Diagnosability and Predictability of pattern in Labelled Petri Nets

Eric Lubat, Camille Coquand, Yannick Pencole, Audine Subias

To cite this version:

Eric Lubat, Camille Coquand, Yannick Pencole, Audine Subias. Diagnosability and Predictability of pattern in Labelled Petri Nets. 33rd International Workshop on Principle of Diagnosis (DX 2022), LAAS-CNRS-ANITI, Sep 2022, Toulouse, France. hal-03773804

HAL Id: hal-03773804
https://hal.science/hal-03773804
Submitted on 9 Sep 2022

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.
1 Introduction

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.

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 the methods and lastly presents experimental results. Finally conclusion and perspectives are given in Section 6.

Definition 1 (Petri Nets). A PN is a tuple \( \langle P, T, \text{Pre}, \text{Post} \rangle \) where \( P \) and \( T \) are the set of places and transitions; \( \text{Pre} : P \times T \rightarrow \mathbb{N} \) and \( \text{Post} : T \times P \rightarrow \mathbb{N} \) 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 \( m : P \rightarrow \mathbb{N} \) from places in \( P \) to natural numbers.

A marked Petri net is a tuple \( N = \langle P, T, \text{Pre}, \text{Post}, m_0 \rangle \) 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 \( \forall p \in P, m(p) \geq \text{Pre}(p,t) \).

The effective firing of transition \( t \) from a marking \( m \) leads to the marking \( m' \) such that \( \forall p \in P, m'(p) = m(p) - \text{Pre}(p,t) + \text{Post}(p,t) \). 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
that \( m_0 \xrightarrow{t_1} \ldots \xrightarrow{t_k} m \). 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 \( \exists P \in N, \forall p \in P, m(p) \leq k \)).

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

**Definition 3 (Labeled Petri Net).** A Labeled Petri Net (LPN for short) is a tuple \( \langle P, T, \text{Pre}, \text{Post}, \Sigma, L, m_0 \rangle \) where \( \langle P, T, \text{Pre}, \text{Post}, m_0 \rangle \) is a marked Petri Net, \( \Sigma \) is an alphabet and \( L : T \rightarrow \Sigma \) 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 Labeled Transition System (LTS), as introduced in [7]. Let \( \mathcal{E}(m) \) denote the set of transitions enabled by a marking \( m \), i.e. \( \mathcal{E}(m) = \{ t \mid t \in T, \forall p \in P, m(p) \geq \text{Pre}(p, t) \} \). A state of an LTS is a couple \( (m, E(m)) \) where \( m \) is a marking and \( E(m) \) 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. \( \mathcal{E}(m) = \{ t \mid t \in T, \forall p \in P, m(p) \geq \text{Pre}(p, t) \} \). A state of an LTS is a couple \( (m, E(m)) \) where \( m \) is a marking and \( E(m) \) is the enabled transitions associated with \( m \). In the LTS formalism, transitions are associated with events (also called actions) that label the underlying LPN.

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

In the following, \((s, \alpha, s') \in \rightarrow\) 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 \((S, s_0, \rightarrow)\) 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 \( N \langle P, T, \text{Pre}, m_0 \rangle \) with the labelling function \( L : T \rightarrow \Sigma \cup \{ \varepsilon \} \) is the Labeled Transition System (LTS) \([N] = (S, s_0, \Sigma, \rightarrow) \) where \( S \) is the smallest set containing \( s_0 \) and closed by \( \rightarrow \), 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 \( \rightarrow \subseteq S \times (\Sigma \cup \{ \varepsilon \}) \times S \) is the relation such that for all states \( (m, \mathcal{E}(m)) \) in \( S \):
  - \( (m, \mathcal{E}(m)) \xrightarrow{\varepsilon} \mathcal{E}(m') \) iff:
  - \( \forall p \in P, m'(p) = m(p) - \text{Pre}(p, t) + \text{Post}(p, t) \)

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

Consecutive transitions can always be grouped together, meaning that when \((m, \mathcal{E}(m)) \xrightarrow{\varepsilon} (m', \mathcal{E}(m')) \) and \((m', \mathcal{E}(m')) \xrightarrow{\varepsilon} (m'', \mathcal{E}(m'')) \) then necessarily \((m, \mathcal{E}(m)) \xrightarrow{\varepsilon} (m'', \mathcal{E}(m'')) \). By contrast, a trace is the word obtained from an execution when only the discrete actions without \( \varepsilon \) 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 \( L : T^* \rightarrow (\Sigma \cup \{ \varepsilon \})^* \) as \( L(t_0 \ldots t_n) = L(t_0), L(t_1) \ldots L(t_n) \). From a state \( s \), a run \( \sigma \) is said to be acceptable (denoted \( s \xrightarrow{\sigma} s' \)) if there exists a state \( s' \) such that \( s \xrightarrow{\sigma} s' \).

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 \( \{ N_i = (P_i, T_i, \text{Pre}_i, \text{Post}_i, \Sigma_i, L_i, m_0^i) \}_{i \in \{1, 2\}} \) be two LPNs. Let \( L \) be an alphabet. The product \( N_1 \times_L N_2 \) is defined as:

- \( \forall t \in T_i \Rightarrow L_{12}(t) = L_i(t) \land \forall p \in P_i, \text{Pre}_{12}(p, t) = \text{Pre}_i(p, t), \text{Post}_{12}(p, t) = \text{Post}_i(p, t), \forall p \in P_j, j \neq i, \text{Pre}_{12}(p, t) = 0, \text{Post}_{12}(p, t) = 0; \)
- \( \forall t = (t_1, t_2), t_1 \in T_1, t_2 \in T_2 \Rightarrow \langle L_1(t_1) = L_2(t_2) = L_{12}(t) \in L \rangle \land \langle \forall p \in P_1 \cup P_2, \text{Pre}_{12}(p, t_1) > 0 \Rightarrow \text{Pre}_{12}(p, t_2) = 0 \rangle \land \langle \forall p \in P_1, \text{Pre}_{12}(p, t_1) = \text{Pre}_1(p, t_1), \text{Post}_{12}(p, t_1) = \text{Post}_1(p, t_1), \forall p \in P_2, \text{Pre}_{12}(p, t_2) = \text{Pre}_2(p, t_2), \text{Post}_{12}(p, t_2) = \text{Post}_2(p, t_2) \rangle \).

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 \( [N_1] = ([S_1], s_1^0, \Sigma_1, \rightarrow_1) \) and \( [N_2] = ([S_2], s_2^0, \Sigma_2, \rightarrow_2) \) are two LTS.

The product of \([N_1] \times_L [N_2] = ([S_1] \times S_2, (s_1^0, s_2^0), \Sigma, \rightarrow) \) with \( \Sigma = \Sigma_1 \cup \Sigma_2 \) and \( \rightarrow \) the smallest relation obeying the following rules (\( \alpha \in \Sigma_1 \cup \Sigma_2 \cup \{ \varepsilon \} \)):

\[
\begin{align*}
S_1 \xrightarrow{\alpha} S_1 & \quad \alpha \in \Sigma_1 \setminus \Sigma_2 \cup \{ \varepsilon \} \\
(s_1, s_2) & \xrightarrow{\varepsilon} (s_1', s_2) \\
S_2 \xrightarrow{\alpha} S_2 & \quad \alpha \in \Sigma_2 \setminus \Sigma_1 \cup \{ \varepsilon \} \\
(s_1, s_2) & \xrightarrow{\varepsilon} (s_1', s_2) \\
S_1 \xrightarrow{\alpha} S_2 & \quad \alpha \in \Sigma_1 \setminus \Sigma_2 \cup \{ \varepsilon \} \\
(s_1, s_2) & \xrightarrow{\varepsilon} (s_1', s_2') \\
S_2 \xrightarrow{\alpha} S_1 & \quad \alpha \in \Sigma_2 \setminus \Sigma_1 \cup \{ \varepsilon \} \\
(s_1, s_2) & \xrightarrow{\varepsilon} (s_1', s_2') \\
S_1 \xrightarrow{\alpha} S_2 & \quad \alpha \in \Sigma_1 \setminus \Sigma_2 \cup \{ \varepsilon \} \\
(s_1, s_2) & \xrightarrow{\varepsilon} (s_1', s_2') \\
S_2 \xrightarrow{\alpha} S_1 & \quad \alpha \in \Sigma_2 \setminus \Sigma_1 \cup \{ \varepsilon \} \\
(s_1, s_2) & \xrightarrow{\varepsilon} (s_1', s_2') \\
\end{align*}
\]
The set of reachable states is such that \( \Delta_{\{s\}}(s, \sigma) \stackrel{def}{=} \{s' \in S \mid s \xrightarrow{\sigma} s'\} \).

## 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, \( \Sigma \), is partitioned in three different sets:

- \( \Sigma_o \) the set of observable events,
- \( \Sigma_u \) 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 \( \rho \) ending with an unobservable event, there exists \( n \in \mathbb{N} \) such that for all continuations \( \rho' \) such that \( \rho, \rho' \in L(N) \) and \( |\rho'| = n \), \( \rho' \) 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 \( \Sigma_u = \{a, f_1, f_2\} \).

![Figure 1: LPN N of the Base System](image)

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 \( M = (P_M, T_M, \text{Pre}_M, \text{Post}_M, F, \mathcal{L}_M, m_0^M) \) is a safe LPN such that:

1. the alphabet \( F \) is a subset of \( \Sigma_o \);
2. there exists a place \( f \in P_M \) such that the marking \( m \) with \( \forall p \in P_M \setminus \{f\}, m(p) = 0 \land m(f) = 1 \) is reachable;
3. only marking \( m_0^M \) is reachable after the fire of a transition from marking \( m \).

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

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: \( F \cap \Sigma_u = \emptyset \).

For instance, consider the following well-formed Pattern \( M \) in Figure 2. This pattern represents the occurrence of one event followed by an event or one event followed by another event. \( M \) represents any behaviour of the system where two events must occur at least once whatever their occurrence order: \( L_\text{found}(M) = \{(f_1 f_3 f_2 f_4 f_1 f_3 f_2 f_1 f_3 f_2) \} \). As opposed to the automata representing patterns in [14] that must represent any event from the system’s alphabet \( \Sigma \), here the description of the pattern is succinct and only requires to model the events of interests (i.e. \( f_1 \) and \( f_2 \)).

![Figure 2: Pattern M](image)

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

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 ap-
plying a product between semantics [N] and [M], as sum-
marized 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:

\[
M(N, M) = \{ \rho = \mathcal{L}(\sigma) \mid (m_0, \mathcal{E}(m_0)) \xrightarrow{\mathcal{L}(\sigma)} (m_0, \mathcal{E}(m)) \in [N][M] \land m(\text{found}) = 1 \}
\]

Looking back at the system in Figure 1 and pattern
in Figure 2, the execution \( \rho_1 = f_1 \text{abcf } f_2 \text{cc} \) of the system
matches the pattern as \( f_1 f_2 \) is a subword of \( \rho_1 \) and \( f_1 f_2 \in L(\text{found}(M)) \). This execution is part of \([N][M]\]
as it is \((([m_0, \mathcal{E}(m_0)), ([m_0', \mathcal{E}(m_0')]) \xrightarrow{f_1 \text{abcf } f_2 \text{cc}} (m, \mathcal{E}(m)), (m, \mathcal{E}(m)) \) with \( m(\text{found}) = 1 \). 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 ob-
servable projection of an execution \( \rho = e, \rho' \in L(N) \) is:

- \( P(\rho) = e.P(\rho') \) if \( e \in \Sigma_o \)
- \( P(\rho) = P(\rho') \) if \( e \notin \Sigma_o \)

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

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

A system is M-predictable if it is always possible to as-
sert 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

\[
\exists n \in \mathbb{N}, \forall \rho_1 \in L(N) \cap \Sigma^* \Sigma_o, \text{ with } \rho_1 \ni M,
\exists (\rho_1', \rho_1'), \rho_1', \rho_1' \in \{ \varepsilon \} \cup L(N) \cap \Sigma^* \Sigma_o, \rho_1' \ni M \text{ such that:}
\]

\[
\forall \rho_2 = \rho_2' \rho_2'' \in L(N), P(\rho_2') = P(\rho_1'),
||P(\rho_2'||) \geq n \Rightarrow \rho_2 \ni M.
\]

**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 exec-
uction can produce \( bcc \). By observing \( bcc \), the diagnos-
er 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 cer-
tainty. 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 \( \rho_1, \rho_2 \)
of the system such that \( P(\rho_1) = P(\rho_2) \) and \( \rho_1 \ni M \) but
\( \rho_2 \not\ni M \). It follows that there is no prefix of \( P(\rho_1) \) 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 pro-
posed. This method is based on the verification of prop-
erties on Kripke structures of products of the LTS of the
system N and the pattern M. As presented in the pre-
vious 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, how-
ever 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 straigh-
toward 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 set-
ing, the twin plant is the LPN \((N_1 \times N_2) \times \Sigma_o (N_2 \times \Sigma_o M_2)\)
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 obser-
vable labels \( \Sigma_o \). A critical pair is an infinitely long execu-
tion of the twin-plant which matches \( M_1 \) but not \( M_2 \) and
then represents a couples of executions from \( N \) with the same
observations but only one matches the pattern (so it is not di-
agnosable). 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 tem-
poral 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 obser-
vable label \( \Sigma_o \), is diagnosable for pattern \( M \) if and only if
all the maximal executions of the product \((N_1 \times M_1) \times \Sigma_o \)
\((N_2 \times F M_2)\) satisfy \( (\diamond \text{found}1) \Rightarrow (\diamond \text{found}2 \lor \text{dead}) \).*
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 \( M_1 \) (\( \text{found} \)) it will eventually matches the copy pattern \( M_2 \) or be a deadlock (\( \text{found} .2 \lor \text{dead} \)), hence the absence of critical pairs.

Our process to check \textit{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 \((N_1 \times F M_1) \times \Sigma, (N_2 \times F M_2)\).

**Step D3.** Use an LTL checker to conclude on the LTL formula \( \langle \text{found} .1 \rangle \Rightarrow \langle \text{found} .2 \lor \text{dead} \rangle \).

### 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 \( \Pi \), called a predictor in [7], and secondly on a formal property to check on \( \Pi \).

**Construction of \( \Pi \)**

Starting with the system \( N \) and the pattern \( M \), the LTS \( [[(N \times F M)]] = (S, s_0, \Sigma, F) \) is first built and the following partition of states \( S \) is considered:

- \( F_o \) is the set of states that hold the property \textit{found}.
- \( I_n \) is the set of states which inevitably lead to states in \( F_o \):

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

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

The computation of \( \Pi \) 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 \( \llbracket N \rrbracket = (S, s_0, \Sigma, \rightarrow) \) be the LTS of an LPN \( N \) and \( \Sigma_o \subseteq \Sigma \) be the observable events of \( N \), the Unobservable-Closure of \( \llbracket N \rrbracket \) is \( U_c(\llbracket N \rrbracket) = (S, s_0, \Sigma_o, \rightarrow_{U_c}) \) where for any \( s, s' \in S, \alpha \in \Sigma_o, s \xrightarrow{\alpha} s' \in U_c(\llbracket N \rrbracket) \) whenever there exists \( \sigma \in \Sigma \) such that \( s \xrightarrow{\sigma} s' \) in \( \llbracket N \rrbracket \).

The second step consists in building a deterministic LTS.

**Definition 14** (Determination of a LTS). Let \( \llbracket N \rrbracket = (S, s_0, \Sigma, \rightarrow) \) be a LTS over the LPN \( N \) and an alphabet \( \Sigma \). The Determination of \( \llbracket N \rrbracket \) is the LTS \( \text{Det}(\llbracket N \rrbracket) = \langle X, X_0, \Sigma, \rightarrow_{\text{Det}} \rangle \) where \( X = 2^S \) (the set of subsets of \( S \) called macro-states), \( X_0 = \{ s_0 \} \), and \( \rightarrow_{\text{Det}} = \{ (s^D, \alpha, \Delta_{\llbracket N \rrbracket}(X, \alpha)), s^D \in X \land \alpha \in \Sigma \} \).

Finally, the predictor \( \Pi \) is:

\[
\Pi = \text{Det}(U_c(\llbracket (N \times F M) \rrbracket)).
\]

Based on the partition \( F_o, I_n, N_o \) of \( [[(N \times F M)]] \), a partition of the states \( S^\Pi \) of \( \Pi \) is defined as follows:

- \( I^\Pi = \{ s \in S^\Pi, s \subseteq F_o \} \) is the set of macro-states such that \textit{found} holds in every state of the macro-state.

- \( N^\Pi = S^\Pi \setminus (F_o \cup I^\Pi) \) is the set of macro-states containing states such that either \textit{found} holds in the state or the state inevitably leads to a state \textit{found} and at least one of them inevitably leads to a state \textit{found}.

**Proposition 3.** As the system \( N \) is \( M \)-diagnosable, any macro-state from \( N^\Pi \) reachable from the initial state of \( \Pi \) that contains a state from \( F_o \) will eventually lead to a macro-state of \( F^\Pi_o \cup I^\Pi_o \).

**Proof sketch:** \( \Pi \) 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 \textit{found} and another one where it has not \textit{(!found)}) but these ambiguous macro-states must be in \( N^\Pi \) as they cannot inevitably lead to a macro-state of \( F^\Pi_o \).

**Example:** Let us focus on the previous example from Figure 1. This LPN \( N \) (called \textit{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 \( \Pi \) obtained by unobservable closure and determinization is presented in Figure 4. Macro-states in \( F^\Pi_o \) are in red, the ones in \( I^\Pi_o \) are in blue.
The predictability of the pattern $M$ can be concluded.

**About the formal property to check in $I$**

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 $F^R_n$. Now, if this state $s$ is in $N^R_n$, 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 $F^R_n$ in $I$ is either a macro-state of $F^R_o$ or a macro-state of $I_n$. 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 $\mu$-calculus to check on local part of the LTS model $I$. With $\mu$-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 $\text{NPREDICTABLE}$ is incrementally described to check on the LTS $I$. As illustrated by Figure 4, any macro-state of $F^R_n$ (in red) holds a property that is denoted:

$$\text{found}.$$

To express the property of a state in $F^R_n$, the CTL operator is used

- $AF$: the properties always finally hold.

A state in $F^R_n$ is not in $F^R_o$ but will eventually lead to such a state, it can be expressed as:

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

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

- $\text{src}(X)$ (resp. $\text{tg}t(X)$) stands for the source (resp. target) state of transition $X$;
- $\text{rsrc}(X)$ (resp. $\text{rt}gt(X)$) stands for the transition whose $X$ is the source (resp. target) state.

It follows:

$$\text{FRONTIER} \equiv \text{found} \land \text{tg}t(\text{rsrc}(\neg(\text{found})))$$

The states $\text{PREDECESSORS}$ are then the predecessors of FRONTIER that are not part of FRONTIER:

$$\text{PREDECESSORS} \equiv (\text{src}(\text{tg}t(\text{FRONTIER}))) \land \neg \text{FRONTIER}$$

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

$$\text{NPREDICTABLE} \equiv \text{PREDECESSORS} \land \neg \text{INEV}.$$  

**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.

- **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 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 $I$ 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-checker state-event on a Kripke transition system given in ktz format produced by Basileus. It is used to automatically check whether $\text{NPREDICTABLE}$ is satisfied on $I$.

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 example are summarized in Table 1, in the $Base$ column.

To check the scalability of the proposed method, the method have been 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).
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 original place.

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:

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>Transports</th>
<th>Transport9</th>
<th>WODES’08</th>
</tr>
</thead>
<tbody>
<tr>
<td>$TWINA(\text{Diag})$</td>
<td>0.012s</td>
<td>1.572s</td>
<td>9.372s</td>
<td>0.117s</td>
</tr>
<tr>
<td>$Selt$</td>
<td>0.027s</td>
<td>0.066s</td>
<td>0.198s</td>
<td>0.015s</td>
</tr>
<tr>
<td>Size LTS(\text{Diag})</td>
<td>58 states</td>
<td>14270 states</td>
<td>83834 states</td>
<td>4617 states</td>
</tr>
<tr>
<td>$TWINA(\text{Pred})$</td>
<td>0.010s</td>
<td>0.064s</td>
<td>0.133s</td>
<td>0.031s</td>
</tr>
<tr>
<td>$Basileus$</td>
<td>0.045s</td>
<td>0.620s</td>
<td>10.949s</td>
<td>0.084s</td>
</tr>
<tr>
<td>Size LTS(Pred)</td>
<td>12 states</td>
<td>780 states</td>
<td>2124 states</td>
<td>159 states</td>
</tr>
<tr>
<td>Size LTS(DET)</td>
<td>5 states</td>
<td>3821 states</td>
<td>27575 states</td>
<td>10 states</td>
</tr>
<tr>
<td>Total</td>
<td>0.109s</td>
<td>2.327s</td>
<td>20.985s</td>
<td>0.249s</td>
</tr>
</tbody>
</table>

Table 1: Complete set of experimental results.

Figure 7: Transport - [Gougam 17]

Figure 8: pattern for “three consecutive $b$ without $e$”

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 originality 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 $\mu$-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 $\Pi$ 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 $\Pi$ 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 arise and the characterization of TPN sub-classes where predictability can be effectively checked might be necessary.

References


