A Short Overview on Diagnosability of Patterns in Timed Petri Net

Eric LUBAT

LAAS - CNRS

June 2020 / MOVEP
Introduction

- Context
- Behaviour of TPN
- Composability of TPN

Diagnosability in timed context

- State of the Art
- Product of TPN

Diagnosability of Patterns

TWINA

- Overview
- Quick example

Conclusion
Diagnosability is a basic property of Discrete Event Systems that relates to the “observability” of concealed events. Basically, it means that every failure (a distinct instance of unobservable event) can be eventually detected after a finite number of observations.
Diagnosability is a basic property of Discrete Event Systems that relates to the “observability” of concealed events. Basically, it means that every failure (a distinct instance of unobservable event) can be eventually detected after a finite number of observations.
Diagnosability is a basic property of Discrete Event Systems that relates to the “observability” of concealed events. Basically, it means that every failure (a distinct instance of unobservable event) can be eventually detected after a finite number of observations.
A simple (untimed) example
A simple (untimed) example
A simple (untimed) example

Possible executions:

\[ t_0 \ldots \| a \ldots \]
A simple (untimed) example

Possible executions:

$t_0 \ t_1 \ \ldots / \ a \ b \ \ldots$
A simple (untimed) example

Possible executions:

\[ t_0 t_1 t_0 t_1 \ldots / a b a b \ldots \]

\[ t_0 f t_3 t_0 t_1 \ldots / a b a b \ldots \]
A simple (untimed) example

Possible executions:

\[ t_0 \, t_1 \, t_0 \, t_1 \, \ldots \, / \, a \, b \, a \, b \, \ldots \]
\[ t_0 \, f \, t_3 \, t_0 \, t_1 \, \ldots \, / \, a \, b \, a \, b \, \ldots \]

Language: \((ab)^*\)
A simple (untimed) language

\[
\begin{align*}
[Sys] \cap [Sys|_f] & \approx [Sys \parallel Sys|_f] \\
\end{align*}
\]
Time Petri Nets

TPN are composed of:
- a Petri net
- an initial marking
- timing constraints $I_s(t)$ (aka static time interval) that constrains the time at which $t$ can fire.

Possible executions:

- $2.1 \ t_0 \ 2 \ t_1 \ \ldots \ / \ 2.1 \ a \ 2 \ b \ \ldots$
- $2.1 \ t_0 \ 1 \ f \ 3 \ t_3 \ \ldots / \ 2.1 \ a \ 4 \ b \ \ldots$

Language: $(ab)^*$
Analysing the “intersection” of TPN is hindered by two problems:

1. **State Space is infinite**

That is because in this case time delays $\delta$ can be arbitrarily small. $[x,y]$ timing constraint can be cut in an infinity $\delta$. 
Analysing the “intersection” of TPN is hindered by two problems:

1. State Space is infinite
2. Composability problem
1. Introduction
   - Context
   - Behaviour of TPN
   - Composability of TPN

2. Diagnosability in timed context
   - State of the Art
   - Product of TPN

3. Diagnosability of Patterns

4. TWINA
   - Overview
   - Quick example

5. Conclusion
A quick state of the art

Problem 1

The problem of State Space?

- A solution was proposed in [Berthomieu 83], where the authors define a state space abstraction based on State Class Graph (SCG).
- Basically, a state class captures a convex set of constraints on the time at which transitions can fire. This approach is used in several model-checking tools, such as Tina [Berthomieu 04].
Problem 2
Composition problem?

- Some works address the diagnosability of TPN [Lieu - 14, Wang - 15, Basile - 17].
- While they propose substantially different method, they all rely on a variation of the SCG construction of [Berthomieu 83].
Our approach: intersection of TPN languages

What we know  Computing intersection of TA is simple $\Rightarrow$ decidable on TPN (using $\text{TA} \supseteq \text{TPN}$ [Bérard et al. - 05]).

Problem: does not give an efficient method.

Idea: adapt State Class construction [Berthomieu - 83] to “product” of TPN
Idea: use a product, $N_1 \times N_2$, and force transitions with same label (e.g. $t_{3.1} \in N_1$ and $t_{1.2} \in N_2$) to fire “synchronously”.

Example
More intuitively, we can always interpret a PTPN as the set of transitions which must be fired together, at the same time. For exemple:

\[ \{ \{ t_0.1, t_0.2 \}, \{ t_1.1, t_1.2 \}, \{ t_3.1, t_1.2 \}, \{ f \} \} \]
PTPN: example of behaviour

Time elapse as in “classical” TPN ⇒ must fire $t_0$ before 1 initially.

Transitions $\{t_0, t_2\}$ and $\{t_1, t_3\}$ must fire simultaneously ⇒ this can create “timelocks”.

Eric LUBAT (LAAS - CNRS)
Using the features of PTPN, we have developed an algorithm to check the diagnosability of a TPN.

Its purpose is to detect cycles after (or with) a fault transition.
1 Introduction
   • Context
   • Behaviour of TPN
   • Composability of TPN

2 Diagnosability in timed context
   • State of the Art
   • Product of TPN

3 Diagnosability of Patterns

4 TWINA
   • Overview
   • Quick example

5 Conclusion
It is possible to extend this method to the observability of “patterns of events” [Gougam - 2017]. We define a pattern as a labelled Petri net, \( P \), representing the set of behaviours (sequences of labels) that we want to detect.

![Pattern Diagram](image)

**Figure:** pattern for “three consecutive b without f ”
We say a pattern is well-formed if it fulfils a set of hypothesis:

- Patterns are total.
- Patterns are deterministic.
- Labels $f$ are unobservable.
First, we make a PTPN with the TPN and its pattern:

Now we have to check for the "found" place!
Now, all we have to do is to test the "diagnosability" of the transitions just before "found". To do this... We simple apply another twin-plant method for a single fault!
Now the idea is to check the diagnosability via the PTPN.
We define the single fault as a pattern we are matching with, like this:

---

Theorem

A TPN $N$ is diagnosable if and only if all the maximal executions of the PTPN $N.1 \times N.2$ satisfy the LTL formula:

$$\Box(\text{found}.1 \Rightarrow \Diamond(\text{dead} \lor \text{found}.2))$$
Theorem

Assuming some well-formedness conditions on the pattern P, TPN N is diagnosable for pattern P iff all maximal executions of the PTPN \((N.1 \times P.1) \times (N.2 \times P.2)\) satisfy the LTL formula:

\[\Box (found.1 \Rightarrow \Diamond (\text{dead} \lor found.2))\]
1 Introduction
   - Context
   - Behaviour of TPN
   - Composability of TPN

2 Diagnosability in timed context
   - State of the Art
   - Product of TPN

3 Diagnosability of Patterns

4 TWINA
   - Overview
   - Quick example

5 Conclusion
• Tool for analyzing the “product” of two Time Petri Nets (TPN).
• Another use of Twina is to check the diagnosability of a net.
• Available at https://projects.laas.fr/twina/ with example.
Example
Example

PTPN NxN’

\[
\begin{align*}
\begin{array}{c}
\text{Example}
\end{array}
\end{align*}
\]
Example

TPN Languages

June 2020 / MOVEP
Example

TWINA

[1,2] \( t_{1.1} \) [2,4] \( t_{0.1} \) a

\( p_0 \)

[3,4] \( t_{3.1} \) b

\( p_1 \)

f

\( p_2 \)

elubat@cid:~/Dev/2TPN$ twina -diag --fault=f ex2tpn.net

# net is diagnosable
#
# states (explored): 3
# markings: 3
# dbms: 3
0.000s
Example

TPN Languages

June 2020 / MOVEP
Figure: Transport timed - [Gougam 17]
Figure: pattern for “three consecutive b without f”
Index

1 Introduction
   • Context
   • Behaviour of TPN
   • Composability of TPN

2 Diagnosability in timed context
   • State of the Art
   • Product of TPN

3 Diagnosability of Patterns

4 TWINA
   • Overview
   • Quick example

5 Conclusion
Motivation for our work

Fault diagnosis for Discrete Event System $\equiv$ properties on trace languages [Sampath - 95].

Addition of time constraints.

- $\Delta$-diagnosability $\equiv$ “reachability of (non-Zeno) runs in product of TA” [Tripakis - 02]

- $\tau$-diagnosability $\equiv$ “same” for T-TPN [Wang - 2015, Basile - 2017] (but ask for a firing sequence)

Problem boils down to checking the intersection of timed languages (Twin-plant method).
We talked about the diagnosability in DES with a timed context.

Our solution is available on https://projects.laas.fr/twina/.

The next step is to analyse opacity in TPN via the use of PTPN.
Conclusion

Thanks for your attention!

Any questions?
A quick state of the art

Overview of fault diagnosis methods based on petri net models

[Basile - 14]
System is diagnosable if we can always detect a fault in a finite number of observations.

A new approach for diagnosability analysis of petri nets using verifier nets

[Cabasino - 12]
There are several efficient methods for checking the diagnosability of single faults in PN. We can divide existing techniques in two groups:

   1. critical pair.
   2. “diagnoser-based” methods.
In [Basile - 17] the author build a Modified SCG that over-approximate the possible (timed) executions.

The system is diagnosable if no critical pair is found. If a critical pair is found, we have to solve a Linear Programming problems (LPP) to check whether this scenario is feasible.

This approach has several limitations. In particular, it may require to solve a large number of LPP.
Diagnoser-based approaches

- In [Liu - 14], the authors define an Augmented State Class (ASC) graph, which are SCG augmented with diagnosability information. He then use a method to split time intervals to only keep deterministic paths in the ASC graph.
  - The interval splitting phase may create a large number of new active states that can lead to a state explosion problem.

- The approach in [Wang - 15] relies on a combination of SCG and an enumeration of all the firing sequences between active states.