Diagnosis of supervision patterns on bounded labeled Petri nets by Model Checking

Yannick Pencolé, Audine Subias

To cite this version:

Yannick Pencolé, Audine Subias. Diagnosis of supervision patterns on bounded labeled Petri nets by Model Checking. 28th International Workshop on Principles of Diagnosis (DX 2017), Sep 2017, Brescia, Italy. 9p. hal-01827437

HAL Id: hal-01827437
https://laas.hal.science/hal-01827437
Submitted on 2 Jul 2018

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

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Diagnosis of supervision patterns on bounded labeled Petri nets by Model Checking

Yannick Pencolé and Audine Subias
LAAS-CNRS, Université de Toulouse, CNRS, INSA, Toulouse, France.
e-mail: (ypencole@laas.fr, subias@laas.fr)

Abstract
This paper investigates the problem of pattern diagnosis of systems modeled as bounded labeled Petri nets that extends the diagnosis problem on single fault events to more complex behaviours. An effective method to solve the diagnosis problem is proposed. It relies on a matching relation between the system and the pattern that turns the pattern diagnosis problem into a model-checking problem.

1 Introduction
Fault diagnosis of Discrete Event Systems (DES) is a generic problem that has been widely studied [1]. The original definition has been introduced in [2] in the automaton formalism. The fault diagnosis problem was based on a system model and on an explicit set of fault events. From this early approach several contributions were proposed exploring different modeling formalisms among them, communicating automata ([3], [4], [5]) or Petri nets ([6], [7], [8], [9], [10]) leading to various diagnosis methods. This original problem has also been extended to consider more complex faulty behaviours than simple fault events. In [11] the authors consider as faulty, behaviours that violate a given formal specification such as reaching a dead locking state, and propose to use linear-time temporal logic (LTL) to express these complex behaviours. In [12] supervision patterns are introduced by means of labelled transition systems to define complex behaviours one may be interested in to diagnose. Supervision patterns as occurrence of one fault, occurrence of multiple faults, ordered occurrence of events faults but also multiple occurrence of a fault are considered by the authors. In [13], such patterns are called diagnosis conditions and are formally represented with LTL properties with past operators. The advantage of such approaches is their generalizing power to tackle a large spectrum of diagnosis problems. Moreover, the use of patterns emphasizes on the separation between the behavioral model of the system and the objectives of the diagnosis tasks defined as patterns [14]. Another advantage is the possibility of pattern reusability to address others diagnosis problems on others systems.

In this article we revisit the diagnosis of patterns in the framework of Petri nets. We propose a concise way to model patterns that stresses pattern advantages. The diagnosis problem is posed as a pattern matching problem, so that the proposed approach does not derive a diagnoser-like machine like in [12] but a machine that captures the matching between the system and a pattern. The construction of this machine is based on a specific product of Petri nets called the system-pattern product. The diagnosis problem of a supervision pattern for a given sequence of observations is then solved by determining the intersection between the system’s evolutions that produce the input sequence and those that match the pattern. What we want to emphasize in this paper is how this task can be totally and efficiently entrusted to a model checker fed by Petri nets. An interesting point is that the proposed approach can be fully coupled with a pattern diagnosability analysis of systems [15].

The Petri net formalism used in the approach and the system and patterns modeling are presented in Section 2. Section 3 states the diagnosis problem into a pattern matching problem. Section 4 focuses on the construction of the Petri machine that capture the matching. The implementation of the diagnosis problem as a model checking problem is given Section 5. We present experimental results in Section 6. Finally, conclusions and some perspectives are given Section 7.

2 Modeling framework
Patterns aim at modeling complex faulty behaviours or any normal behaviour of interest. This section presents the modeling framework of patterns and systems.

To have a complete characterization of the diagnosis problem, we use Labeled Petri nets with transition priorities ([16]). This formalism extends the classical Labeled Petri net. The use of priorities is a convenient way to manage transition conflicts in a model and then to enhance the expressiveness of the model. In particular, transition priorities will be used here to overcome the problem of modeling the interactions between the system and a pattern. This section first presents the formalism, called L-type Labeled Prioritized Petri Nets, that is used throughout this paper. Then the modeling of the system and the pattern are presented.

2.1 L-type Labeled Prioritized Petri Nets
Definition 1 (LLPPN). An L-type Labeled Prioritized Petri Net (LLPPN for short) is a 7-uple \( N = (P, T, A, >, \Sigma, Q) \) where:
- \( P \) is a finite set of nodes called places;
- \( T \) is a finite set of nodes called transitions and \( P \cap T = \emptyset \);
- \( A \subseteq (P \times T) \cup (T \times P) \) is a binary relation transitive, not reflexive and not symetrical [17].
• $\Sigma$ is a finite set of transition labels (events);
• $\ell : T \rightarrow \Sigma$ is the transition labeling function;
• $Q$ is the set of final markings.

The state of an LLPPN is defined as a marking. A marking $M$ is a function $M : P \rightarrow N$ which maps any place of the net to a number of tokens contained in this place.

A marked LLPPN is a couple $(N, M_0)$ also denoted as the 8-tuple $(P, T, A, \succ, \ell, \Sigma, Q, M_0)$. The preset $\text{pre}(n)$ of a node $n$ is the set of nodes $\text{pre}(n) = \{n' \in P \cup \mathcal{T} : (n', n) \in A\}$ and the postset $\text{post}(n)$ of a node $n$ is the set of nodes $\text{post}(n) = \{n' \in P \cup \mathcal{T} : (n, n') \in A\}$. By construction the preset and postset of a transition (respectively a place) are two sets of places (respectively two sets of transitions).

An LLPPN is a classical Petri Net formalism with some additional constraints or pieces of information.

Static priorities between transitions are introduced (relation $\succ$) to manage transition conflicts. Priorities add expressiveness to the Petri net and modify the model’s semantics. Indeed a transition cannot be fired if a transition with higher priority is fireable at the same time: a net transition $t$ is fireable from a given marking $M$ if and only if $(\forall p \in \text{pre}(t), M(p) > 0) \land (\forall p' \in \text{post}(t), M(p') = 0)$. This firing rule ensures that any firable transition $t$ is fireable in the usual sense (when $\succ = \emptyset$) but it avoids the choice of firing transitions with a lower priority that are also fireable in the usual sense.

Firing a transition $t$ consumes a token from any place in the preset and adds a token in any place of the postset. From a given marking $M_0$, the fire of a transition $t$ leads to the new marking $M' = M \setminus \text{pre}(t) \cup \text{post}(t)$, which is denoted $M \xrightarrow{t} M'$. A marking $M$ is reachable in a marked LLPPN if it exists a sequence of firable transitions

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots \xrightarrow{t_k} M,$$

where $r$ is the sequence $t_1 \ldots t_k$, the sequence $r$ is called a run of the net. For a given LLPPN $N$ and a given marking $M$, the set of reachable markings from $M$ is denoted: $R(N, M)$.

An LLPPN also contains a transition labeling function $\ell$ that maps any transition to a label of a set $\Sigma$, this is the way to associate physical events with transitions. Last but not least, an LLPPN has a set of final markings $Q$. The set $Q$ is the way to define the language that is associated with an LLPPN (L-language of [18]).

Let $r = t_1 \ldots t_k \in T^*$ and let $\ell(r)$ denote the sequence $\ell(t_1) \ldots \ell(t_k) \in \Sigma^*$, the language generated by a given LLPPN is defined as follows:

**Definition 2** (Language of an LLPPN). The language generated by an LLPPN $N = \langle P, T, A, \succ, \ell, \Sigma, Q, M_0 \rangle$ is:

$$L(N) = \{\ell(r) : r \in T^* \land M_0 \xrightarrow{r} M \land M \in Q\}.$$  

### 2.2 System model

We propose in this paper to model a system over the set of events $\Sigma$ as an LLPPN $\Theta = \langle P, T, A, \succ, \ell, \Sigma, Q, M_0 \rangle$.

As introduced in [2], systems generate prefix-closed regular languages: for any run $r$ of the system, any of its prefix $r' : r = r'r''$ is indeed a run. It follows that such a system can be represented by a bounded LLPPN (regular language) and with a set of final markings corresponding to the set of reachable markings $Q = R(\Theta, M_0)$ (prefix-closed language). The set of events $\Sigma$ generated by the system is partitioned into two subsets: $\Sigma_o$ is the set of observable events and $\Sigma_u$ the set of unobservable ones.

**Definition 3** (System model). The model of a system is an LLPPN $\Theta = \langle P, T, A, \succ, \ell, \Sigma, Q, M_0 \rangle$ such that:

- $Q$ is the set of reachable markings $R(\Theta, M_0)$;
- the Petri net is bounded $(3n \in \mathbb{N}^+ : \forall M \in R(\Theta, M_0), \forall p \in P, M(p) \leq n)$.

![Figure 1: System example $\Theta_1$ with $\Sigma_o = \{c, d\}$](image)

Figure 1 presents an example of such a model $\Theta$. Bold events $c$ and $d$ are the observable events of the system whereas the events $\{a, b, c, f\} = \Sigma_u$ are the unobservable events. The dashed line represents the priority between the transitions of event $b$ and $a$. It ensures that the transition labeled with $b$ is always fired before the one labeled with $a$ in a marking $M$ such that $M(p_0) \geq 1 \land M(p_1) \geq 1$ (as the initial marking of the system).

### 2.3 Pattern model

The notion of pattern requires the definition of a general and flexible framework allowing to design a large spectrum of diagnosis problems including simple faulty event, multiple faults, fault repetitions...and more generally any behaviour of interest (faulty or not). The use of fault pattern helps to clearly separate the system and the diagnosis objectives (see [14]). The reusability is then reinforced to tackle new problems. For complex patterns this separation principle improves the applicability. Indeed, a modification of the system model in order to merge with the pattern is a tricky task that might even be impracticable in real cases. This would suppose to search in the system model for all the traces where the pattern has occurred and to modify them accordingly to the pattern model. Moreover, the system model would have to be modified each time a new pattern has to be analyzed.

**Definition 4** (Pattern). A pattern $\Omega$ is an LLPPN

$$\Omega = \langle P, T, A, \succ, \ell, \Sigma, Q, M_0 \rangle$$

such that:

1. $M_0 \notin Q$ i.e. the language of the pattern does not contain $\varepsilon$;
2. from any reachable marking $M$ of $R(\Omega, M_0)$ there exists a sequence of transitions $r \in T^*$ such that $M \xrightarrow{r} M'$ and $M' \in Q$;
3. from any reachable marking $M$ of $R(\Omega, M_0)$ there is no event $e \in \Sigma$ that labels more than one firable transition;
It is managed by a specific product operator that is defined by a set of final markings.

Definition 5 relies on the definition of sequence matching.

Definition 6 (Sequence matching). A sequence $\rho \in \Sigma^*$ matches another sequence $\sigma \in \Sigma^*$, denoted $\rho \models \sigma$, if:

1. $\sigma$ is empty ($\sigma = \epsilon$); or
2. $\sigma = s_1 \cdots s_n \in \Sigma^*$ and there exist two sequences $\rho_0, \rho_1 \in \Sigma^*$ such that:
   - $\rho = \rho_0 \rho_1$;
   - $\rho_1 \models \sigma_1$.

A sequence $\rho$ matches a sequence $\sigma$ if $\rho$ contains any event of the sequence $\sigma$ and these events occur in the same order as in $\sigma$. Obviously $\rho$ can contain several collections of events that gather the events of $\sigma$ in the same order; in this case $\sigma$ occurs several times in $\rho$. The pattern $\rho$ matches $\sigma$ if $\sigma$ occurs at least once. Figure 5 illustrates how the sequence $\rho = bcabcabc$ matches the sequence $\sigma = acc$. Definition 6 is recursively applied three times till $\sigma_3 = \epsilon$. In this example, $\sigma$ occurs twice in $\rho$. Back to the example of Figure 1, the system $\Theta_1$ matches both the patterns $\Omega_{a \rightarrow e}$ and $\Omega_{a \rightarrow b}$ of Figure 2. The sequence $bade$ is a possible run of $\Theta_1$ and $bade$ $\models$ $ae$. $\Theta_1$ also matches $\Omega_{a \rightarrow b}$ as the sequence $badeebd$ is a possible run of $\Theta_1$ and $badeebd$ $\models$ $ab$.

Based on the notion of pattern, the classical fault diagnosis problem on DESs can be generalized as proposed in [12]. Let $\mathcal{P}_{\Sigma_1 \rightarrow \Sigma_2} : \Sigma_1^* \rightarrow \Sigma_2^*$ denote the projection function of any sequence of $\Sigma_1$ to a sequence of $\Sigma_2$ defined by:

$$
\mathcal{P}_{\Sigma_1 \rightarrow \Sigma_2}(\epsilon) = \epsilon,
\mathcal{P}_{\Sigma_1 \rightarrow \Sigma_2}(\rho.s) = \mathcal{P}_{\Sigma_1 \rightarrow \Sigma_2}(\rho) \text{ if } \rho \in \Sigma_1^*, s \not\in \Sigma_2,
\mathcal{P}_{\Sigma_1 \rightarrow \Sigma_2}(\rho.s) = \mathcal{P}_{\Sigma_1 \rightarrow \Sigma_2}(\rho),s \text{ if } \rho \in \Sigma_1^*, s \in \Sigma_2.
$$

For a given pattern $\Omega$, the diagnosis problem consists in defining an $\Omega$-diagnoser function that takes as input a sequence of observable events and that produces one of the three symbols $\{\Omega \models \text{certain}, \Omega \models \text{safe}, \Omega \models \text{ambiguous}\}$ ([201]).

Definition 7 (\(\Omega\)-diagnoser). Let $\Theta$ be the model of a system based on the set of events $\Sigma = \Sigma_u \cup \Sigma_o$, an $\Omega$-diagnoser is a function

$$
\Delta_\Omega : \Sigma_1^* \rightarrow \{\Omega \models \text{certain}, \Omega \models \text{safe}, \Omega \models \text{ambiguous}\}
$$

such that:

- $\Delta_\Omega(\sigma) = \Omega \models \text{certain}$ if for any run $\rho \in \mathcal{L}(\Theta)$ that is consistent with $\sigma$ (i.e. $\mathcal{P}_{\Sigma \rightarrow \Sigma_o}(\rho) = \sigma$), $\rho \models \Omega$.
called the system-pattern product and is denoted: \( \Theta \times \Omega \). Informally speaking, \( \Theta \times \Omega \) consists of the union of the places of \( \Theta \) and \( \Omega \). The set of transitions is composed of the set of transitions of \( \Theta \) and a set of synchronized and prioritized transitions replacing the transitions from \( \Omega \).

**Definition 8** (system-pattern product). Let 
\[ \Theta = (P_\Theta, T_\Theta, A_\Theta, \cdot_\Theta, (\cdot_\Theta, \triangleleft_\Theta, Q_\Theta, M_\Theta)) \] 
and 
\[ \Omega = (P_\Omega, T_\Omega, A_\Omega, \cdot_\Omega, (\cdot_\Omega, \triangleleft_\Omega, Q_\Omega, M_\Omega)) \] 
be respectively a system and a pattern such that \( P_\Theta \cap P_\Omega = \emptyset \) and \( T_\Theta \cap T_\Omega = \emptyset \) and the system-pattern product \( \Theta \times \Omega \) is the LLPPN \( \langle P, T, A, \succ, \ell, \Sigma, Q, M_0 \rangle \) defined as follows.

- \( P = P_\Theta \cup P_\Omega \).
- \( T = T_\Theta \cup T_\Omega \) such that
  - \( T_s = \bigcup_{t_\Theta \in T_\Theta} \{ t_\Theta \mid t_\Theta \in T_\Theta \wedge \exists t_\Omega \in T_\Omega \wedge t_\Theta(t_\Theta) = t_\Omega(t_\Omega) = \ell \} \) is the set of synchronized transitions, the transition denoted \( t_\Theta \) results from the synchronization of \( t_\Theta \in T_\Theta \) and \( t_\Omega \in T_\Omega \).
- \( A = A_\Theta \cup A_\Omega \) such that
  - \( A_s = \{ (t_\Theta, t_\Omega) : t_\Theta = t_\Omega \mid t_\Theta \in T_\Theta \} \) \( \cup \bigcup_{t_\Theta \in T_\Theta} \{(t_\Theta, t_\Omega) : t_\Theta = t_\Omega \mid t_\Theta \in T_\Theta \} \) \( \cup \{(t_\Theta, t_\Omega) : t_\Theta = t_\Omega \mid t_\Theta \in T_\Theta \} \) \( \cup \{(t_\Theta, t_\Omega) : t_\Theta = t_\Omega \mid t_\Theta \in T_\Theta \} \) \( \cup \{(t_\Theta, t_\Omega) : t_\Theta = t_\Omega \mid t_\Theta \in T_\Theta \} \).
- \( \succ = \succ_\Theta \cup \succ_\Omega \), with:
  - \( \succ_\Theta = \succ_\Theta \cup \succ_\Omega \), such that:
  - \( \forall \ell \in T_\Theta, t \in T_\Theta \), \( \ell(t) = t_\Theta(t_\Theta) \), \( \ell(t) = t_\Omega(t_\Omega) \), \( \ell(t) = t_\Theta(t_\Theta) \).
  - \( \Sigma = \Sigma_\Theta \cup \Sigma_\Omega \),
  - \( Q = \{ M_\Theta \cup M_\Omega : M_\Theta \in Q_\Theta \wedge M_\Omega \in Q_\Omega \} \),
  - \( M_0 = M_\Theta \cup M_\Omega \).

Intuitively, the pattern \( \Omega \) is applied as a filter on the system \( \Theta \) by using the product \( \Theta \times \Omega \). Any transition of label \( e \) in the system is synchronized with a transition of the pattern \( \Omega \) labeled with the same event. The product is asymmetric in the sense that \( \Theta \times \Omega \) contains all the transitions of \( \Theta \) whereas the transitions of \( \Omega \) with such an event \( e \) have been replaced by a set of synchronized transitions with \( \Theta \). Given the current marking of \( \Theta \times \Omega \) and the occurrence of a new event \( e \), two cases can happen. If the current marking of \( \Theta \times \Omega \) enables a synchronized transition labeled with \( e \), as it is prioritized, it will be triggered (the event \( e \) is therefore part of the pattern), otherwise only a transition from the system \( \Theta \) can be triggered in \( \Theta \times \Omega \) (the system keeps running by producing the event \( e \) but \( e \) is not part of the pattern).

With the help of this product, it is possible to identify in the system \( \Theta \) the set of runs where the pattern \( \Omega \) has occurred. The following results assert this intuition.

Let \( \Theta \times \Omega \) and \( M_{\Theta \times \Omega} \) be the initial marking of \( \Theta \times \Omega \). Let \( M_{\Theta \Theta} \) (resp. \( M_{\Theta \Omega} \)) be the initial marking of \( \Theta \) (resp. \( \Omega \)).

**Theorem 1.** Let \( \Theta \) be the LLPPN of a system over the alphabet \( \Sigma \) and \( \Omega \) be the LLPPN of a pattern:

\[ \mathcal{L}(\Theta \times \Omega) = \{ \rho \in \mathcal{L}(\Theta) : \rho \ni \Omega \}. \]
Theorem 1 is the fundamental property of the operator \( \land \). The set of runs that are characterized by the language \( \mathcal{L}(\Theta \times \Omega) \) are exactly the set of runs that match the pattern \( \Omega \) [15].

5 Implementation by Model-Checking

Through the Petri model of the system-pattern product \( (\Theta \times \Omega) \), we have a model that determines for each possible run of the system if this evolution matches or not the pattern. This model can then be exploited to solve the diagnosis problem.

For an observable input sequence generated by the system \( (\sigma) \), the pattern diagnosis problem can be solved by determining the intersection between the system’s evolutions that generate this input sequence and those that match the pattern. This intersection can be characterized through the synchronized product between the Petri net model of the pattern. This intersection can be characterized through the synchronized product between the Petri net model of the system-pattern product and the Petri net model \( O \) of the observed input sequence \( (\sigma) \) produced by the system \( : \Theta||O \) [21].

Considering \( \sigma = e_1, e_2, \ldots e_k \), the Petri net model \( O \) is an \( LLPPN \) defined by: \( O = (P_O, T_O, A_O, \rightarrow_O, e, \Sigma_O, Q_O, M_{O0}) \) with:

- \( P_O = \bigcup_{i=1}^{k+1} \{ p_i \} \) and \( T_O = \bigcup_{i=1}^{k} \{ t_i \} \);
- \( A_O = \bigcup_{i=1}^{k} \{ (p_i, t_i), (t_i, p_{i+1}) \} \);
- \( \rightarrow_O = \emptyset \);
- \( \ell(t_i) = e_i, \forall t_i \in T \) and \( i \in 1..k \);
- \( \Sigma_O = \{ e_1, \ldots e_k \} \);
- \( Q_O = \{ \{ p_{k+1} \} \} \);
- \( M_{O0} = \{ p_1 \} \).

Once the synchronized product \( \Theta||O \) built, it is then possible to use model checking techniques to determine the results produced by the \( \Omega \)-diagnoser. Here we give a complete description of the implementation by Model-Checking.

5.1 Model checking problem

We propose here to implement the \( \Omega \)-diagnoser function by using the TINA toolkit of [22] and [17] that provides all the necessary tools to directly generate the Kripke structure on which the model-checking problem is defined [23]. TINA actually generates enriched labeled Kripke structures from the analyzed Petri nets to check properties written in SELLT (State/Event Linear Temporal Logic) which extends LTL in the way presented here above.

A formula \( \psi \) is a SE-LTL formula if it is a universally quantified formula

\[ \psi ::= \forall \varphi \]

such that

\[ \varphi ::= r | \neg \varphi | \varphi \lor \varphi | \varphi \land \varphi | (| r \| \varphi | \varphi | \varphi | \varphi | \varphi \Upsilon \varphi \]

\[ r ::= e | e \ominus e \]

\[ e ::= p \mid a \mid c | e \nabla e \]

with \( p \) a place symbol, \( a \) a transition symbol, \( e \in \mathbb{N}, \Delta \in \{=, <, >, \leq, \geq\} \) and \( \nabla \in \{+, -, \ast, \}/ \). The operators \( \bigcirc \) (next), \( \square \) (always), \( \bigtriangleup \) (eventually) and \( \Upsilon \) (until) have their usual LTL semantics.

The \( \Omega \)-diagnoser is based on two questions written in SELLT concerning the Petri net model \( \Theta||O \):

1. Question \( \varphi_{CERTAIN} \): Are we sure that the pattern is recognized in each system’s behaviour that produces the observations? If the result is false the following question is asked to the model checker:

2. Question \( \varphi_{SAFE} \): Are we sure that the pattern is never recognized in each system’s behaviour that produces the observations?

More formally, each question corresponds to a SE-LTL formula. Given \( M \) a marking of \( O || \Theta_O \), \( M_O \) denotes the restriction of \( M \) to \( \Omega \) i.e. the marking of the pattern’s places.

In a similar way \( M_O \) denotes the restriction of \( M \) to \( O \).

1. \( \varphi_{CERTAIN} \equiv \square((M_O \in Q_O) \Rightarrow (M_Q \in Q_Q)) \), in others words, is it always true (\( \square \)) that if the system generates \( (M_O \in Q_Q) \) then \( (\Rightarrow) \) it matches the pattern \( (M_Q \in Q_Q) \).

2. \( \varphi_{SAFE} \equiv \square((M_O \in Q_Q) \Rightarrow (M_Q \not\in Q_Q)) \), in others words, is it always true (\( \square \)) that if the system generates \( (M_O \in Q_Q) \) then \( (\Rightarrow) \) it does not match the pattern \( (M_Q \not\in Q_Q) \).

The complete and automated implementation of the method to solve the pattern diagnosis problem on a system \( \Theta \) is:

1. Compute \( \Theta||O = \Theta \times \Omega \).
2. Compute \( O \) the LLPPN that represents the sequence of observations \( \sigma \).
3. Compute the synchronisation of \( \Theta_O \) and \( O: \Theta||O \).
4. Compute \( \varphi_{CERTAIN} \) from \( \Theta||O \) and check \( \varphi_{CERTAIN} \) on the Kripke structure of \( \Theta||O \).
5. If the result is false, then compute \( \varphi_{SAFE} \) from \( \Theta||O \) and check \( \varphi_{SAFE} \) on the Kripke structure of \( \Theta||O \).
6. \( \Delta_O(\sigma) \) returns \( \Omega_{\ldots CERTAIN} \) if \( \varphi_{CERTAIN} \) is true, else it returns \( \Omega_{\ldots SAFE} \) if \( \varphi_{SAFE} \), otherwise it returns \( \Omega_{\ldots ambiguous} \).

6 Case study

6.1 Description of the system

The case study that we propose in this paper is a product transportation system. It is a two-level system composed of two product sites (namely sites 1 and 2) at level 2 where a set of products is stored and two assembly stations at level 1 that request products from level 2. A lift is used between the two levels. Each site has a conveyor belt to move a product from the site to the lift and each station also has a conveyor belt to get the products from the lift. Figure 6 presents the system model. Once a product is detected on the site 1, a Product1 signal (noted \( P_{r1} \)) is emitted. The product is then put in a box and becomes available. The box is then pushed and sent into the lift (action \( PushBox \) that starts with event \( P_1 \) and ends with event \( EP_{1} \)). Site 2 behaves in a similar manner. A product from site 1 has a priority access to the lift. In the Petri net model of Figure 6, this priority relation is indicated by a dashed edge between the transitions labeled
by $P_1$ and $P_2$. After a product has been pushed into the lift, the lift goes down (action $\text{Down}$) to reach level 1 (signal $D$ detects the end of the $\text{Down}$ action). At this level the box is directed to the station that makes the request (either request $\text{Req}_1$ or request $\text{Req}_2$). If it is request $\text{Req}_1$, then the box is pushed on the conveyor belt of station 1 (action $\text{PushReq}_1$ that starts with event $\text{Req}_1$ and ends with event $\text{ERReq}_1$). Then the conveyor belt performs a move-left action ($\text{LeftReq}_1$, $\text{ELReq}_1$) to deliver the box with the product and a move-right action to go back to initial position ($\text{RightReq}_1$, $\text{ERRreq}_1$). The behaviour of station 2 is similar except that its conveyor belt moves right first ($\text{RightReq}_2$, $\text{ERRreq}_2$) and then moves left ($\text{LeftReq}_2$, $\text{ELReq}_2$). Once the box has been pushed on a conveyor belt, the lift goes up (action $\text{Up}$ ending with the emission of signal $U$) to level 2.

### 6.2 Monitoring of a single event class

We first present some experiments with a single event pattern $\Omega_1$ as defined in Figure 3 with $k = 1$ and $e = \text{Req}_1$. The aim of this first example is to show how the proposed framework is able to diagnose the occurrence of single events as the classical diagnosis methods do. The pattern $\Omega_1$ represents the occurrence of the event $\text{Req}_1$ in the system. Table 1 shows the obtained results for the following scenario:

$$\sigma = \text{Pr1}.\text{Pr2}.\text{D}.\text{U}.\text{D}.\text{ELReq}_2.\text{ERReq}_2.\text{ELReq}_1.\text{ERRreq}_1.\text{U}$$

In order to show the evolution of the diagnosis, any line of Table 1 represents one step of the scenario $\sigma$ (i.e. line 1 is $\text{Pr1}$, line 2 is $\text{Pr1}.\text{Pr2}$, etc.). The second column shows the number of states in the corresponding Kripke structure and the third column shows the number of transitions. The first two lines show a $\Omega_1 - \text{safe}$ diagnosis. Diagnoses between lines 3 and 7 are $\Omega_1 - \text{ambiguous}$. Finally, lines 8-10 show an $\Omega_1 - \text{certain}$ diagnosis. These results can be intuitively explained as follows: $\text{Req}_2$ cannot occur before the lift goes down (lines 1-2) and as long as $\text{ELReq}_1$ is not observed there is an ambiguity about the occurrence of $\text{Req}_1$.

In classical diagnosis methods, events are sometimes partitioned into a set of fault classes. Here, also it is possible to represent this type of event class. With a pattern $\Omega_2$ as represented in Figure 4, it is possible to represent one occurrence of event $e_1 = \text{Req}_1$ or one occurrence of event $e_2 = \text{Req}_2$ (two transitions $t_1$ and $t_2$ only), i.e. an event of class $\{\text{Req}_1, \text{Req}_2\}$. In this example, the set of accepting markings is composed of the markings $\{M_1, M_2\}$ where $M_1(p_{21}) = 1$ and for any other place $p$ $M_1(p) = 0$ and $M_2(p_{22}) = 1$ and for any other place $p$ $M_2(p) = 0$. In this example, the $\varphi_{\text{certain}}$ question can have the following form: let $p_{\text{obs}}$ denote the last place of the sequence of observations.

$$\varphi_2 = \square((p_{\text{obs}} = 1) \Rightarrow (p_{21} = 1 \lor p_{22} = 1)).$$

Table 2 shows the diagnosis results (left columns) on the same scenario as above for $\Omega_2$. When the first $D$ is observed one of the $\text{Req}$ may silently happen afterwards but not necessary hence the ambiguity. But as soon as we observe $U$, $\Omega_2$ has certainly occurred.

### 6.3 Monitoring of the occurrence of multiple events

Another interesting pattern has the same structure as the one of Figure 4, it is the occurrence of both events $\text{Req}_1$ and $\text{Req}_2$ where, as explained in Figure 4, the only accepting marking is $M(p_{21}) = 1$ and $M(p_{22}) = 1$ and $M(p) = 0$ for the other two places $p$. In this case the $\varphi_{\text{certain}}$ question becomes:

$$\varphi_3 = \square((p_{\text{obs}} = 1) \Rightarrow (p_{21} = 1 \land p_{22} = 1)).$$

This type of pattern is interesting if we want to supervise that a set of $n$ specific types of event have occurred. Table 2 shows the diagnosis results (right columns) on the same scenario as above for this pattern. The diagnosis becomes certain (line 8) as soon as we both observe $\text{ELReq}_2$ and $\text{ELReq}_1$.

### 6.4 Monitoring of a complex behaviour

The previous pattern examples show that the presented framework can model classical diagnosis problems where we look for the occurrence of a set of single events. However the purpose of this framework is to allow for the diagnosis of more complex behaviours. Back to our example, the global function of the system is to transfer items from conveyors at Level 2 to conveyors at Level 1. Figure 6 actually models the behaviour of the system but no single faulty event is present (any event that can happen is normal).

Figure 7 presents a complex behaviour that one might want to diagnose within the system of Figure 6. This pattern $\Omega_3(m, n)$ models the consecutive occurrence of $m = 3$ events of type $\text{Req}_1$ that are not interleaved with occurrence of $\text{Req}_2$ events (the left conveyor keeps having its requests fulfilled while the right conveyor get stuck) and this pattern should occur $n = 2$ times (the pattern has a counter). This pattern is made up of normal events but defines a behaviour.
Figure 6: Case study: the product transportation system

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Size</th>
<th>nb(Req₁)</th>
<th>counter</th>
<th>nS(Γ)</th>
<th>nT(Γ)</th>
<th>Time (ms)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₁</td>
<td>20</td>
<td>5</td>
<td>1</td>
<td>92</td>
<td>107</td>
<td>11</td>
<td>safe</td>
</tr>
<tr>
<td>S₂</td>
<td>200</td>
<td>5</td>
<td>1</td>
<td>1028</td>
<td>1223</td>
<td>11</td>
<td>safe</td>
</tr>
<tr>
<td>S₃</td>
<td>2000</td>
<td>5</td>
<td>1</td>
<td>10388</td>
<td>12383</td>
<td>235</td>
<td>safe</td>
</tr>
<tr>
<td>S₄</td>
<td>5000</td>
<td>5</td>
<td>1</td>
<td>25988</td>
<td>30983</td>
<td>1105</td>
<td>safe</td>
</tr>
<tr>
<td>S₅</td>
<td>2000</td>
<td>2</td>
<td>10</td>
<td>107534</td>
<td>128379</td>
<td>1993</td>
<td>ambiguous</td>
</tr>
<tr>
<td>S₆</td>
<td>2000</td>
<td>5</td>
<td>10</td>
<td>10388</td>
<td>12383</td>
<td>228</td>
<td>safe</td>
</tr>
<tr>
<td>S₇</td>
<td>2000</td>
<td>10</td>
<td>10</td>
<td>10388</td>
<td>12383</td>
<td>239</td>
<td>safe</td>
</tr>
<tr>
<td>S₈</td>
<td>2000</td>
<td>2</td>
<td>10</td>
<td>7201</td>
<td>8400</td>
<td>156</td>
<td>certain</td>
</tr>
<tr>
<td>S₉</td>
<td>3500</td>
<td>10</td>
<td>10</td>
<td>17009</td>
<td>20266</td>
<td>577</td>
<td>certain</td>
</tr>
<tr>
<td>S₁₀</td>
<td>1000</td>
<td>5</td>
<td>2</td>
<td>907987</td>
<td>2155290</td>
<td>15745</td>
<td>certain</td>
</tr>
<tr>
<td>S₁₁</td>
<td>1000</td>
<td>5</td>
<td>10</td>
<td>1201381</td>
<td>2877901</td>
<td>21018</td>
<td>certain</td>
</tr>
<tr>
<td>S₁₂</td>
<td>2000</td>
<td>10</td>
<td>8</td>
<td>2737559</td>
<td>6482932</td>
<td>65066</td>
<td>certain</td>
</tr>
<tr>
<td>S₁₃</td>
<td>2000</td>
<td>20</td>
<td>8</td>
<td>5999626</td>
<td>14245740</td>
<td>244379</td>
<td>certain</td>
</tr>
<tr>
<td>S₁₄</td>
<td>1750</td>
<td>20</td>
<td>8</td>
<td>4812356</td>
<td>11444801</td>
<td>148629</td>
<td>safe</td>
</tr>
</tbody>
</table>

Table 3: Diagnosis results on the pattern Ω₃.
that could be unexpected (both conveyors at Level 1 should get items regularly).

Table 3 presents some experimental results for a set of scenarios $S_1 \ldots S_{14}$ where the occurrence of the pattern $\Omega_5(m, n)$ is diagnosed. For each scenario, Table 3 shows the size of the observable sequence (column 2), the number $m$ of $\text{Req}$ events in the pattern $\Omega_5(m, n)$ (column 3) behaviours as well as the number $n$ of occurrences (counter). The size of the involved Kripke structure as a number of states ($\alpha S(\Gamma)$) and transitions ($\alpha T(\Gamma)$) for a given scenario is also given. The time here represents the computation time of all the diagnosis process (the computation of the Kripke structure and the verification of the questions). Scenarios $S_1 \ldots S_4$ diagnose the occurrence of $\Omega_5(5, 1)$ based on a certain amount of $\sigma$ cycles (see Section 6.2). The sequence $\sigma$ being composed of 10 events, $S_1$ diagnoses 2 cycles of $\sigma$. $S_2$ diagnoses 10 of them, etc. Based on $\sigma$, 5 consecutive occurrences of $\text{Req}_1$ can never happen that is why the result is always safe.

Scenarios $S_5 \ldots S_7$ diagnoses the occurrence of $\Omega_5(m, 10)$, $m = 2, 5, 10$ so here we attempt to diagnose that $m$ consecutive occurrences of $\text{Req}_1$ happen 10 times. For $m = 5, 10$, the result is safe for the same reason as above. For $n = 2$, there is an ambiguity. Indeed, consider two consecutive cycles of $\sigma$, once $\text{ELReq}$ is observed in the first cycle, it is possible that one $\text{Req}$ has occurred just before $\text{ELReq}_1$. Within the second cycle, before observing $\text{ELReq}_2$, one second occurrence of $\text{Req}_1$ may have happened (followed by one occurrence of $\text{Req}_2$), so it is possible (but not certain) that two consecutive occurrences of $\text{Req}_1$ have occurred and, as there are 200 cycles, this can happen 10 times, hence the ambiguous result. Scenario $S_5$ is similar to $S_2$ but the observable sequence is a set of cycles $\sigma'$ where any $\text{ELReq}$, $\text{ERR}$ in $\sigma$ have been replaced by $\text{ELReq}_1$, $\text{ERR}_1$, the diagnosis is therefore certain. Scenario $S_6$ presents a scenario composed of 100 cycles of $\sigma$ followed by 75 cycles of $\sigma'$ followed by 100 cycles of $\sigma$ followed by 75 cycles of $\sigma'$ (i.e. 350 cycles): 10 consecutive occurrences of $\text{Req}_1$ happen more than 10 times in $S_6$. The last five scenarios $S_{10} \ldots S_{14}$ rely on the fact that the initial state of the system is unknown (unknown states of the conveyors and the lift). These scenarios illustrate the fact that the method can be applied on an observation window [24] that does not contain the first observations of the system.

7 Conclusion

This paper addresses the problem of patterns diagnosis by a generic framework for pattern modeling and model checking techniques. The diagnosis method is designed in the framework of Petri nets (L-type Labeled Prioritized Petri Nets) and it can be viewed as a pattern matching based method. The construction of a Petri net machine capturing the matching of a pattern by the system is done by means of a specific Petri net product. Then a translation of the diagnosis problem into a model-checking problem allows an efficient resolution that is entrusted to the Petri net analyser TINA. The paper reports the results of experimenting this pattern diagnosis approach on an illustrative example. Future work could consolidate some aspects of the method. In particular, the expressivity of patterns that currently generate regular language may be extended to context-free-language. Another research direction is to extend the work to time systems. We currently investigate the use of Time Petri Nets to diagnose more interesting behaviours.

References


[10] Maria Paola Cabasino, Alessandro Giua, and Carla Seatzu. Diagnosability of discrete-event systems using


