vincent, Eds. Chapman and Hall, 251-261, 1995.

[26] MAGDA project. Paper on modeling, to be available on the MAGDA project web page http://magda.elibel.tm.fr/

[27] K. McMillan. Using Unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: *4th Workshop on Computer Aided Verification*, pp. 164–174, 1992.

[28] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures, and domains. Part I. *Theoretical Computer Science* **13**:85–108, 1981.

[29] W. Reisig. *Petri nets.* Springer Verlag, 1985.

[30] L. Rozé and M-O. Cordier. Diagnosing discrete event systems: extending the diagnoser approach to deal with telecommunications networks. *Discrete Event Dynamic Systems: theory and application,* special issue on WODES'98, 12(1), 43–82, Jan. 2002.

[31] G. Rozenberg and J. Engelfriet. Elementary Net Systems. In: *Lectures on Petri Nets I: Basic Models.* LNCS **1491**, pp. 12–121, Springer, 1998.

[32] A. Sahraoui, H. Atabakhche, M. Courvoisier, and R. Valette. Joining Petri nets and knowledge-based systems for monitoring purposes. *Proc. of the IEEE Int. Conf. on Robotics Automation,* 1160–1165, 1987.

[33] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Trans. Autom. Control* 40(9), 1555-1575, 1995.

[34] R. Sengupta. Diagnosis and communications in distributed systems. In *Proc. of* WODES *1998, international Workshop On Discrete Event Systems,* 144-151, IEE, London, England, 1998.

[35] S. Tripakis. Undecidable problems of decentralized observation and control. In *Proc. of the 40th IEEE Conf. on Decision and Control,* Orlando, Dec. 2001.

[36] G. Winskel. Event structures. In *Advances in Petri nets,* LNCS vol. 255, 325–392, Springer Verlag, 1987.