Diagnosability and Predictability of pattern in Labelled Petri Nets

Eric Lubat, Camille Coquand, Yannick Pencole, Audine Subias

Diagnosability and Predictability of pattern in Labelled Petri Nets

Éric Lubat[1], **Camille Coquand**[1], **Yannick Pencolé**[2] and **Audine Subias**[1]
[1]CNRS, LAAS, Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France
(e-mail: firstname.lastname@laas.fr)
[2]CNRS, LAAS, Univ de Toulouse, LAAS, F-31400 Toulouse, France
(e-mail: yannick.pencole@laas.fr)

Abstract

This paper addresses the problem of checking predictability of event patterns in labelled Petri nets. After formally introducing the predictability problem of an event pattern, a method for automatically checking predictability is proposed. The proposed method has two steps. The first one consists in checking diagnosability of event patterns which is a necessary condition for predictability. And, if diagnosability holds, the second step is launched and concludes about the predictability of the investigated event pattern. The proposed method uses a model-checking approach and is fully implemented with the help of a model-checking toolchain.

1 Introduction

The problems of fault event diagnosis and diagnosability in Discrete Event Systems (DES) have been widely addressed with various formalisms (e.g automata, Petri nets, statecharts) and several extensions to consider time and probabilistic aspects or different decisional structures [1]. Informally speaking, diagnosability of DES aims to determine the possibility of concluding with certainty about the occurrences of fault events based on a finite number of observations. Even if the diagnosability property is essential to assist a diagnoser design, it does not allow to avoid unsafe situations. As a consequence, the predictability property, that states if faults can be predicted with certainty before their occurrence, has also been investigated [2]. Note that predictability is the problem of prognosis for discrete event systems but the two term are sometimes mingled. As for the diagnosability, extensions have been developed to consider a large spectrum of problems. In [3] for instance the authors explore the problem on labeled Petri nets and use the concept of *critical pairs* to conclude on the predictability of a fault. [4; 5; 6] mainly focuses on decentralized solution. The work of [7] considers the predictability of more complex faulty situations. This predictability property is directly linked to the diagnosability one. Indeed, if a failure is to be predicted it needs to be a diagnosable failure (since you cannot predict future behaviors which are not diagnosable). An overview on Diagnosability and Prognosability is given in [8].

The objective of this article is to extend the sequence patterns predictability problem of [7] based on automata, to the Petri net framework. Formal Petri Net based definitions of diagnosability and predictability of patterns are then proposed. Moreover, a fully implemented method that both checks diagnosability and predictability is proposed. This method relies on a model-checking formulation of the two properties that are solved by an efficient model-checking tool chain. The method is illustrated and experimental results on several examples are provided.

The remainder of the paper is as follows: in Section 2, we define the mathematical terminology and notions used throughout the paper, especially regarding the Petri net model and its semantics. Section 3 details the problem statement about checking both diagnosability and predictability of patterns in Label Petri Nets. The method to check both properties is then presented in Section 4. Section 5 presents the tool chain that is used, describes the implementation of the methods and lastly presents experimental results. Finally conclusion and perpectives are given in Section 6.

2 Formal background on Petri Nets

In this section the modeling formalism used in this paper is presented. First the Labeled Petri Nets are presented, then an abstraction of Petri Nets called Labeled Transition System is defined.

2.1 Petri nets

A Petri Net (PN) [9] is a formalism for modeling discrete event systems using tokens to indicate the current state of a system.

Definition 1 (Petri Nets)**.** *A PN is a tuple* $\langle P, T, \mathbf{Pre}, \mathbf{Post}\rangle$ *where* P *and* T *are the set of places and transitions;* $\mathbf{Pre} : P \times T \to \mathbb{N}$ *and* $\mathbf{Post} : T \times P \to \mathbb{N}$ *are the precondition and postcondition functions.*

The current state of the system is denoted as a *marking* m that describes the set of places where the tokens currently lie:

Definition 2 (Marking)**.** *A marking* m *is a (total) function* $m : P \to \mathbb{N}$ *from places in* P *to natural numbers.*

A *marked Petri net* is a tuple $N = \langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0\rangle$ where m_0 is the initial marking of the net. To fire a transition t of the net, the condition is that t must be *enabled* in the current marking m, that is $\forall p \in P, m(p) \geq \mathbf{Pre}(p,t)$. The effective firing of transition t from a marking m leads to the marking m' such that $\forall p \in P, m'(p) = m(p) - \mathbf{Pre}(p,t) + \mathbf{Post}(p,t)$. The fire of t from m to m' is denoted $m \xrightarrow{t} m'$. A marking m is reachable if $m = m_0$ or there exists a sequence of transitions $\langle t_1, \ldots, t_n\rangle$ such

that $m_0 \xrightarrow{t_1} \dots \xrightarrow{t_n} m$. In the following work, a Petri net will always be considered as marked (initial marking m_0) and bounded (i.e. every reachable marking m is such that $\exists k \in \mathbb{N}, \forall p \in P, m(p) \leq k$).

Finally to end this sub-section, an extension of Petri Nets that associates with every transition t a label $\mathcal{L}(t)$ from a finite alphabet Σ is recalled.

Definition 3 (Labeled Petri Net)**.** *A Labeled Petri Net (LPN for short) is a tuple $\langle P, T, \mathbf{Pre}, \mathbf{Post}, \Sigma, \mathcal{L}, m_0\rangle$ where $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0\rangle$ is a marked Petri Net, Σ is an alphabet and $\mathcal{L} : T \to \Sigma$ is a labelling function.*

2.2 Labeled Petri net semantics as Labeled Transition System

The semantics of a Label Petri net can be seen as a Labelled Transition System (LTS), as introduced in [7]. Let $\mathcal{E}(m)$ denote the set of transitions enabled by a marking m, i.e. $\mathcal{E}(m) = \{t \mid t \in T, \forall p \in P, m(p) \geq \mathbf{Pre}(p,t)\}$. A *state* of an LTS is a couple $(m, \mathcal{E}(m))$ where m is a marking and $\mathcal{E}(m)$ is the enabled transitions associated with m. In the LTS formalism, transitions are associated with events (also called *actions*) that label the underlying LPN.

The semantics of a labeled Petri net can be seen as a Labeled Transition System (LTS), as introduced in [7]. Let $\mathcal{E}(m)$ denote the set of transitions enabled by a marking m, i.e. $\mathcal{E}(m) = \{t \mid t \in T, \forall p \in P, m(p) \geq \mathbf{Pre}(p,t)\}$. A *state* of an LTS is a couple $(m, \mathcal{E}(m))$ where m is a marking and $\mathcal{E}(m)$ is the enabled transitions associated with m. In the LTS formalism, transitions are associated with events (also called *actions*) that labels the underlying LPN.

Definition 4 (Labeled Transition Systems)**.** *A Labeled Transition System (LTS) over the set of actions A is a tuple $[\![N]\!] = \langle S, s_0, A, \to\rangle$ where S is the set of states, $s_0 \in S$ is the initial state, $\to \subseteq S \times (A \cup \{\varepsilon\}) \times S$ is the set of edges.*

In the following, $(s, \alpha, s') \in \to$ is denoted $s \xrightarrow{\alpha} s'$. The following definition of the semantics of a LPN is quite standard, see for instance [10; 11]. In general terms, the semantics of a PN is a LTS structure $\langle S, s_0, \to\rangle$ with only one possible kind of action: a transition t is fired. A transition t can fire from the marking m if t is enabled. More formally, the semantics of the LPN is defined as:

Definition 5 (LPN semantics)**.** *The semantics of a LPN N $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0\rangle$ with the labelling function $\mathcal{L} : T \to \Sigma \cup \{\varepsilon\}$ is the Labeled Transition System (LTS) $[\![N]\!] = \langle S, s_0, \Sigma, \to\rangle$ where S is the smallest set containing s_0 and closed by $\to$, where:*

- *$s_0 = (m_0, \mathcal{E}(m_0))$ is the initial state, with m_0 the initial marking and $\mathcal{E}(m_0)$ the set of initially enabled transitions;*
- *the state transition relation $\to \subseteq S \times (\Sigma \cup \{\varepsilon\}) \times S$ is the relation such that for all states $(m, \mathcal{E}(m))$ in S:*
  - *$(m, \mathcal{E}(m)) \xrightarrow{\mathcal{L}(t)} (m', \mathcal{E}(m'))$ iff :*
    - *$t \in \mathcal{E}(m)$*
    - *$\forall p \in P, m'(p) = m(p) - \mathbf{Pre}(p,t) + \mathbf{Post}(p,t)$*

Like with nets, the alphabet of a LTS is the set of labels, in Σ, associated with discrete actions. Each firing sequence σ of N (also called a run) is associated with an *execution* ρ that is a sequence in its semantics $[\![N]\!]$. Sequence ρ is an event word over the alphabet containing the labels (in $\Sigma \cup \{\varepsilon\}$).

Consecutive transitions can always be grouped together, meaning that when $(m, \mathcal{E}(m)) \xrightarrow{\mathcal{L}(t)} (m', \mathcal{E}(m'))$ and $(m', \mathcal{E}(m')) \xrightarrow{\mathcal{L}(t')} (m'', \mathcal{E}(m''))$ then necessarily $(m, \mathcal{E}(m)) \xrightarrow{\mathcal{L}(t).\mathcal{L}(t')} (m'', \mathcal{E}(m''))$. By contrast, a *trace* is the word obtained from an execution when only the discrete actions without ε are kept. Then the *language* of a LPN N, denoted $L(N)$, is the set of all its (finite) traces.
For the sake of clarity, the labelling function is extended to $\mathcal{L} : T^* \to (\Sigma \cup \{\varepsilon\})^*$ as $\mathcal{L}(t_0 \dots t_n) = \mathcal{L}(t_0).\mathcal{L}(t_1) \dots \mathcal{L}(t_n)$. From a state s, a run σ is said to be acceptable (denoted $s \xrightarrow{\mathcal{L}(\sigma)}$) if there exists a state s' such that $s \xrightarrow{\mathcal{L}(\sigma)} s'$.

By definition, the language of a LPN is prefix-closed; and it is regular when the net is bounded [12].

2.3 Products

To end the section about the formal background, the product of LPN and LTS that will be used throughout this paper is finally presented.

Definition 6 (Product of LPN)**.** *Let $\{N_i = \langle P_i, T_i, \mathbf{Pre}_i, \mathbf{Post}_i, \Sigma_i, \mathcal{L}_i, m_0^i\rangle\}_{i\in\{1,2\}}$ be two LPNs. Let L be an alphabet. The product $N_1 \times_L N_2 = \langle P_1 \cup P_2, T_{12}, \mathbf{Pre}_{12}, \mathbf{Post}_{12}, \Sigma_1 \cup \Sigma_2, \mathcal{L}_{12}, m_0^{12}\rangle$ is such that: for $i \in \{1,2\}$*

- *$\forall p \in P_i, m_0^{12}(p) = m_0^i(p)$;*
- *$\forall t \in T_{12}$,*
  1. *$t \in T_i \Rightarrow \mathcal{L}_{12}(t) = \mathcal{L}_i(t) \notin L, \forall p \in P_i, \mathbf{Pre}_{12}(p,t) = \mathbf{Pre}_i(p,t), \mathbf{Post}_{12}(p,t) = \mathbf{Post}_i(p,t), \forall p \in P_j, j \neq i, \mathbf{Pre}_{12}(p,t) = 0, \mathbf{Post}_{12}(p,t) = 0$;*
  2. *$t = (t_1, t_2), t_1 \in T_1, t_2 \in T_2 \Rightarrow (\mathcal{L}_1(t_1) = \mathcal{L}_2(t_2) = \mathcal{L}_{12}(t) \in L) \wedge (\forall p \in P_1 \cup P_2, \mathbf{Pre}_1(p,t_1) > 0 \Rightarrow \mathbf{Pre}_2(p,t_2) = 0) \wedge (\forall p \in P_1, \mathbf{Pre}_{12}(p,t) = \mathbf{Pre}_1(p,t_1), \mathbf{Post}_{12}(p,t) = \mathbf{Post}_1(p,t_1), \forall p \in P_2, \mathbf{Pre}_{12}(p,t) = \mathbf{Pre}_2(p,t_2), \mathbf{Post}_{12}(p,t) = \mathbf{Post}_2(p,t_2))$.*

This product is also used as an equivalence to the LTS product and is weak-time bisimilar (see [13]). The synchronous product of two LTS is defined as:

Definition 7 (Product of LTS)**.** *Assume $[\![N_1]\!] = \langle S_1, s_1^0, \Sigma_1, \to_1\rangle$ and $[\![N_2]\!] = \langle S_2, s_2^0, \Sigma_2, \to_2\rangle$ are two LTS. The product of $[\![N_1]\!]$ by $[\![N_2]\!]$ is the LTS $[\![N_1]\!] \| [\![N_2]\!] = \langle S_1 \times S_2, (s_1^0, s_2^0), \Sigma, \to\rangle$ with $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\to$ the smallest relation obeying the following rules ($\alpha \in \Sigma_1 \cup \Sigma_2 \cup \{\epsilon\}$):*

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \ \alpha \in (\Sigma_1 \setminus \Sigma_2) \cup \{\varepsilon\}}{(s_1, s_2) \xrightarrow{\alpha} (s_1', s_2)}$$

$$\frac{s_2 \xrightarrow{\alpha}_2 s_2' \ \alpha \in (\Sigma_2 \setminus \Sigma_1) \cup \{\varepsilon\}}{(s_1, s_2) \xrightarrow{\alpha} (s_1, s_2')}$$

$$\frac{s_1 \xrightarrow{\alpha}_1 s_1' \ s_2 \xrightarrow{\alpha}_2 s_2' \ \alpha \neq \varepsilon}{(s_1, s_2) \xrightarrow{\alpha} (s_1', s_2')}$$

The set of reachable states is such that $\Delta_{[\![N]\!]}(s,\sigma) \stackrel{\text{def}}{=} \{s' \in S | s \xrightarrow{\sigma} s'\}$.

3 Problem statement

This paper addresses the problem of monitoring of the occurrence of specific but unobservable event patterns in discrete event systems. Such patterns can represent, among others, critical situations, unexpected situations, faults. By analysing the model of the system, we aim at formally checking whether the system produces enough observable information to decide with certainty whether a pattern has definitely occurred (which would mean that the pattern is diagnosable) or will definitely occur (which would mean that the pattern is predictable). This section formally describes both the diagnosability and the predictability problem. Note that, as it will be explained later, checking diagnosability is required before checking predictability.

3.1 Modelling

A system is modeled as a bounded LPN N. The alphabet of such LPN, Σ, is partitionned in three differents sets:

- Σ_o the set of observable events,
- Σ_u the set of unobservable events,
- $\{\varepsilon\}$ the empty sequence, (i.e. the event associated with the transition is not significant)

All along this paper, the usual assumptions that the system has no deadlock and that the observability of the system is live are made, i.e. for any execution ρ ending with an unobservable event, there exists $n \in \mathbb{N}$ such that for all continuations ρ' such that $\rho.\rho' \in L(N)$ and $|\rho'| = n$, ρ' has at least one observable event.

To illustrate this type of model, the example from the paper of Jéron and Lafortune [7] (called *Base* all along this paper) has been translated into a LPN model (see Figure 1). In this example the observable labels are $\Sigma_o = \{b, c\}$ and the unobservable labels are $\Sigma_u = \{a, f1, f2\}$.

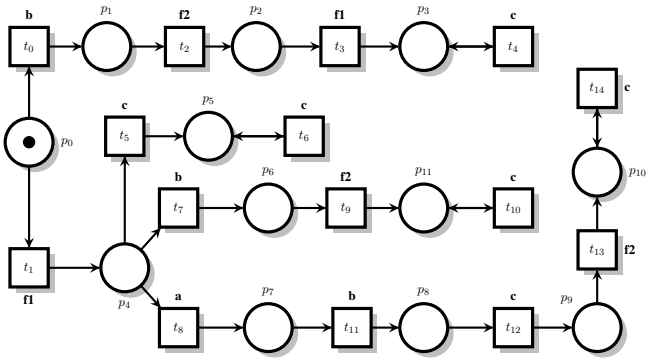

Figure 1: LPN N of the *Base* System

Event patterns, initially introduced as *supervision patterns* in [14], model complex but unobservable behaviours of interest. As opposed to the patterns of [14] that are modeled with automata, the proposed patterns are modeled as LPNs.

Definition 8 (Pattern)**.** *A pattern* $M = \langle P_M, T_M, \mathbf{Pre}_M, \mathbf{Post}_M, F, \mathcal{L}_M, m_0^M \rangle$ *is a safe LPN such that:*

- *the alphabet* F *is a subset of* Σ_u*;*
- *there exists a place* $found \in P_M$ *such that the marking* m_{found} *with* $\forall p \in P_M \setminus \{found\}, m_{found}(p) = 0 \wedge m_{found}(found) = 1$ *is reachable;*
- *only marking* m_{found} *is reachable after the fire of a transition from marking* m_{found}*.*

The marking m_{found} is distinguished as a witness for detection. From the initial marking m_0^M, as soon as the current marking is m_{found}, it means that the run σ produced by the pattern is a possible behaviour of interest produced by the pattern. In the following $L_{found}(M)$ denotes the sub-language of $L(M)$ that contains the set of executions $\rho = \mathcal{L}(\sigma)$ of M associated with runs σ that lead to the final marking m_{found}.

In this paper, the pattern are considered to be *well-formed*, which consists in adding three more conditions.

Definition 9 (Well-formed Pattern)**.** *A well-formed pattern is defined as followed:*

1. *Patterns are* total*: they should always allow transitions on the labels in* F*, at any time (they never block or prioritize a transition).*
2. *Patterns are* deterministic*: the same labels should lead to the same states.*
3. *Labels in* F *are unobservable:* $F \cap \Sigma_o = \emptyset$*.*

For instance, consider the following well-formed Pattern M in Figure 2. This pattern represents the occurrence of one event $f1$ followed by an event f_2 or one event f_2 followed by an event f_1, so basically M represents any behaviour of the system where two events f_1 and f_2 must occur at least once whatever their occurrence order: $L_{found}(M) = \{(f_1 f_1^* f_2 + f_2 f_2^* f_1)(f_1 + f_2)^*\}$. As opposed to the automata representing patterns in [14] that must represent any event from the system's alphabet Σ, here the description of the pattern is succint and only requires to model the events of interests (i.e. f_1 and f_2).

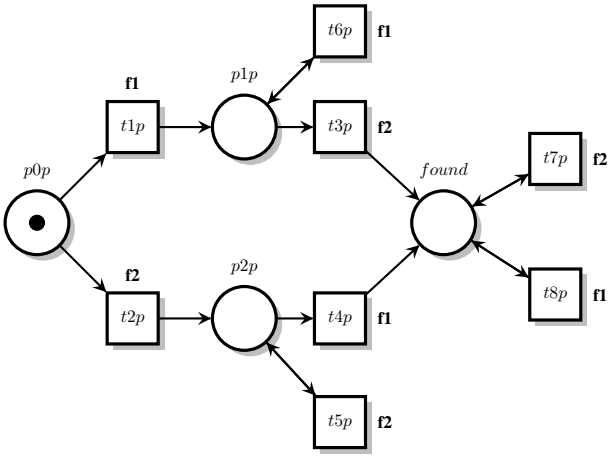

Figure 2: Pattern M

Definition 10 (Pattern matching)**.** *An execution* $\rho \in L(N)$ *matches a pattern* M *(denoted* $\rho \ni M$*) if there exists a subword* ρ' *of* ρ *such that* $\rho' \in L_{found}(M)$*.*

An execution of the system that matches the pattern contains as a subword the occurrence of a behaviour of interest that is modelled in the pattern, so in other words the pattern

has occurred in the execution of the system. It is possible to determine the set of executions that match a pattern by applying a product between semantics $[\![N]\!]$ and $[\![M]\!]$, as summarized by the following result which is straightforward.

Proposition 1. *Let N be a system and M be a pattern, the set of executions $\rho \in L(N)$ that match M is:*

$$\mathcal{M}(N, M) = \{\rho = \mathcal{L}(\sigma) | (m_0, \mathcal{E}(m_0)) \xrightarrow{\mathcal{L}(\sigma)} (m, \mathcal{E}(m)) \in [\![N]\!] || [\![M]\!] \wedge m(found) = 1\}.$$

Looking back at the system in Figure 1 and pattern in Figure 2, the execution $\rho_1 = f_1abcf_2cc$ of the system matches the pattern as f_1f_2 is a subword of ρ_1 and $f_1f_2 \in L_{found}(M)$. This execution is part of $[\![N]\!]||[\![M]\!]$ as it is $((m_0, \mathcal{E}(m_0)), (m_0^M, \mathcal{E}(m_0^M))) \xrightarrow{\mathcal{L}(t_1t_8t_{11}t_{12}t_{13}t_{14}t_{14})} ((m, \mathcal{E}(m)), (m_f, \mathcal{E}(m_f))$ with $m_f(found) = 1$. The set of executions of the system that finally match the pattern is $\{((\Sigma \setminus \{f_1\})^* f_1 (\Sigma \setminus \{f_2\})^* f_2 + (\Sigma \setminus \{f_2\})^* f_2 (\Sigma \setminus \{f_1\})^* f_1).\Sigma^*\}$.

3.2 Diagnosability and predictability

This section updates the formal definitions of diagnosability and predictability of patterns with regards to the formal LPN framework presented in this paper. Both properties rely on the notion of observable projection of executions. The observable projection of an execution $\rho = e.\rho' \in L(N)$ is:

- $\mathcal{P}(\rho) = e.\mathcal{P}(\rho')$ if $e \in \Sigma_o$
- $\mathcal{P}(\rho) = \mathcal{P}(\rho')$ if $e \notin \Sigma_o$

A system is M-diagnosable if the occurrence of M in an execution ρ of the system can be always decided with certainty after observing a finite number of events produced in ρ. Formally:

Definition 11. *A system is M-diagnosable if* $\forall \rho_1 \in L(N), \exists n \in \mathbb{N}^*$ *s.t.* $\rho_1 = \rho_1'.\rho_1'', \rho_1' \ni M, ||\rho_1''|| \geq n$, $\forall \rho_2 \in L(N), \mathcal{P}(\rho_1) = \mathcal{P}(\rho_2) \Rightarrow \rho_2 \ni M$.

A system is M-predictable if it is always possible to assert the future occurrence of the pattern M, strictly before its actual occurrence. This prediction is only based on the observable labels of the system. Predictability of patterns is formally defined as follows:

Definition 12. *A system is M- predictable if*

$$\exists n \in \mathbb{N}, \forall \rho_1 \in L(N) \cap \Sigma^*\Sigma_o \text{ with } \rho_1 \ni M,$$

$$\exists (\rho_1', \rho_1''), \rho_1'.\rho_1'' = \rho_1, \rho_1' \in \{\varepsilon\} \cup (L(N) \cap \Sigma^*\Sigma_o), \rho_1' \not\ni M$$

such that:

$$\forall \rho_2 = \rho_2'.\rho_2'' \in L(N), \mathcal{P}(\rho_2') = \mathcal{P}(\rho_1'), ||\mathcal{P}(\rho_2'')|| \geq n \Rightarrow \rho_2 \ni M.$$

Example: Consider again the system *Base* from Figure 1 and the pattern M from Figure 2. The system is M-diagnosable. Any execution of the system that matches M eventually produces the observation bcc and no other execution can produce bcc. By observing bcc, the diagnoser can assert M has occurred with certainty. Note also that the system is M-predictable. As soon as b is observed, a predictor can assert that M will eventually occur with certainty. Finally, the result from [2] that links diagnosability and predictability is recalled:

Proposition 2. *If a system is M-predictable then it is M-diagnosable.*

In other words, this result states that diagnosability is a necessary condition for predictability. Informally speaking, if a system is not M-diagnosable, it means that there exists an infinite set of couples of arbitrarily long executions ρ_1, ρ_2 of the system such that $\mathcal{P}(\rho_1) = \mathcal{P}(\rho_2)$ and $\rho_1 \ni M$ but $\rho_2 \not\ni M$. It follows that there is no prefix of $\mathcal{P}(\rho_1)$ that can be used to decide that M will definitely occur as $\rho_2 \not\ni M$.

The rest of the paper is now devoted to automatically check M-diagnosability and then M-predictability by the use of formal model-checking techniques.

4 Verification of Diagnosability and Predictability

In this article a method to check both diagnosability and predictability of patterns in Labeled Petri Nets is proposed. This method is based on the verification of properties on Kripke structures of products of the LTS of the system N and the pattern M. As presented in the previous section, M-diagnosability is a necessary condition for M-predictability to hold in the system N. The proposed method will first check whether M-diagnosability holds in the system and if the answer is positive it will run the test for M-predictability. Both steps for checking diagnosability and predictability are computationally independent, however running the test for M-predictability is conclusive only if the system is M-diagnosable. In other words, running the test for M-predictability that is proposed in this method is inconclusive if the system is not M-diagnosable. Both tests are implemented as model-checking problems.

4.1 Checking Diagnosability

The method that is used to check diagnosability straightforwardly derives from [15]. Indeed, it has been shown that the diagnosability of a pattern can be checked as the inexistence of *critical pairs* in the twin-plant of the system. In our setting, the twin plant is the LPN $(N_1 \times_F M_1) \times_{\Sigma_o} (N_2 \times_F M_2)$ where N_1, N_2 are identical copies of N and M_1, M_2 are identical copies of M that are synchronised on the observable labels Σ_o. A critical pair is a infinitely long execution of the twin-plant which matches M_1 but not M_2 and then represents a couple of executions from N with the same observations but only one matches the pattern (so it is not diagnosable). Diagnosability is a *global property* that can be checked on the LTS of the twin-plant by checking an LTL property. LTL (Linear Temporal Logic) is a modal temporal logic with modalities referring to time. In LTL, one can encode formulae about the future of state sequences in the Kripke structure. For the diagnosability test, two operators of LTL are mainly used:

- $\Diamond$: *Finally* which means that the properties linked to $\Diamond$ has to finally hold in the sequence.
- $\Rightarrow$: the classical *implication*.

The diagnosability test is then defined as follows:

Theorem 1 (Diagnosability of a Pattern [15])**.** *Given a well-formed pattern M, with labels F, the net N, with observable label Σ_o, is diagnosable for pattern M if and only if all the maximal executions of the product $(N_1 \times_F M_1) \times_{\Sigma_o} (N_2 \times_F M_2)$ satisfy $(\Diamond found.1) \Rightarrow \Diamond (found.2 \vee dead)$.*

Informally speaking, if the previous result holds for a given system N and a pattern M, it means that for every execution of the twin-plant which eventually matches pattern M_1 ($\Diamond found.1$) it will eventually matches the copy pattern M_2 or be a deadlock ($\Diamond(found.2 \vee dead)$), hence the absence of critical pairs.

Our process to check *Diagnosability* can then be decomposed into 3 steps.

Step D1. Construct the synchronous product between the LPN N_1 and its Pattern M_1 with the alphabet F $((N_1 \times_F M_1))$ and duplicate it as $((N_2 \times_F M_2))$

Step D2. Construct the synchronous product and its LTS semantics $[\![(N_1 \times_F M_1) \times_{\Sigma_o} (N_2 \times_F M_2)]\!]$.

Step D3. Use an LTL checker to conclude on the LTL formula $(\Diamond found.1) \Rightarrow \Diamond (found.2 \vee dead)$.

4.2 Checking Predictability

This section presents the predictability test of a pattern M in a system N. Intuitively it follows the same steps as in [7] but is translated as a model-checking problem to be solved effectively. The test first relies on the building of a LTS Π, called a predictor in [7], and secondly on a formal property to check on Π.

Construction of Π

Starting with the system N and the pattern M, the LTS $[\![(N \times_F M)]\!] = \langle S, s_0, \Sigma, \rightarrow \rangle$ is first built and the following partition of states S is considered:

- F_o is the set of states that hold the property $found$.
- I_n are the states which inevitably lead to states in F_o:

$$I_n = \{s \in S \setminus F_o | \exists n \geq 0, \forall \sigma \in \Sigma^*, s \xrightarrow{\sigma} s' \wedge ||\sigma|| \geq n \Rightarrow \Delta_{[\![N \times_F M]\!]}(s, \sigma) \subseteq F_o\}$$

- $N_o = S \setminus (F_o \cup I_n)$.

The computation of Π is a two-step construction based on $[\![(N \times_F M)]\!]$. First, in order to abstract out the unobservable labels of $[\![(N \times_F M)]\!]$, the unobservable closure of $[\![(N \times_F M)]\!]$ is performed.

Definition 13 (Unobservable-Closure of a LTS)**.** *Let $[\![N]\!] = \langle S, s_0, \Sigma, \rightarrow \rangle$ be the LTS of an LPN N and $\Sigma_o \subseteq \Sigma$ be the observable events of N, the Unobservable-Closure of $[\![N]\!]$ is $U_c([\![N]\!]) = \langle S, s_0, \Sigma_o, \rightarrow_{U_c} \rangle$ where for any $s, s' \in S, o \in \Sigma_o, s \xrightarrow{o}_{U_c} s'$ in $U_c([\![N]\!])$ whenever there exists $\rho \in \Sigma_u^*$ such that $s \xrightarrow{\rho.o} s'$ in $[\![N]\!]$.*

The second step consists in building a deterministic LTS.

Definition 14 (Determinization of a LTS)**.** *Let $[\![N]\!] = \langle S, s_0, \Sigma, \rightarrow \rangle$ be a LTS over the LPN N and an alphabet Σ. The determinization of $[\![N]\!]$ is the LTS $Det([\![N]\!]) = \langle X, X_0, \Sigma, \rightarrow_{Det} \rangle$ where $X = 2^S$ (the set of subsets of S called macro-states), $X_0 = \{s_0\}$, and $\rightarrow_{Det} = \{(s^D, \alpha, \Delta_{[\![N]\!]}(X, \alpha)), s^D \in X \wedge \alpha \in \Sigma\}$.*

Finally, the predictor Π is:

$$\Pi = Det(U_c([\![(N \times_F M)]\!])).$$

Based on the partition F_o, I_n, N_o of $[\![(N \times_F M)]\!]$, a partition of the states S^Π of Π is defined as follows:

- $F_o^\Pi = \{s \in S^\Pi, s \subseteq F_o\}$ is the set of macro-states such that $found$ holds in every state of the macro-state.
- $I_n^\Pi = \{s \in S^\Pi, s \cap N_o = \varnothing, s \cap I_n \neq \varnothing\}$ is the set of macro-states containing states such that either $found$ holds in the state or the state inevitably leads to a state $found$ and at least one of them inevitably leads to a state $found$.
- $N_o^\Pi = S^\Pi \setminus (F_o^\Pi \cup I_n^\Pi)$.

Proposition 3. *As the system N is M-diagnosable, any macro-state from N_o^Π reachable from the initial state of Π that contains a state from F_o will eventually lead to a macro-state of $F_o^\Pi \cup I_n^\Pi$.*

Proof sketch: Π is a diagnoser by construction. It cannot have cycles of macro-states that are ambiguous (macro-states that contain at least a state where the pattern has matched ($found$) and another one where it has not ($\neg found$)) but these ambiguous macro-states must be in N_o^Π as they cannot inevitably lead to a macro-state of F_o^Π.

Example: Let us focus on the previous example from Figure 1. This LPN N (called *Base*) is analysed with the pattern M in Figure 2. The LTS of our product $N \times_F M$ is conducted and the separation with F_o in red and I_n in blue is processed in Figure 3 (labels i stands for unobservable events) and the resulting LTS Π obtained by unobservable closure and determinization is presented in Figure 4. Macro-states in F_o^Π are in red, the ones in I_n^Π are in blue.

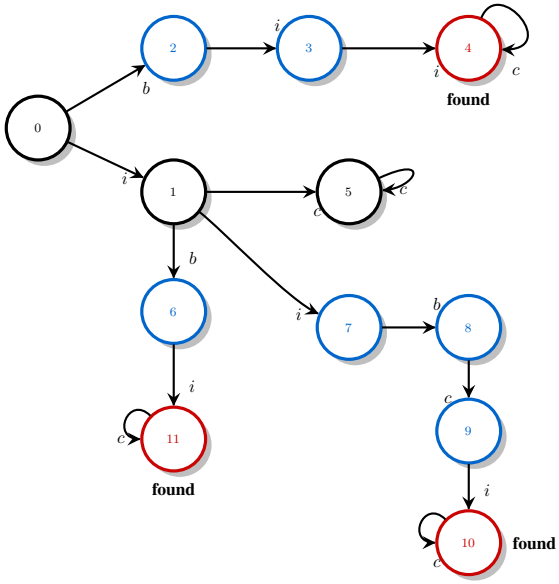

Figure 3: LTS of our product $N \times_F M$

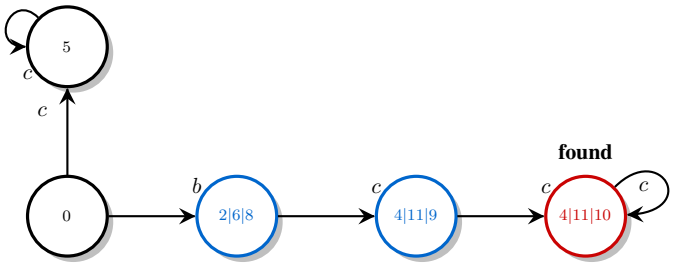

Figure 4: LTS $\Pi = Det(U_c([\![N \times_F M]\!]))$.

The predictability of the pattern M can be concluded.

About the formal property to check in Π

As the system is M-diagnosable, by Proposition 3 and by definition of I_n, it follows that as soon as a macro-state s contains a state from F_o it will eventually lead to a macro-state in F_o^Π. Now, if this state s is in N_o^Π, the system is not M-predictable. Therefore, checking predictability, as detailed in [7], consists now in checking that every predecessor of a macro-state of F_o^Π in Π is either a macro-state of F_o^Π or a macro-state of I_n^Π. As opposed to diagnosability, this property is local and not global so LTL is not well-suited to express this property. It is proposed here to use μ-*calculus* to check on local part of the LTS model Π. With μ-calculus, CTL operators (i.e. AF) can be exploited as well as MEC4 primitives (i.e. $src, rsrc, tgt, rtgt$) as detailed herebelow. The property NOTPREDICTABLE is incrementally described to check on the LTS Π. As illustrated by Figure 4, any macro-state of F_o^Π (in red) holds a property that is denoted:

$$found.$$

To express the property of a state in I_n^Π, the CTL operator is used

- AF : the properties *always finally* hold.

A state in I_n^Π is not in F_o^Π but will eventually lead to such a state, it can be expressed as:

$$\text{INEV} \equiv AF(found) \wedge \neg found$$

Now we want to express the states FRONTIER that are $found$ but that have at least a predecessor that is not a $found$ state. To do so, the MEC primitives tgt and $rsrc$ are required.

- $src(X)$ (resp. $tgt(X)$) stands for the source (resp. target) state of transition X;
- $rsrc(X)$ (resp. $rtgt(X)$) stands for the transition whose X is the source (resp. target) state.

It follows:

$$\text{FRONTIER} \equiv found \wedge tgt(rsrc(\neg(found)))$$

The states PREDECESSORS are then the predecessors of FRONTIER that are not part of FRONTIER:

$$\text{PREDECESSORS} \equiv (src(rtgt(\text{FRONTIER}))) \wedge \neg\text{FRONTIER}$$

Finally, the system is not M-predictable iff there are states in PREDECESSORS that do not inevitably lead to $found$ states:

$$\text{NOTPREDICTABLE} \equiv \text{PREDECESSORS} \wedge \neg\text{INEV}.$$

Example: For Figure 4, we have the following sets:

$$\text{INEV} = \{2|6|8, 4|11|9\}$$
$$\text{FRONTIER} = \{4|11|10\}$$
$$\text{PREDECESSORS} = \{4|11|9\}$$

5 Implementation and Experimental Results

To implement the proposed method, several tools are used in a process chain. Those tools mainly come from the TINA [16] toolbox. Firstly, to check diagnosability, the chain process presented in Figure 5 can be exploited.

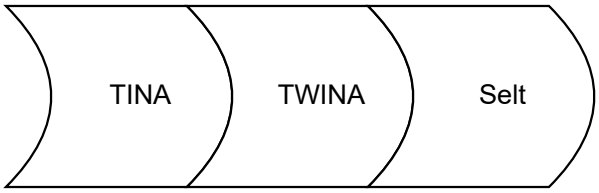

Figure 5: Chain process of the analysis of diagnosability of a pattern.

- TINA: TINA builds various state space abstractions for Petri nets and Time Petri nets. It is used to build our original net N and its pattern M.
- TWINA: TWINA is a tool for analysing the "product" of two Time Petri Nets (TPN), with possibly inhibitor and read arcs. It is used to process our product (with $\times_F$) on classical PN. The output is the LTS $[\![N \times_F M)]\!]$. It is also used to process the copy of our product on the diagnosability analysis.
- Selt : A State/Event LTL model checker. It is used to check our diagnosability formula.

Secondly, to check predictability, the following chain process is performed as presented in Figure 6. This second

Figure 6: Chain process of the analysis of predictability of a pattern

chain process starts by using the same tools as the chain process for diagnosability but ends with another couple of tools that are described herebelow:

- Basileus : Process U_c and Det to get the LTS Π in the TINA format (ktz). This process is a new tool that has been specifically implemented to process the *Unobservable-Closure* and the *Determinization*. This new tool is called *Basileus*. The Unobservable closure is optimized via an OCaml recursive software.
- Muse : Muse model-checks state-event on a Kripke transition system given in ktz format produced by Basileus. It is used to automatically check whether NOTPREDICTABLE is satisfied on Π.

The Tina tools are all available online and the tool Basileus will be after some optimizations.

For the sake of completeness with the regards to the running example *Base* (see Figure 1 and Figure 2), the execution times of the Base exemple are summarized in Table 1, in the *Base* column.

To check the scalability of the proposed method, the method have benn tested on larger examples. For this purpose, our method have been first applied to the following example from [17]. The system is defined in Figure 7. It is the modelling of a transport system, in a untimed version (see the TWINA webpage on the benchmark patterns for more information).

	Base	Transports	Transport9	WODES'08
$TWINA(Diag)$	0.012s	1.572s	9.372s	0.117s
$Selt$	0.027s	0.066s	0.198s	0.015s
Size LTS(Diag)	58 states	14270 states	83834 states	4617 states
$TWINA(Pred)$	0.010s	0.064s	0.133s	0.031s
$Basileus$	0.045s	0.620s	10.949s	0.084s
$Muse$	0.005s	0.005s	0.333s	0.002s
Size LTS(Pred)	12 states	780 states	2124 states	159 states
Size LTS(DET)	5 states	3821 states	27575 states	10 states
$Total$	0.109s	2.327s	20.985s	0.249s

Table 1: Complete set of experimental results.

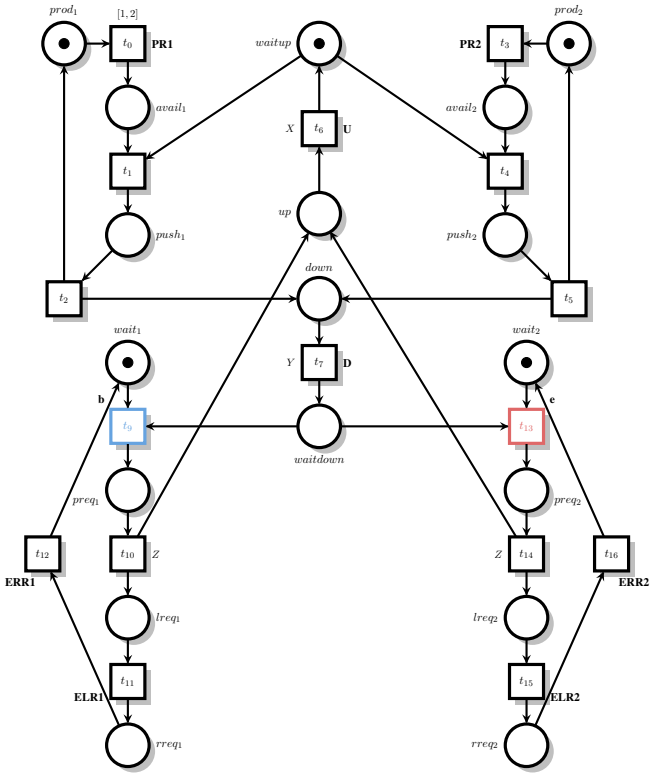

Figure 7: Transport - [Gougam 17]

This model has been slightly modified to add b and e transitions and created the following *3b without e* pattern (Figure 8). Every time an e label occurs, the pattern go back to its orign=inal place.

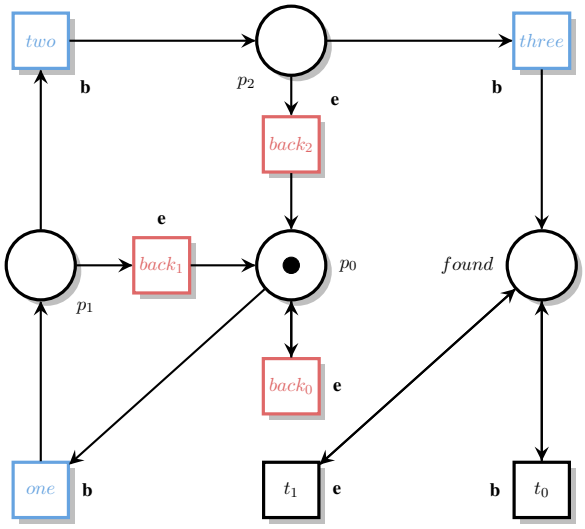

Figure 8: pattern for "three consecutive b without e"

The proposed pattern is not diagnosable and consequently not predictable. However, to check the scalability of the method (product computations, property checking), both diagnosability and predictability tests have been run to measure the performance (see Table 1, Transports column).

We also tried to process the same PN (see Figure 7) with a pattern "nine consecutive b without e" (see Table 1, Transport9 column). And to conclude on our scalability test, we also process the example from WODES'08 [18] with a single fault pattern F (see Table 1, WODES'08 column).

Regarding the analysis of these systems and their respective patterns, the following results have been obtained:

	Base	Transports	Transport9	WODES'08
$Diagnosability$	X			X
$Predictability$	X			

Those results show that our methods can be used to analyse systems up to 100000 states in an efficient way. However, the construction of Π is of exponential complexity and could become cumbersome in much larger systems. Alternatives shoud be investigated.

6 Conclusion

This paper proposes a fully implemented method for checking both diagnosability and predictability of event patterns in a system that is represented as a Labeled Petri Net. The originalilty of this approach is the fact that both checking problems have been translated into two classical model-checking problems that can be solved by off-the shelf model-checking tools. Diagnosability being a global property, the use of LTL formulae to check a property on a twin-plant is quite natural. However, as far as predictability is concerned, one of the difficulty is to express the set of states that inevitably lead to *found* states. To express such a property, we had to build a predictor, resulting from a projection and a determinization, and to express the predictability using a μ-calculus tool (Muse) in which we can benefit of CTL-operators and MEC4 primitives to express a local property. The experimental results that are detailed in the paper show the feasibility of the approach. However, the complexity of the approach lies in the construction of Π that

is exponential in the size of $[\![N]\!] \times_F [\![M]\!]$ due to the unobservable closure and the determinization. Checking the predictability of M in N without the construction of Π is still an open question.

Similarly to the extension of diagnosability to timed systems as in [15; 19], the extension of predictability of patterns to such systems shall be investigated. By extending for instance to time Petri nets (TPN), some undecidability issues may raise and the characterization of TPN sub-classes where predictability can be effectively checked might be necessary.

References

[1] J. Zaytoon and S. Lafortune. Overview of fault diagnosis methods for Discrete Event Systems. *Annual Reviews in Control*, 37(2):308–320, December 2013. 1

[2] Sahika Genc and Stéphane Lafortune. Predictability of event occurrences in partially-observed discrete-event systems. *Autom.*, 45(2):301–311, 2009. 1, 4

[3] Xiang Yin. Verification of prognosability for labeled petri nets. *IEEE Trans. Autom. Control.*, 63(6):1738–1744, 2018. 1

[4] Shigemasa Takai and Ratnesh Kumar. A generalized inference-based prognosis framework for discrete event systems. *IFAC-PapersOnLine*, 50(1):6819–6824, 2017. 20th IFAC World Congress. 1

[5] Xiang Yin and Zhaojian Li. Decentralized fault prognosis of discrete event systems with guaranteed performance bound. *Autom.*, 69:375–379, 2016. 1

[6] Ana T.Y. Watanabe, André Bittencourt Leal, José Eduardo Ribeiro Cury, and Max Hering de Queiroz. Safe controllability using online prognosis. *IFAC-PapersOnLine*, 50:12359–12365, 2017. 1

[7] Thierry Jéron, Hervé Marchand, Sahika Genc, and Stéphane Lafortune. Predictability of Sequence Patterns in Discrete Event Systems. Research Report PI 1834, 2007. 1, 2, 3, 5, 6

[8] Amaury Vignolles, Elodie Chanthery, and Pauline Ribot. An overview on diagnosability and prognosability for system monitoring. In *EUROPEAN CONFERENCE OF THE PROGNOSTICS AND HEALTH MANAGEMENT SOCIETY (PHM Europe)*, (Virtual conference), Italy, July 2020. 1

[9] C. A. Petri. Fundamentals of a theory of asynchronous information flow. In *Information Processing, Proceedings of the 2nd IFIP Congress 1962, Munich, Germany, August 27 - September 1, 1962*, pages 386–390. North-Holland, 1962. 1

[10] Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. Comparison of the expressiveness of timed automata and time petri nets. In Paul Pettersson and Wang Yi, editors, *Formal Modeling and Analysis of Timed Systems, Third International Conference, FORMATS 2005, Uppsala, Sweden, September 26-28, 2005, Proceedings*, volume 3829 of *Lecture Notes in Computer Science*, pages 211–225. Springer, 2005. 2

[11] B. Berthomieu, F. Peres, and F. Vernadat. Bridging the gap between timed automata and bounded time Petri nets. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 4202 of *LNCS*. Springer, 2006. 2

[12] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. In *Proceedings IFIP*, 1983. 2

[13] Éric Lubat, Silvano Dal Zilio, Didier Le Botlan, Yannick Pencolé, and Audine Subias. A state class construction for computing the intersection of time petri nets languages. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 11750 of *LNCS*. Springer, 2019. 2

[14] Thierry Jéron, Hervé Marchand, Sophie Pinchinat, and Marie-Odile Cordier. Supervision patterns in discrete event systems diagnosis. In *International Workshop on Discrete Event Systems*, 2006. 3

[15] Éric Lubat, Silvano Dal-Zilio, Didier Le Botlan, Yannick Pencolé, and Audine Subias. A new product construction for the diagnosability of patterns in time petri net. In *59th IEEE Conference on Decision and Control, CDC 2020, Jeju Island, South Korea, December 14-18, 2020*, pages 104–109. IEEE, 2020. 4, 8

[16] B. Berthomieu, P.-O. Ribet, and F. Vernadat. The tool TINA–construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14), 2004. 6

[17] Houssam-Eddine Gougam, Yannick Pencolé, and Audine Subias. Diagnosability analysis of patterns on bounded labeled prioritized Petri nets. *Discrete Event Dynamic Systems*, 27(1), 2017. 6

[18] Alessandro Giua. A benchmark for diagnosis. In *Benchmark Session of WODES'08 Int. Workshop on Discrete Event Systems*, 2007. 7

[19] Yannick Pencolé and Audine Subias. Diagnosability of event patterns in safe labeled time Petri nets: a model-checking approach. *IEEE Transactions on Automation Science and Engineering*, 2021. 8