A Model Checking method to solve the event pattern diagnosis problem in safe labeled time Petri nets
Yannick Pencolé, Audine Subias, Camille Coquand

To cite this version:
Yannick Pencolé, Audine Subias, Camille Coquand. A Model Checking method to solve the event pattern diagnosis problem in safe labeled time Petri nets. 32nd International Workshop on Principles of Diagnosis DX’21, Sep 2021, Hamburg, Germany. hal-03348338

HAL Id: hal-03348338
https://laas.hal.science/hal-03348338
Submitted on 18 Sep 2021

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.
A Model Checking method to solve the event pattern diagnosis problem in safe labeled time Petri nets

Yannick Pencolé\textsuperscript{1} and Audine Subias\textsuperscript{2} and Camille Coquand\textsuperscript{2}
\textsuperscript{1} LAAS-CNRS, Univ. Toulouse, CNRS, Toulouse, France.
e-mail: yannick.pencole@laas.fr
\textsuperscript{2} LAAS-CNRS, Univ. Toulouse, INSA, Toulouse, France.
e-mail: audine.subias@laas.fr, camille.coquand@laas.fr

Abstract

This paper addresses the problem of identifying whether a given pattern of unobservable events has occurred in a partially observable timed discrete event system. The systems that are considered here are modelled with safe labeled time Petri nets. To solve the problem, we propose to use a model checking approach. Given the model of the system, the pattern to analyse, and the sequence of observations produced by the system, the method firstly builds automatically a labeled time Petri net and a pair of queries that represent the pattern diagnosis problem to solve. The second step then consists in using a model-checker for time Petri net that provides the answer to the queries. The solution provided by the model-checker is then analysed to determine whether the pattern under investigation has definitely occurred or not.

1 Introduction

Classically, the problem of model-based fault diagnosis in discrete event systems (DES) is the problem of determining whether a set of unobservable and faulty events has occurred in the system based on a complete behavioural model and a sequence of observations produced by the system. This initial problem, originally introduced in [11], has been addressed for decades based on different formalisms such as automata, Petri nets, process algebra and a survey of these techniques can be found in [2] and for specifically Petri nets in [3].

Based on these previous contributions, two independent extensions have been proposed. The first one is about dealing with time. In the initial definition of the problem, time is just characterized by events occurring sequentially but their dates of occurrences are not taken into account. The date itself, as a numerical piece of information, is an observation and depending on the date of a given observation the diagnosis might be different. The extension of the problem to deal with time has been formally addressed in [4] where the considered formalism is time automata. The decidability question as well as some complexity results can also be found in [5]. A few method based on time automata have then been proposed in [6] and more recently in [7]. Extensions to deal with time have also been addressed with formalisms like time Petri nets [8; 9; 10; 11].

The second type of extension is about the nature of what is diagnosed. In the previous cited pieces of work, the result of the diagnosis method is about the occurrence of faults represented as single events. In [12], the notion of supervision patterns is introduced as automata, also called temporal specification in [13] or semantic patterns in [14]. A supervision pattern is a specific assembling of events and this assembling is considered as faulty or more generally as critical enough so that knowing whether this pattern of events has definitely occurred in the system is of importance to assist any further decision (repair, reconfiguration, maintenance). A succinct representation of such patterns is proposed as Petri nets in [15] and a method for diagnosing such patterns on labeled Petri net is proposed in [16].

The goal of this paper is to extend the method proposed in [16] to Labeled Time Petri nets (LTPN) so that we can benefit of the observation of dates to make the pattern diagnosis more accurate. We propose to turn the pattern diagnosis problem over LTPN as a reachability problem over LTPN that can be solved by a model-checker. Given a pattern, the model of a system and a sequence of observations, the proposed method builds a single LTPN that gathers all the information and designs a set of formal properties over this LTPN that will produce the diagnosis result. The effective construction of this LTPN requires specific products that have been recently introduced in [17].

2 Background

This section recalls the mathematical background that will be used throughout this paper.

2.1 Timed language

A timed sequence over a set of labels $\Sigma$ is a sequence $\rho$ of pairs $(s,d)$ where $d$ is a duration and $s$ is a symbol of $\Sigma \cup \{\lambda\}$. Throughout this paper, symbol $\lambda$ will represent the occurrence of time and will be always considered as a silent event ($\lambda$ may be part of $\Sigma$ or not). The set of timed sequences over $\Sigma$ is denoted $T(\Sigma)$ and $T(\Sigma) \subseteq (\Sigma \cup \{\lambda\} \times \mathbb{R}^+)^*$. Each timed sequence $\rho$ has a canonical representation denoted $[\rho]$ with only one $\lambda$ at the end. Suppose for instance that $\Sigma = \{a, b, c\}$, then $\rho = (a,3).(c,4).(\lambda,2).(b,1).(b,2).(\lambda,4)$ is a timed sequence of $T(\Sigma)$ also denoted $3a4c2\lambda1b2b4\lambda$. The canonical representation of $\rho$ is $[\rho] = 3a4c2\lambda1b2b4\lambda$. Throughout this paper, only canonical representations are considered. Given two alphabets $\Sigma_1, \Sigma_2$, the projection $P_{\Sigma_1 \rightarrow \Sigma_2}$; $T(\Sigma_1) \rightarrow T(\Sigma_2)$ of a canonical timed sequence is defined as follows.

\[ T(\Sigma_1) \rightarrow T(\Sigma_2) \]
2.2 Labeled time Petri nets

Definition 1 (LPN). A Labeled Petri Net (LPN for short) is a 5-uple \(N = (P, T, A, E, \ell)\) such that:

- \(P\) is a finite set of places;
- \(T\) is a finite set of transitions (\(P \cap T = \emptyset\));
- \(A \subseteq (P \times T) \cup (T \times P)\) is a binary relation that models the arcs between the places and the transitions;
- \(E\) is a finite set of transition labels;
- \(\ell: T \rightarrow E\) is the transition labeling function.

The \textit{preset} \(\text{pre}(t)\) of a transition \(t\) is the set of (input) places \(\text{pre}(t) = \{p \in P : (p, t) \in A\}\) and the \textit{post-set} \(\text{post}(t)\) of \(t\) is the set of (output) places \(\text{post}(t) = \{p \in P : (t, p) \in A\}\). A state of an LPN is a \textit{marking} that is a function \(M: P \rightarrow \mathbb{N}\) that assigns to each place a number of tokens (or marks). All along this paper, we will consider that any LPN is safe i.e. \(\text{preset}(t) \leq \text{post}(t)\) for all \(t\) in \(T\). Throughout this paper, the static interval \(I\) is an interval of reals whose bounds are either positive rationals or \(\pm\infty\).

Definition 2 (LTPN). A Labeled Time Petri Net (LTPN for short) is a 6-uple \(N = (P, T, A, E, \ell, I)\) such that:

- \(\langle P, T, A, E, \ell, I \rangle\) is an LPN;
- \(I_s: T \rightarrow I_{\mathbb{Q}^+}\) is the static time interval function that maps any transition to an interval of \(I_{\mathbb{Q}^+}\) (an interval of \(I_{\mathbb{Q}^+}\) is an interval of reals whose bounds are either positive rationals or \(\pm\infty\)).

A state of an LTPN is a couple \(S = \langle M, I \rangle\) where \(I\) is a partial firing interval application \(I: T \rightarrow I_{\mathbb{R}^+}\) that associates to any enabled transition \(t\) a time interval of \(I_{\mathbb{R}^+}\) with bounds in \(I_{\mathbb{R}^+}\) in which transition \(t\) can be fired. The lower bound of \(I(t)\), also called the date of earlier firing, is denoted \([I(t)]\), and its upper bound, also called the date of later firing, is denoted \([-I(t)]\). The initial state of a marked LTPN is given by \(S_0 = \langle M_0, I_0 \rangle\) where \(I_0\) is such that \(I_0(t) = I_s(t)\) for any transition \(t\) enabled by \(M_0\).

Intuitively, in an LTPN a transition is firable from a marking \(M\) if it is enabled for a sufficient amount of time i.e an amount of time within its own static time interval. Formally, a net transition \(t\) is firable from a state \(S = \langle M, I \rangle\) at a relative date \(\theta\) if and only if:

1. \(t \in \text{enabled}(M)\);
2. \(\theta \geq [I(t)]\) and for any transition \(t'\) enabled by \(M, \theta\) is not greater than the later firing date of \(t': \theta \leq [-I(t')]\) if \(I(t')\) is right-closed.

Firing a firable transition \(t\) at the relative date \(\theta\) from the state \(S = \langle M, I \rangle\) leads to a state \(S' = \langle M', I' \rangle\) such that:

- \(M'\) is such that \(\forall p \in \text{pre}(t) \setminus \text{post}(t), M'(p) = 0, \forall p \in \text{post}(t) \setminus \text{pre}(t), M'(p) = 1\), else \(M'(p) = M(p)\);
- For any transition \(t' \in T\) \((t' \neq t)\) enabled by \(M\) after the fire of \(t\) and enabled by \(M'\)
  - if \(I(t') = [a, b], I'(t') = [\max(0, a - \theta), b - \theta]\);
  - if \(I(t') = [a, +\infty], I'(t') = [\max(0, a - \theta), +\infty]\).
  - else \(I'(t') = I_s(t')(t')\).

In the following, the fire of a transition \(t\) at the relative date \(\theta\) is denoted: \(\langle M, I \rangle \theta \rightarrow \langle M', I' \rangle\). A state \(S\) is reachable in a marked LTPN if it exists a sequence of firable transitions \(S_0 \theta_1 \rightarrow S_1 \theta_2 \rightarrow S_2 \rightarrow \ldots \) and enabled \(S_0 \rightarrow S\) where \(r\) is the timed transition sequence \(r = \theta_1 t_1 \theta_2 t_2 \ldots \theta_k t_k\).

Consider now a run \(r\) of an LTPN \(S_0 \rightarrow S = \langle M, I \rangle\), we can associate with \(r\) a set of canonical timed sequences, denoted \(\theta_r(r)\) as follows:

\[
\theta_r(r) = \{[\theta_r \ell(r)](d)l, d \in [0, d_{\max}]\},
\]

(1)
where \([\theta_r \ell(r)](d)l\) is the earliest timed sequence of \(r\):

- if \(r\) is the empty run (no transition) then \([\theta_r \ell(r)](r) = 0\);
- if \(r = \theta_1 t_1 \theta_2 t_2 \ldots \theta_k t_k, k > 0\) then \([\theta_r \ell(r)](r) = [\theta_{\ell(t_1)} \theta_{\ell(t_2)} \ldots \theta_{\ell(t_k)}](0)\).

Date \(d_{\max}\) is the latest possible duration stay in state \(S\) (i.e. the minimal upper bound of the current firing intervals \(d_{\max} = \min_{t \in \text{enabled}(M)}([I(t)])\)). Based on the sets \(\theta_r(r)\) we can define a timed language generated by an LTPN. Let \(Q\) be a subset of \(R(N, M_0)\), a timed sequence belongs to the language generated by the marked LTPN if there is a run of the LTPN that generates the sequence and that leads to a state whose marking is in \(Q\); such a run is called an admissible run.

Definition 3 (Language of an LTPN over Q). The language generated by a marked LTPN \(N\) over \(Q\) is

\[
\mathcal{L}(N, Q) = \bigcup_{r:S_0 \rightarrow r S = \langle M, I \rangle \wedge M \in Q} \theta_r(r)
\]

2.3 LTPN extensions

We recall here two classical LTPN extensions that will be also used in this paper: \textit{inhibitor arc} and \textit{transition priorities}. An inhibitor arc modifies the enabling conditions of transition \(t\) that were detailed previously for the so-called regular arcs. With an inhibitor arc \((p, t)\), the input place \(p\) must be empty to enable \(t\). A \textit{priority} defines a binary relation between two transitions of an LTPN, denoted \(t_1 \succ t_2\) (\(t_1\) has priority over \(t_2\)), that is transitive, not reflexive and not symmetrical. A firable transition is not allowed to fire if a transition with a higher priority is firable at the same date.

3 Problem statement

The fault diagnosis problem has been formally defined with the help of timed automata in [4]. We propose here to extend this definition to deal with more complex behaviors than single faults.
3.1 System model

We consider throughout this paper that the supervised system is modelled as a safe LTPN denoted $\Theta = \langle P_\Theta, T_\Theta, A_\Theta, \Sigma_\Theta, \iota_\Theta, I_\Theta, \lambda_\Theta \rangle$ with an initial marking denoted $M_{\Theta\Theta}$. $\Sigma_\Theta$ represents the set of event types that are effectively generated by the system ($\lambda \not\subseteq \Sigma_\Theta$). In this paper, we assume that according to the instrumentation only a subset of transitions can be observed. These transitions are associated with a sensor that generates an output turned into an observable event by a measurement function. We consider also unobservable transitions that represent effective system’s event but not associated with any kind of sensors. Transition labels $\Sigma_\Theta$ are then partitioned into two subsets: $\Sigma^O_\Theta$ is the set of observable events and $\Sigma^U_\Theta$ the set of unobservable events. Finally, we assume that the model does not contain any zeno run (a zeno run is an infinite sequence of transitions that can occur in a finite amount of time which is unrealistic in real systems).

Figure 1 presents the model of a system $\Theta$. It consists of two concurrent processes synchronized to perform together a specific activity. The first process (on the left part of the figure) is composed of several activities represented by the places $p^1_i, i \in \{1, \ldots, 5\}$ while the activities of the second process (on the right part of the figure) are modeled by the places $p^2_i, i \in \{1, \ldots, 6\}$. Both processes are synchronized on transition $t^{12}_1$ whose firing leads to the marking of a shared place $p^{12}$ representing the activity that requires the two processes. The end of this activity is modeled by the firing of transition $t^{22}_1$ that leads to the marking of the places $p^3_3$ and $p^3_2$ so that each process can evolve independently to perform its own activities. The initial state of each process is given by the initial marking (places with black dots). Transition labels that are in bold represent observable events while the others are unobservable events. It must be noticed that Process 1 (resp. Process 2) only generates $\alpha_1$ (resp. $\alpha_2$) events as observations.

3.2 Event pattern model

Event patterns gather a set of specific combinations of untimed and unobservable events whose occurrence in the system leads to a situation of interest (i.e. failure, critical/dangerous situation, safe/unsafe behavior,...). Patterns that are studied throughout this paper are the same as in [17] and are defined as LPNs. As for the system model, transitions labels represent events of the system.

Definition 4 (Pattern). A pattern $\Omega$ is an LPN $\Omega = \langle P_\Omega, T_\Omega, A_\Omega, \Sigma_\Omega, M_{\Omega\Omega} \rangle$ associated with a set of final markings $Q_\Omega \subseteq R(\Omega, M_{\Omega\Omega})$ such that:

1. (unobservable) events of $\Sigma_\Omega$ are unobservable ($\Sigma_\Omega \subseteq \Sigma^U_\Theta$);
2. (initialized) $M_{\Omega\Omega} \not\in Q_\Omega$;
3. (well-formed) from any reachable marking $M$, there exists a run that leads to a marking $M' \in Q_\Omega$;
4. (deterministic) from any reachable marking $M$ of $R(\Omega, M_{\Omega\Omega})$ there is no event $e \in \Sigma_\Omega$ that labels more than one enabled transition;
5. (stable) from any reachable marking $M$ of $R(\Omega, M_{\Omega\Omega})$ such that $M \in Q_\Omega$ any possible run leads to a marking $M'$ such that $M' \in Q_\Omega$.

Patterns are LPN that aim at representing succinct behaviors that are unobservable (Condition 1) ($L(\Omega) \subseteq \Sigma^*_\Theta$). Condition 2 ensures that the pattern does not represent an empty event sequence. Condition 3 guarantees that any run of the pattern is always a prefix of a matching run. Condition 4 avoid ambiguities within the pattern and is required to ensure the correctness of the combination of a pattern and a system as detailed later. Finally Condition 5 ensures that once a pattern has occurred, its effect is definitive (any system’s run that has a matching run as a prefix is a matching run).

Figure 2 illustrates different types of patterns of interest for the system of Figure 1. Figure 2a presents $k$ occurrences of a single event $b$ (pattern $\Omega^b(k)$). The marking of the final place $pp^b_k$ indicates the pattern has occurred then the set of final markings is given by $Q_{\Omega^b(k)} = \{M : M(pp^b_k) = 1\}$. This pattern extends the classical single fault event pattern that is usually studied in the literature. Throughout this paper, we will similarly denote $\Omega^b(k)$ the pattern modeling $k$ occurrences of the single event $f$. Figure 2b presents a pattern that characterizes $k$ occurrences of one event among a predefined set of events, here among $\{b,f\}$. The structure of the pattern depends on the value of $k$ required. If one wants to consider one occurrence of one event among $\{b,f\}$, it is necessary to consider the marking of the place $pp^b_k$ and then $Q_{\Omega^b(k)} = \{M : M(pp^b_k) = 1\}$.

This pattern is used to model $k$ consecutive occurrences of event $b$ that are not interleaved with any occurrence of event $f$. This pattern can also be extended into $\Omega^b(k)$ to model $k$ consecutive occurrences of an event without any occurrence of another type of event. This last pattern is interesting as it represents some fairness issues in the system that might be of interest to detect, $Q_{\Omega^b(k)} = \{M : M(pp^b_k) = 1\}$.

3.3 Pattern diagnosis

This subsection introduces the problem of diagnosing patterns in timed discrete event systems. For a given pattern $\Omega$, the diagnosis problem consists in defining an $\Omega$-diagonser function that takes as input a timed sequence of observations and checks for any run of the system that can generate these observations whether it matches the pattern $\Omega$ or not. It results in the generation of three possible symbols: $\Omega$—faulty (all the possible runs match $\Omega$), $\Omega$—safe (none of them match $\Omega$), $\Omega$—ambiguous (some of them match $\Omega$, some of them do not) [15]. A run $r$ of the system matches a pattern $\Omega$ if one can find in a word $\rho = \theta_1 e_1 \ldots \theta_n e_n \theta_{n+1} \lambda$ of $L(\Omega)$ a sub-word $\left(\sum_{i=j_1}^{j_2} \theta_i\right) e_{j_1} \left(\sum_{i=j_{1+1}}^{j_2+1} \theta_i\right) e_{j_2} \ldots \left(\sum_{i=j_{k-1}}^{j_k} \theta_i\right) e_{j_k}$ such that $\rho_\Theta = e_{j_1} \ldots e_{j_k}$ is a word of $L(\Omega)$; this will be denoted by $\rho \supset \rho_\Theta$ and more generally by $\rho \supset \Omega$ and $r \supset \Omega$.

Checking whether a run of $\Theta$ can generate the given observations (i.e. the run is consistent with the observations)
relies on the timed language projection from the alphabet \( \Sigma_b \) to the observable alphabet \( \Sigma_o \) (see Section 2.1). The \( \Omega \)-diagnoser can then be formally defined as follows.

**Definition 5** (\( \Omega \)-diagnoser). An \( \Omega \)-diagnoser is a function

\[
\Delta_\Omega : T(\Sigma_o) \rightarrow \{\Omega-\text{faulty, } \Omega-\text{safe, } \Omega-\text{ambiguous}\}
\]

- \( \Delta_\Omega(\sigma) = \Omega-\text{faulty if for any } \rho \in \mathcal{L}(\Theta) \) that is consistent with \( \sigma \) (i.e. \( \mathcal{P}_{\Sigma_o} \rightarrow \mathcal{S}_o(\rho) = \sigma \), \( \rho \not\in \Omega \));
- \( \Delta_\Omega(\sigma) = \Omega-\text{safe if for any } \rho \in \mathcal{L}(\Theta) \) that is consistent with \( \sigma \), \( \rho \not\in \Omega \);
- \( \Delta_\Omega(\sigma) = \Omega-\text{ambiguous otherwise.} \)

It must be noticed that, even if Definition 5 uses Petri nets (\( \Theta \) and \( \Omega \) notations), this definition is purely language-based. The diagnosis problem is defined over the set of timed sequences \( \rho \) in \( \mathcal{L}(\Theta) \), the matching relation is defined over event sequences and consistency is achieved by sequence projection. It follows that Definition 5 defines a problem independently from the underlying model structure but with respect to their language expressiveness (time Petri nets, timed Event Graphs, timed automata...). Considering now a set of patterns \( \Omega_1, \ldots, \Omega_n \), the diagnoser function of a system can be defined as \( \Delta : T(\Sigma_o) \rightarrow \bigcup_{i=1}^n \{\Omega_1-\text{faulty, } \Omega_1-\text{safe, } \Omega_1-\text{ambiguous}\} \). Let us now illustrate the diagnosis problem with a few examples based on the system of Figure 1 and the family of patterns of Figure 2.

**Example 1.** Suppose that the timed sequence that is received by the diagnoser from the system of Figure 1 is \( \sigma_1 = 301_202_201_11 \) (that is the reception of event \( \sigma_1 \) at date 3 then the reception of \( \sigma_2 \) at date 2 and another reception of \( \sigma_2 \) at date 13 followed by \( \sigma_1 \) at date 15 and finally the observed sequence ends at date 16). The pattern \( \Omega^b_1(1) \) models the single occurrence of \( \sigma \). Based on \( \sigma_1 \), we know that the first occurrence of \( \sigma_2 \) is at the absolute date 5 which means that an occurrence of \( \sigma \) has definitely occurred. So \( \Delta_{\Omega^b_1(1)}(\sigma_1) = \Omega^b_1(1) \)-faulty. Considering now pattern \( \Omega^b_1(2) \) (i.e. two occurrences of \( \sigma \)). If \( \sigma \) has occurred twice, the second occurrence is after the second occurrence of \( \sigma_2 \) at date 13. As \( \sigma_1 \) lasts at date 16, it is possible that \( \sigma \) has occurred twice (the second occurrence would be at date in \([15, 16]\) but not necessary: \( \Delta_{\Omega^b_1(2)}(\sigma_1) = \Omega^b_1(2) \)-ambiguous).

The pattern \( \Omega^b_2(2) \) models two occurrences of an event among \( \{b, f\} \). Based on \( \sigma_1 \), we know that \( b \) has definitely occurred once. As the second occurrence of event \( \sigma_3 \) occurs at date 15, it enforces that transition \( t^3_3 \) has been fired once so the event \( f \) has certainly occurred and \( \Delta_{\Omega^b_2(2)}(\sigma_1) = \Omega^b_2(2) \)-faulty. Finally, \( \Delta_{\Omega^b_3(3)}(\sigma_1) = \Omega^b_3(3) \)-safe (event \( f \) has necessarily occurred once and \( b \) at most twice).

**4 Proposed method**

The proposed method relies on the formal framework proposed in [17] that is used for analyzing pattern diagnosability on the same type of systems. Figure 3 presents an overview of the proposed method.

The main principle of the method is to synthesize an LTPN, denoted \( \Pi_{\sigma, \Theta}^{(1)} \), that represents as a whole the set of runs of \( \Theta \) that are consistent with \( \sigma \) and for each run of these runs, the LTPN should also represent the fact that the pattern \( \Omega \) is matched or not. Once \( \Pi_{\sigma, \Theta}^{(1)} \) is obtained, the remaining step is to check into \( \Pi_{\sigma, \Theta}^{(1)} \) whether the runs match the pattern or not. To do so, we propose first to synthesize a set of formal queries and secondly to use a model-checker on \( \Pi_{\sigma, \Theta}^{(1)} \) to solve the queries and get the diagnosis (see Section 5 for details about this part). This section now presents how the LTPN \( \Pi_{\sigma, \Theta}^{(1)} \) is computed based on the LTPN \( \Theta \), the LPN \( \Omega \) and the timed sequence \( \sigma \). The computation is based on specific compositions of LTPNs that have been formally introduced in [17] and that are described in the next subsections.

**4.1 About composing LTPNs**

Usually, composition of Petri nets relies on the synchronization of some transitions. However, in time Petri nets, compositionality is kept only when synchronizations involve transitions with an interval \([0, +\infty)\) [19] and transitions with a time interval \([\alpha, \beta]\) cannot be synchronized as is. Indeed, synchronizing two transitions means that it is necessary to determine a clock enabled only when the transitions are both enabled and that stops as soon as one of the local clocks stops. This synchronization is then impossible by simply merging both transitions and assigning a time interval to it based on their respective time intervals as the stop of the synchronized clock depends on the enabling time of each transition and not on the enabling time of their synchronization. It is then required to perform a time decomposition of transition \( t \) to ensure that the synchronization is performed only on transitions with \([0, +\infty)\)
$\Omega_1^{(k)}(k): k$ occurrences of $b$ with $Q_{\Omega_1^{(k)}}(k) = \{M: M(p_{p1}) = 1\}$.

(b) $\Omega_2^{(k)}(k): k$ occurrences of $b$ or $f$ with $Q_{\Omega_2^{(k)}}(k) = \{M: M(pp_{p2}) = 1\}$.

(c) $\Omega_3(3), 3$ consecutive occurrences of $b$ without any occurrence of $f$, $Q_{\Omega_3(3)} = \{M: M(p_{p3}) = 1\}$.

Figure 2: A set of patterns.

Figure 3: Overview of the method for diagnosing an occurrence of $\Omega$ in the system $\Theta$ based on the observations $\sigma$.

Figure 4: Time decomposition of a transition $t$ of a LTPN $N$ with $\ell(t) = a$ and $I_a = [\alpha, \beta], \alpha > 0$.

4.2 Pattern matching product

The first step of the computation of $\Pi_{\Theta, \Omega}$ is to compute an LTPN that represents how a system $\Theta$ is actually matching a pattern $\Omega$: this step is based on the pattern matching product. This product relies on a specific product between the transitions of $\Theta$ and $\Omega$. Consider now $t$ as a transition of $\Theta$ (with $\ell_\Theta(t) = a$, $I_{\Theta(a)}(t) = [\alpha, \beta], \alpha > 0$)\footnote{Recall that the system has no zeno runs so we suppose that $\alpha > 0$.} with $\text{pre}(t) = \{p_1, \ldots, p_n\}$ and $\text{post}(t) = \{p_{1}', \ldots, p_{m}'\}$.

Without loss of generality, we assume that $t$ is conflict-free (no other transition has the same preset).\footnote{Indeed, if $t$ is in conflict with another transition $t'$, $\Theta$ can be transformed by adding two silent transitions $\lambda[0,0]$ enabled by the preset of $t$ and $t'$. Each silent transition has a postset with one new place that replaces the preset of either $t$ or $t'$. New versions of $t$ and $t'$ are then conflict-free while the generated language $L(\Theta)$ remains the same.} Let $t_\Omega$ be a transition of $\Omega$ labeled with $a$ with $\text{pre}(t_\Omega) = \{q_1, \ldots, q_m\}$ and $\text{post}(t_\Omega) = \{q_{1}', \ldots, q_{m}'\}$. To effectively obtain the pattern matching product between $t$ and $t_\Omega$, first a time decomposition of $t$ is performed in order to replace $t$ with a transition $e_t$ as detailed in the previous subsection. As $t_\Omega$ is part of an LPN and not a LTPN, it means that the implicit static time interval of $t_\Omega$ is $[0, +\infty]$ so no time decomposition is required for $t_\Omega$. The pattern matching product consists then in replacing $e_t$ and $t_\Omega$ by the set of transitions/places defined in Figure 5.

On the left, transition $e_t$, resulting from the time decomposition of $t$, is kept as is. On the right, the transition $t_\Omega$ is replaced by a synchronized transition $e_t|t_\Omega$ whose preset (resp. postset) is the union of the presets (resp. postsets) of $e_t$ and $t_\Omega$. Finally, a priority constraint is added between
Generally speaking, model-checking is a paradigm that gathers the set of techniques that aim at formally checking whether a property \( \varphi \) holds in a finite-state model \( M \):

\[
\mathcal{M} \models \varphi. \tag{2}
\]

A model-checker then returns a binary answer:

- either \( \varphi \) holds in \( M \) and the model-checker returns true;
- or \( \varphi \) does not hold in \( M \) and the model-checker returns false and a counter-example which proves the claim (a counter-example \( e \) is such that \( e \models M \land \neg \varphi \)).

We propose to solve the pattern diagnosis problem by using the LTPN model checker TINA [20; 21]. Consider a LTPN \( N \) and a property \( \varphi \) written as a SE-LTL formula (State/Event Linear Temporal Logic), TINA then solves the problem \( N \models \varphi \). SE-LTL is a temporal logic that is able to express patterns in a more natural way than LTL.

Note that the problem \( N \models \varphi \) is generally undecidable however TINA is able to determine sufficient conditions for decidable subclasses. In our specific case, the problem we solve are decidable by construction of \( \Pi^{\Theta,\Omega}_{\sigma} \) (safe LTPNs).
to express properties about runs of a LTPN. A formula $\psi$ is a SE-LTL formula if it is a universally quantified formula $\psi ::= \forall \varphi$ where $\varphi ::= \text{const} \lor r \lor \neg \varphi \lor \varphi \lor \varphi \lor \bigcirc \varphi \lor \diamond \varphi \lor \diamond \varphi \lor \varphi U \varphi$. The constant $\text{const}$ can be $\bot$ (false), $\top$ (true), dead (deadlock), div (temporal divergence), sub (partially known state). $r$ defines constraints $e \Delta < \Delta$ with $\Delta \in \{=,<,\leq,\geq\}$ between arithmetical expressions $e$ involving place and transition symbols from the underlying LTPN. The operators $\bigcirc$ (next), $\Diamond$ (always), $\Diamond$ (eventually) and U (until) have their usual LTL semantics.

Based on SE-LTL, it is in particular possible to implicitly characterize sets of markings in a given LTPN. For instance, suppose an LTPN $N$ composed of $n \geq 2$ places $p_1, p_2, \ldots, p_n$. The set $Q$ of markings $M \in Q$ of $N$ such that $M(p_1) = 1$ and $M(p_2) = 1$ can be characterized by the SE-LTL property denoted $\text{MARKINGS}(N, Q)$:

$$\text{MARKINGS}(N, Q) \equiv (p_1 = 1 \land p_2 = 1)$$

and the problem of checking whether in $N$, from its initial marking $M_0$, it is always true (\Box) that a run eventually (\Diamond) leads to a marking $M$ such that $M(p_1) = 1$ and $M(p_2) = 1$ can be formulated as follows:

$$N \models \Box \Diamond \text{MARKINGS}(N, Q)$$

### 5.2 Design of the diagnosis properties

To solve the diagnosis problem, two SE-LTL properties must be checked. The first one, denoted $\text{FAULTY}(\Theta, \Omega, \sigma)$, will formally express the property that every run of $\Theta$ that is consistent with the observation $\sigma$ matches the pattern $\Omega$. If the model-checker returns true to the problem $\Pi^\theta_\sigma(\Omega) \models \text{FAULTY}(\Theta, \Omega, \sigma)$ then it means that the $\Omega$-diagnoser must return:

$$\Delta_\Omega(\sigma) = \Omega - \text{faulty}. \ \ (3)$$

If the model-checker returns false, then it also returns as a counter-example a run consistent with $\sigma$ that does not match $\Omega$. Then two cases hold:

1. either every run that is consistent with $\sigma$, does not match $\Omega$ and the diagnosis is $\Omega$-safe
2. or there is another run consistent with $\sigma$ that matches $\Omega$ which means then that the diagnosis is $\Omega$-ambiguous.

To determine whether it is Case 1 or Case 2, a second model-checking problem needs to be solved. The second SE-LTL property to check is denoted $\text{SAFE}(\Theta, \Omega, \sigma)$ and expresses that every run of $\Theta$ that is consistent with the observation $\sigma$ does not match pattern $\Omega$. If finally the model-checker returns true then it is Case 1, otherwise it is Case 2. Note that, in case of an ambiguity, the method provides two runs, one is matching the pattern $\Omega$ and the other is not.

Both formulas $\text{FAULTY}(\Theta, \Omega, \sigma)$ and $\text{SAFE}(\Theta, \Omega, \sigma)$ express a property on runs of $\Theta$ that are represented as runs leading to admissible markings in $\Pi^\theta_\sigma(\Omega)$. A marking $M$ from $\Pi^\theta_\sigma(\Omega)$ is admissible as soon as for one place of type $Na\Theta$ (see Section 4.2), we have $M(Na\Theta) \neq 0$ or if $M(Na_{obs}) \neq 0$. To define the set of admissible markings of $\Pi^\theta_\sigma(\Omega)$ in SE-LTL:

$$\text{ADM}(\Pi^\theta_\sigma(\Omega)) \equiv \bigwedge_{Na\Theta \in \Pi^\theta_\sigma(\Omega)} Na\Theta = 0 \land Na_{obs} = 0.$$ 

The second property is the matching property. By construction of $\Pi^\theta_\sigma(\Omega)$ (see Proposition 1), a run matches $\Omega$ if it is admissible and leads to a marking which includes an accepting marking of $Q_\Omega$. $\text{MATCH}(\Pi^\theta_\sigma(\Omega))$ is the formula representing such markings:

$$\text{MATCH}(\Pi^\theta_\sigma(\Omega)) \equiv \text{ADM}(\Pi^\theta_\sigma(\Omega)) \land \text{MARKINGS}(\Omega, Q_\Omega).$$

Similarly, the property $\text{NO MATCH}(\Pi^\theta_\sigma(\Omega))$ asserts the admissible marking $M$ of $\Pi^\theta_\sigma(\Omega)$ is not a matching property, i.e. its restriction $M_{\Omega}$ does not belong to the final markings of $\Omega$.

$$\text{NO MATCH}(\Pi^\theta_\sigma(\Omega)) \equiv \text{ADM}(\Pi^\theta_\sigma(\Omega)) \land \neg \text{MARKINGS}(\Omega, Q_\Omega).$$

From Proposition 2, a run of $\Theta$ that is consistent with $\sigma$ is characterized by an admissible run of $\Pi^\theta_\sigma(\Omega)$ that leads to a marking that includes the marking of $p_{\sigma}^{ok}$.

$$\text{CONSISTENT}(\Pi^\theta_\sigma(\Omega)) \equiv \text{ADM}(\Pi^\theta_\sigma(\Omega)) \land \text{MARKINGS}(\Omega, \{p_{\sigma}^{ok}\}).$$

Finally, $\text{FAULTY}(\Theta, \Omega, \sigma)$ is defined as follows: it is always (\Box) true that if the run is consistent then it matches the pattern.

$$\text{FAULTY}(\Theta, \Omega, \sigma) \equiv \Box(\text{CONSISTENT}(\Pi^\theta_\sigma(\Omega)) \Rightarrow \text{MATCH}(\Pi^\theta_\sigma(\Omega))).$$

$$\text{SAFE}(\Theta, \Omega, \sigma)$$ is defined as follows: it is always (\Box) true that if the run is consistent then it does not match the pattern.

$$\text{SAFE}(\Theta, \Omega, \sigma) \equiv \Box(\text{CONSISTENT}(\Pi^\theta_\sigma(\Omega)) \Rightarrow \neg \text{MATCH}(\Pi^\theta_\sigma(\Omega))).$$

### 5.3 First experimental results

Table 1 presents the implementation results for the scenarios that are presented in Example 1. Each line presents some details about the LTPN $\Pi^\theta_\sigma(\Omega)$ that is generated (number of places, transitions, arcs, priorities). To check the properties $\text{FAULTY}$ and $\text{SAFE}$, the model-checker TINA first precomputes a Strong State Class Graph (SSG) that is then used as a Kripke structure. Table records the details (states, transitions) for the precomputed SSGs. The result for each query is also presented and shows that there are consistent with the expected result detailed in Example 1.

### 6 Conclusion

This paper proposes a method to solve the pattern diagnosis problem in timed discrete event systems. This method turns the problem to a couple of model-checking problems over a safe LTPN. The method only takes into account untimed patterns. One perspective would be to extend the method to also deal with timed patterns. Such an extension is not straightforward due to the specificities of the system-pattern product. We will firstly aim at better understanding the pattern matching product for timed patterns by characterizing it as a set of temporal constraints.

### References

Table 1: Results for the running example scenarios.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>$\Omega_1$</th>
<th>$\Omega_2$</th>
<th>$\Omega_3$</th>
<th>$\Omega_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>44 56</td>
<td>217 57</td>
<td>57</td>
<td>273 456 T</td>
</tr>
<tr>
<td>$S_2$</td>
<td>45 57</td>
<td>222 60</td>
<td>60</td>
<td>269 451 F</td>
</tr>
<tr>
<td>$S_3$</td>
<td>47 62</td>
<td>243 69</td>
<td>69</td>
<td>273 456 T</td>
</tr>
<tr>
<td>$S_4$</td>
<td>48 63</td>
<td>248 72</td>
<td>72</td>
<td>270 452 F</td>
</tr>
</tbody>
</table>

*pl* $\Pi^{[9,\Omega]}$, tr $\Pi^{[5,\Omega]}$, arcs $\Pi^{[4,\Omega]}$, prio $\Pi^{[8,\Omega]}$, st $\text{SSG}$, tr $\text{SSG}$, *FAULTY*, *SAFE*, *time*