Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 51-7 (2018) 316–322

# PetriBaR: A MATLAB Toolbox for Petri Nets Implementing Basis Reachability Approaches

**Siqi Liu** *, **Yin Tong** *,1, **Carla Seatzu** **, **Alessandro Giua** ***

* *School of Information Science and Technology, Southwest Jiaotong University, Chengdu 611756, China (e-mail: bk20122485@my.swjtu.edu.cn; yintong@swjtu.edu.cn).*

** *Department of Electrical and Electronic Engineering, University of Cagliari, 09123 Cagliari, Italy (e-mail: seatzu@diee.unica.it)*

*** *DIEE, University of Cagliari, Cagliari 09124, Italy, Aix-Marseille Univ, Université de Toulon, CNRS, ENSAM, LSIS, Marseille 13397, France (e-mail: giua@diee.unica.it)*

**Abstract:** This paper presents a MATLAB toolbox, called *PetriBaR*, for the analysis and control of Petri nets. PetriBaR is a package of functions devoted to basic Petri net analysis (including the computation of T-invariants, siphons, reachability graph, etc.), monitor design, reachability analysis, state estimation, fault diagnosis, and opacity verification. In particular, the functions for reachability analysis, state estimation, fault diagnosis, and opacity verification exploit the construction of the *Basis Reachability Graph* to avoid the exhaustive enumeration of the reachable set, thus leading to significant advantages in terms of computational complexity. All functions of PetriBaR are introduced in detail clarifying the syntax to be used to run them. Finally, they are illustrated via a series of numerical examples. PetriBaR is available online for public access.



*Keywords:* Petri nets, MATLAB toolbox.

## 1. INTRODUCTION

Petri nets (PNs) are a powerful discrete event model. The analysis of PNs basically concerns the study of some behavioral properties including reachability, boundedness, liveness, reversibility, repetitiveness, etc. and some structural properties related to the presence of P-invariants, T-invariants, siphons, traps, etc. (Murata, 1989).

Concerning the control of PNs under static specifications, one of the most popular approaches is based on the notion of *generalized mutual exclusion constraints* (GMECs). In such a case the set of legal markings is defined as a set of linear inequalities and this benefits from the main feature of PNs. In particular, Giua (Giua, 1992) demonstrated that, in the case of controllable and observable transitions, such constraints could be enforced simply adding some places called *monitors*, also guaranteeing maximal permissiveness. In (Moody and Antsaklis, 2000) such a theory has also been extended to the case where some transitions are uncontrollable and/or unobservable.

When dealing with problems of state estimation, fault diagnosis, opacity verification, etc. the system is typically modeled using *labeled Petri nets* (LPN) to describe the fact that certain transitions may produce no observation when they fire (silent transitions) or may produce the same observation of other transitions (indistinguishable transitions). To avoid exhaustively enumerating all reachable markings and firing vectors, the notions of basis marking and minimal explanation vector have been introduced in (Cabasino et al., 2010) to solve the problem of fault diagnosis. In (Cabasino et al., 2010) the authors also show that in the case of bounded nets, such notions allow one to describe the system's behaviour without enumerating all the reachable markings (as in the reachability graph) and introduce the *Basis Reachability Graph* (BRG). The BRG has also been efficiently used to solve a series of other problems, in particular, reachability analysis, state estimation and opacity verification (Ma et al., 2017; Tong et al., 2016, 2017).

The goal of this paper is that of illustrating a MATLAB tool implementing all the above mentioned approaches. We notice that there are many other tools for PN analysis and simulation, such as Integrated Net Analyzer (INA) (INA, 2003), TAPAAL (TAPAAL, 2017), PN Tool (Pastravanu et al., 2004), etc. While all these tools can be used to study basic properties and simulating the behavior of purely logical PNs, PNs with time, and also high level PNs, few of them can handle the problems of states estimation, fault diagnosis, and opacity verification. Furthermore, all the above tools offer a user-friendly graphical interface, which provides users an easy access but makes it hard to modify the codes and to extend them for other purposes. On the contrary, the PN toolbox *PetriBaR* we present consists of several MATLAB functions and is meant as support for research activities and classroom problem solving. Its functions can be classified into six categories:(1) basic analysis,

[1] Corresponding Author


10.1016/j.ifacol.2018.06.319

e.g., boundedness analysis, siphons computation, etc.; (2) monitor design for GMECs; (3) reachability analysis; (4) state estimation; (5) fault diagnosis, including diagnosability analysis; (6) state-based opacity verification. Functions in categories 3-6 exploit the approach based on BRG, therefore they are able to handle some relatively large-sized nets. PetriBaR can be freely downloaded from a web site (PetriBaR, 2017).

Summarizing, the main features of the tool.

• Besides structural and behavioral properties analysis, PetriBaR can also solve control problems and problems of reachability analysis, state estimation, fault diagnosis, and opacity verification taking advantage of the BRG approach.

• The MATLAB environment has been widely used to implement software tools for the control of continuous-time systems. In addition there exists also a few other Petri net tools based on MATLAB (e.g., Pastravanu et al. (2004)). A tool, called HYPENS (Sessego et al., 2008), has also been implemented in MATLAB by some of the authors of this paper to simulate and analyze First-Order Hybrid Petri nets (Balduzzi et al., 2000). The tool illustrated in this paper allows one to fill the gap of the MATLAB implementation of purely logic PNs and may lead to the MATLAB solutions to problems of state estimation, fault diagnosis, etc. in hybrid Petri nets.

• In most universities, MATLAB is taught in basic courses for engineering students. Thus, it is easy for students to get started. Moreover, PetriBaR is open source and built in a modular way. Once the user becomes familiar with the implemented function, he/she can easily modify and extend them depending on his/her own requirement. Finally, MATLAB runs on many platforms (Windows, Unix, MacOS).

## 2. BASIC PETRI NET ANALYSIS FUNCTIONS

A *Petri net system* (PN system) is a pair $(N, M_0)$, where $N = (P, T, Pre, Post)$ is the net structure, $P = \{p_1, p_2, \ldots, p_m\}$ is the set of *places*, $T = \{t_1, t_2, \ldots, t_n\}$ is a set of *transitions*, $Pre : P \times T \to \mathbb{N}$ and $Post : P \times T \to \mathbb{N}$ are the *pre-* and *post-incidence functions* that specify the arcs directed from places to transitions and from transitions to places, respectively, $M_0 \in \mathbb{N}^m$ is the initial marking. The pre- and post-incidence functions are usually described by $m \times n$ dimensional pre- and post-incidence matrices, and clearly they uniquely determine the structure of the PN. Thus, to input a PN system to a function of the toolbox, users need to give the pre- and post-incidence matrices and a column vector that represents the initial marking. PetriBaR contains a function, `draw`, to assist the user in inputting the PN system and obtaining the corresponding matrices. Running function `draw`, a graphical interface appears (see Fig. 1), and the user can input the PN system by clicking "Place"/"Transition"/"Arc" and drawing them in the blank area. By double clicking a place (or an arc), the user can edit the number of tokens in the place (or the weight of the arc). Finally, choosing "Save Data", the value of the pre- and post-incidence matrices, and the initial marking are written in file `getMatrix`.

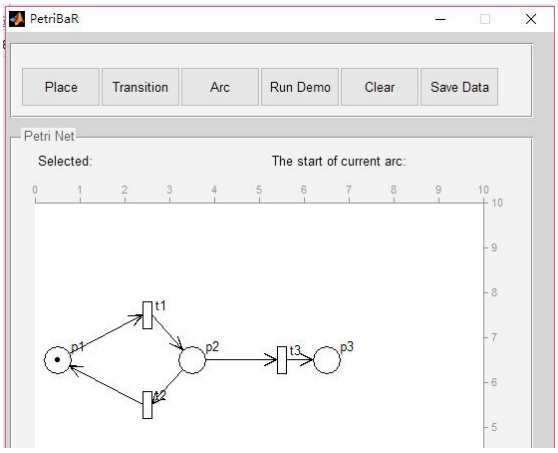


Fig. 1. GUI for inputting the PN system.

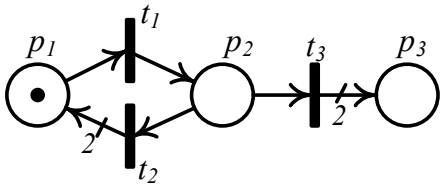


Fig. 2. The PN system in Exmaple 1.

A series of functions for basic PN structural and behavioral analysis, including the construction of the reachability/coverability graph/tree (RG/CG or RT/CT), are contained in the directory *PN_BASIC* of the toolbox. The most significant ones are listed in Table 1.

For sake of brevity, and since our focus here is on the notion of basis reachability graph, in the following example we restrict our attention on the function `graphPN` that allows us to construct the RG of a given bounded PN system.

*Example 1.* Input the PN system in Fig. 2 through function `draw`. Function `getMatrix` returns matrices *Pre*, *Post*, and $M0$. We check if the PN system is bounded using function `bounded`. The output of the function is $b = 0$: the net is unbounded (see Fig. 3). Then we input command `G=graphPN(Pre,Post,M0)` to construct its CG in the form of a matrix, as Fig. 4 shows. Let us now briefly explain the data structure of $G$.

Matrix $G$ is partitioned into several submatrices:
• $G_{11} = G(1 : 6, 1 : 3)$: each row of $G_{11}$ represents a marking (transposed) in the CG, and the index of each marking is their corresponding row number [2], i.e., $G_{11}(i, \cdot)^T = M_i$;
• $G_{12} = G(1 : 6, 4)$: the $i$-th element $G_{12}(i)$ of $G_{12}$ denotes a father node of $M_i$;
• $G_{13} = G(1 : 6, 5)$: $G_{13}(i) = -2$ (resp., -1) means $M_i$ is not a dead marking (resp., is a dead marking);
• $G_{14}$, $G_{15}$ and $G_{16}$ (every two columns after 5-th column and rows 1 to 6 compose a submatrix): each row of $G_{14}$, $G_{15}$ and $G_{16}$ describes a transition. For instance, the firing of transition $G_{14}(i, 1)$ at $M_i$ leads to marking $M_j$, where $j = G_{14}(i, 2)$;
• $G_{21} = G(7, 1 : 3)$: the first element of $G_{21}$ denotes the number of places, the second one $G_{21}(2)$ is the number of

[2] The initial marking is at the first row, i.e., corresponds to $M_1$.

Table 1. Main Functions in *PN_BASIC*

| Name | Function | Name | Function |
|---|---|---|---|
| `graphPN` | builds the RG/CG of a PN system | `tree` | builds the RT/CT of a PN system |
| `show` | shows the RG/CG in a readable text form | `plottree` | draws the graph of a RT/CT |
| `play` | simulates the evolution of a PN system | | |
| `bounded` | checks if a PN system and its places are bounded | `live` | checks if a PN system and its transitions are live |
| `dead` | checks deadlock markings in a PN system | `reachable` | checks if a marking is reachable from $M_0$ |
| `firable` | checks if a sequence of transitions is enabled at $M_0$ | `reversible` | checks if a PN system is reversible |
| `pinvar` | computes P-invariants (or P-semiflows) | `tinvar` | computes T-invariants (or T-semiflows) |
| `siphons` | computes siphons | `traps` | computes traps |

```
>> draw; [Pre, Post, M0] = getMatrix;
>> b = bounded(Pre,Post,M0)
M =
           Inf     0     0
****The net is unbounded!****
b =
           0
```

Fig. 3. The PN system in Fig. 2 is bounded.

```
>> G = graphPN(Pre,Post,M0)
G =
        G11          G12   G13   G14   G15   G16
  1     0     0      0    -2    1  2  0  0  0  0
  0     1     0      1    -2    2  3  3  4  0  0
 Inf    0     0      2    -2    1  5  0  0  0  0
  0     0     2      2    -1    0  0  0  0  0  0
 Inf   Inf    0      3    -2    1  5  2  5  3  6
 Inf   Inf   Inf     5    -2    1  6  2  6  3  6
G21 3   3     1      0     0    0  0  0  0  0  0
```

Fig. 4. RG (in matrix form) of the PN system in Fig. 2.

transitions, and $G_{21}(3) = 1$ means the matrix represents the CG. ◇

The size of $G$ depends on the number of markings in the RG/CG and the number of transitions. The functions using the RG/CG to analyze the behavioral properties are listed in Table 1. In the case of unbounded nets the CG gives necessary and sufficient conditions for boundness. However, for the absence of dead states, reversibility, liveness, reachability it only provides necessary conditions.

## 3. MONITOR DESIGN

A set of *generalized mutual exclusion constraints* (GMECs) $(W, K)$, with $W = [w_1 \dots w_r]$ and $K = [k_1; \dots; k_r]$, defines a set legal markings

$$\mathcal{M}(W,K) = \bigcap_{i=1}^{r} \{M \in \mathbb{Z}^m | w_i^T \cdot M \leq k_i\} = \{M \in \mathbb{Z}^m | W^T \cdot M \leq K\},$$

where $w_i \in \mathbb{Z}^m$ and $k_i \in \mathbb{Z}$ (with $i = 1, 2, \dots, r$). As shown in (Giua, 1992; Moody and Antsaklis, 2000), a set of $r$ GMECs can be enforced adding $r$ places, called monitors, with appropriate initial marking, to the given net system. In the presence of uncontrollable transitions, if the given GMECs $(W, K)$ are uncontrollable[3], a set of *controllable* GMECs $(W', K')$ may be computed such that the reachable marking set of the obtained system is contained in the set of legal markings $\mathcal{M}(W, K)$.

[3] A set of GMECs is uncontrollable if there is a pre arc from a monitor place to uncontrollable transitions.

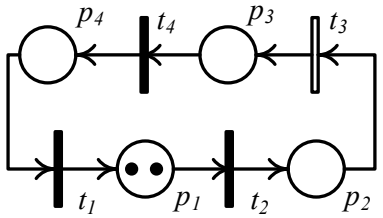


Fig. 5. The PN system in Exmaple 2, where $t_3$ is uncontrollable.

```
>> draw; [Pre, Post, M0] = getMatrix; C = Post-Pre;
w1 = [1 0 0 2]'; w2 = [0 0 2 1]';
k1 = 2; k2 = 2; u = [3];
W = [w1, w2]; K = [k1; k2];
[W2c, K2c] = controllablegmec(C, M0, w2, k2, u);
W2 = [w1,W2c']; K2 = [k1;K2c];
>> [CS, MS0] = monitorplaces(C, M0, W2, K2, u)
CS =
     1    1    0   -2
     1   -2    0    1
MS0 =
     0
     2
```

Fig. 6. Computation of monitor places enforcing a given set of GMECs.

Functions in directory *PN_MONIT* solve the monitor design problem. The most significant ones are listed in Table 2. The following example shows how to use them to compute monitor places enforcing a given set of GMECs.

Table 2. Main Functions in *PN_MONIT*

| Name | Function |
|---|---|
| `checkgmecs` | tests if a set of GEMCs is controllable wrt a PN system |
| `controllablegmec` | finds all minimally restrictive controllable GMECs satisfying the given uncontrollable GMEC |
| `monitordesign` | designs a closed loop net system given a GMEC |
| `monitorplaces` | finds the row of the incidence matrix of the monitor place used to satisfy the set of controllable GMECs |

*Example 2.* Consider the PN system in Fig. 5, where $t_3$ is uncontrollable. Let $(W = [w1, w2]$, $K = [k_1; k_2])$ be the set of GMECs, where $w_1 = [1\ 0\ 0\ 2]^T$, $k_1 = 2$, $w_2 = [0\ 0\ 2\ 1]^T$, $k_2 = 2$. Inputting the incidence matrix $C = [1\ -1\ 0\ 0;\ 0\ 1\ -1\ 0;\ 0\ 0\ 1\ -1;\ -1\ 0\ 0\ 1]$, the initial marking $M_0 = [2\ 0\ 0\ 0]^T$, the set of GMECs $(W, K)$, and the uncontrollable transitions $u = [3]$ to function `[Wc,Kc,Wu,Ku]=checkgmecs(C,M0,W,K,u)`, we obtain that $W_c = w_1$, $K_c = k_1$, $W_u = w_2$ and $K_u = k_2$,

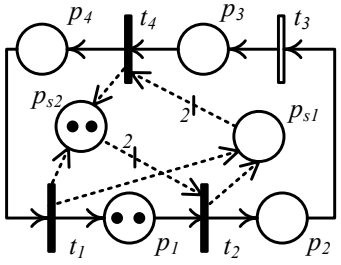


Fig. 7. Closed loop system in Example 2.

i.e., $(w_1, k_1)$ is controllable while $(w_2, k_2)$ is not controllable. Use function `[W2, K2] = controllablegmec(C, M0, w2, k2, u)` to compute a set of controllable GMECs $(W_2, K_2)$ such that $(W_2, K_2)$ is minimally restrictive and satisfies $\mathcal{M}(W_2, K_2) \subseteq \mathcal{M}(w_2, k_2)$, and the result is $W_2 = [0\ 2\ 2\ 1]^T$ and $K_2 = 2$. Finally, we use function `monitorplaces` to compute the incidence matrix $CS$ of monitor places and their initial marking $M_{S0}$. The obtained results and the whole procedure are illustrated in Fig. 6. The corresponding closed loop system is illustrated in Fig. 7. Note that given a set of controllable/uncontrollable GMECs, function `monitordesign` returns the closed loop system such that its reachable markings are all legal but may not be maximally permissive [4]. ◇

## 4. FUNCTIONS USING BRG-BASED TECHNIQUES

In (Ma et al., 2017), a compact representation of the reachability graph (RG) is proposed. Partitioning the set of transitions into implicit and explicit transitions, it is shown that if the implicit transitions induced subnet is acyclic, only a subset of reachable markings, called *basis markings*, need to be enumerated, while other reachable markings are characterized by linear systems, one for each basis marking.

To help the reader in understanding the idea behind the programs, we briefly recall the notion of basis marking. Let $(P, T, Pre, Post, M_0)$ be the PN system, $T_e \subseteq T$ the set of *explicit* transitions, and $T_i = T \setminus T_e$ the set of *implicit* transitions. First, the initial marking $M_0$ is a basis marking. Then, given an explicit transition $t_e$ if there is a *minimal* (in terms of its firing vector) firable sequence of implicit transitions $\sigma_i \in T_i^*$ enabling $t_e$, then the marking $M$ reached by firing $\sigma_i t_e$ from $M_0$ is also a basis marking. Iteratively, the set $\mathcal{M}_b$ of basis markings is obtained. Obviously, the set of basis markings is a subset of the reachability set since basis markings are only markings reachable by firing the minimal implicit transition sequences and an explicit transition. Finally, we have the following important result.

*Theorem 3.* (Ma et al., 2017) Let $(P, T, Pre, Post, M_0)$ be a PN system, $T_e$ a set of explicit transitions, and $\mathcal{M}_b$ a set of basis markings wrt $T_e$. Assume the $T_i$ induced subnet [5] is acyclic. A marking $M \in \mathbb{N}^m$ is reachable in the PN system if and only if there exists a basis marking $M_b \in \mathcal{M}_b$ such that $M = M_b + C_I \cdot y$ has an integer solution $y \in \mathbb{N}^{n_I}$,

[4] If there is no solution, the function will output "This method can not be used".

[5] The $T_i$-induced subnet of $(P, T, Pre, Post)$ is the net obtained by removing from $(P, T, Pre, Post)$ all transitions not in $T_i$.

Table 3. Main Functions in *PN_REACH*

| Name | Function |
|---|---|
| `BRG` | computes the corresponding BRG of the net system wrt a given feasible $T_e$ |
| `showBRG` | shows the BRG in the form of text |
| `IfReachTe` | checks if a marking is reachable, given a feasible $T_e$ |
| `CompTe` | computes a minimal feasible set $T_e$ |
| `IfReach` | checks if a marking is reachable and outputs a minimal feasible set $T_e$ |

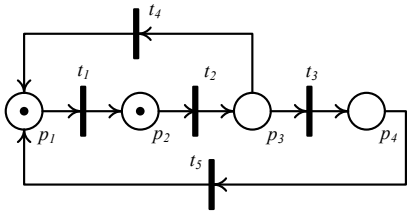


Fig. 8. The PN system in Example 4.

```
>> draw; [Pre, Post, M0] = getMatrix; Te = [1 4 5];
>> [B] = BRG(Pre,Post,M0,Te)
B =
    [1] [4×1 double] [1×3 double] {1×3 cell} [1] {1×3 cell}
    [2] [4×1 double] [1×3 double] {1×3 cell} [1] {1×3 cell}
    [3] [4×1 double] [1×3 double] {1×3 cell} [1] {1×3 cell}
```

Fig. 9. Construction of the BRG of the PN system in Fig. 8.

where $m$ is the number of places, $C_I$ is the incidence matrix of the implicit transitions induced subnet, and $n_I$ is the number of implicit transitions.

In other words, after the set of basis markings are computed, the reachability problem reduces to the solution of the integer linear programming problem, and the union of convex sets associated with the different basis markings coincides with the set of reachable markings. A *basis reachability graph* (BRG) is a graph that describes the basis markings and their transition relations, and thus it compactly represents the RG. Based on the BRG representation, algorithms very efficient in practice have been proposed in (Cabasino et al., 2010; Ma et al., 2017; Tong et al., 2017; Cabasino et al., 2011) to solve the reachability problem, the fault diagnosis problem and the opacity verification problem. The efficiency of the BRG-based approaches with respect to other RG-based methods has been extensively shown in the aforementioned work.

Typically $T_e$ is not a set of special transitions. As long as the set of transitions can be partitioned into $T_e$ and $T_i$ such that the $T_i$-induced subnet is acyclic (such a set $T_e$ is called *feasible*), Theorem 3 applies. Thus, in some cases a feasible set $T_e$ needs to be computed first so that BRG-based techniques can be applied to analyzing the system.

### 4.1 Reachability Analysis

Functions for reachability analysis are included in directory *PN_REACH* and listed in Table 3. All functions are coded using the results in (Ma et al., 2017), where the reader can also find the technical details.

```
>> showBRG(B)
    Basis Reachability Graph node's number n = 3
    # Marking M0=[1 1 0 0]'
    Observable transitions enabled to fire:
        (t1) -> M1: e=[0 0]
        (t4) -> M2: e=[1 0]
        (t5) -> M2: e=[1 1]
          *************
    # Marking M1=[0 2 0 0]'
    Observable transitions enabled to fire:
        (t4) -> M0: e=[1 0]
        (t5) -> M0: e=[1 1]
          *************
    # Marking M2=[2 0 0 0]'
    Observable transitions enabled to fire:
        (t1) -> M0: e=[0 0]
          *************
```

Fig. 10. Output of function `showBRG`.

```
>> Te=CompTe(Pre,Post)
Te =
     1
```

Fig. 11. Set of explicit transitions computed by function `CompTe`.

*Example 4.* Consider the LPN system in Fig. 8. Let $T_e = \{t_1, t_4, t_5\}$ be a feasible set of explicit transitions. First, we input the net system $(Pre, Post, M0)$ and the set $T_e = [1\ 4\ 5]$, and then compute its BRG through function `B=BRG(Pre,Post,M0,Te)` (see Fig. 9). Let us now briefly explain the data structure[6] of $B$. Entries in columns 1 to 6 are: the index associated with the current basis markings, the basis marking, explicit transitions whose minimal explanation vectors are not empty, the minimal explanation vector and the index of the basis marking reached, the tag recording if the basis marking has been analyzed, the explicit transition and the corresponding minimal explanations fired. Therefore, $B$ contains all the information about the BRG, and in the future the reader can develop their own functions based on function `BRG` as Sections 4.2 to 4.4 show. The structure of the BRG can be presented in the form of text, as Fig. 10 shows, by using function `showBRG(B)`.

Given an arbitrary marking $M = [0\ 1\ 0\ 1]^T$, the result of running `r=IfReachTe(M,Pre,Post,M0,Te)` is $r = 1$, i.e., marking $M$ is reachable from the given initial marking. On the contrary, if $M = [1\ 1\ 0\ 1]^T$, the output is $r = 0$, which means that the marking is not reachable.

Suppose the feasible set $T_e$ of explicit transitions is not given. In this case, there are two ways to check if a marking is reachable. One method consists in first using `Te=CompTe(Pre,Post)` to obtain a minimal feasible set $T_e$ (the result is $T_e = \{t_1\}$), then using function `IfReachTe` to check whether the given marking is reachable or not. Note that the reachability of a marking does not depend on the value of $T_e$. The other method consists in using function `IfReach` directly, as shown in Fig. 12. ◇

[6] We point out that the data structure of the BRG constructed later for LPNs, or fault diagnosis, or opacity verification is similar to the one presented here. For brevity, a detailed discussion on differences is omitted here and the reader is addressed to the comments inside the MATLAB functions.

```
>> M = [0 1 0 1]'; [r, Te] = IfReach(M, Pre, Post, M0)
r =
     1
Te =
     1
```

Fig. 12. Output of function `IfReach`.

### *4.2 State Estimation*

In an LPN system $(Pre, Post, M_0, L, E)$ the labeling function $L$ associates an output symbol with each transition. It could either be a symbol in a given alphabet $E$, or the empty word $\varepsilon$. Therefore, the transitions are partitioned into the set $T_o$ of observable transitions, whose labels are symbols in $E$, and the set $T_u$ of unobservable transitions, whose labels are the empty word.

The considered state estimation problem consists in determining the set of current markings consistent with the observation $w \in E^*$. Furthermore, the initial state estimation problem consists in determining the set of possible initial markings from which the observation $w \in E^*$ can be generated. Typically, the methods of solving the problems of current state estimation and initial state estimation are based on the construction of the observer and the initial state estimator, respectively, of the RG, whose complexity is known to be exponential wrt the number of reachable markings. However, if the unobservable transitions induced subnet is acyclic, the BRG-based technique can be applied (Cabasino et al., 2011). In such a case the complexity is still exponential but wrt the number of basis markings. To build the BRG for LPNs, observable transitions are taken as explicit transitions, while the unobservable transitions are the implicit transitions. Moreover, the label of an observable transition is also tagged on the BRG.

Let $\mathcal{C}(w)$ be the set of markings consistent with an observation $w$ and $\mathcal{C}_b(w)$ the set of basis markings consistent with $w$. By Theorem 3.7 in (Tong et al., 2017), it holds that $\mathcal{C}(w) = \bigcup_{M_b \in \mathcal{C}_b(w)} \{M | M = M_b + C_u \cdot y, y \in \mathbb{N}^m\}$, where $C_u$ is the incidence matrix of the unobservable transitions induced subnet. Namely, using the BRG-based technique, the set $\mathcal{C}(w)$ of consistent markings is represented by a set of linear constraints associated with basis markings. Therefore, computing the set $\mathcal{C}_b(w)$ of basis markings consistent with the observation, the set $\mathcal{C}(w)$ is also obtained.

Functions for state estimation are in directory *PN_ESTM* and are summarized in Table 4. Due to limited space, in Example 5 we mainly illustrate the methods based on the BRG.

*Example 5.* Consider again the PN system in Fig. 8. Let $M_0 = [1\ 0\ 0\ 0]^T$, $T_u = \{t_2, t_4, t_5\}$, the labels assigned to $t_1$ and $t_3$ be $a$ and $b$, respectively, and the observation $w = aab$. For current state estimation, we can use function `CurEst_BRG` to compute $\mathcal{C}_b(w)$ directly. The procedure and the result is presented[7] in Fig. 13,

[7] Since the output *Cbw* is defined as a cell, the value of *Cbw* is not directly shown in the command window.

Table 4. Main Functions in *PN_ESTM*

| Name | Function |
|---|---|
| `BRG_L` | builds the corresponding BRG of the LPN system |
| `showBRG_L` | shows the BRG in a readable text form |
| `obsRG` | computes the observer of the RG |
| `obsBRG` | computes the observer of the BRG |
| `estRG` | computes the initial state estimator of the RG |
| `estBRG` | computes the initial state estimator of the BRG |
| `CurEst_RG` | computes the set of markings consistent with the observation |
| `CurEst_BRG` | computes the set of basis markings consistent with the observation |
| `IniEst_RG` | computes the set of markings generating the observation |
| `IniEst_BRG` | computes the set of basis markings generating the observation |

```
>> draw; [Pre, Post, M0] = getMatrix; E = {['a'];['b']}; L =
{[1];[3]};
w = 'aab';
>> [Cbw] = CurEst_BRG(Pre,Post,M0,L,E,w)
Cbw =
      [4×1 double]
```

Fig. 13. Computation of the set of basis markings consistent with *aab*.

```
>> B = BRG_L(Pre, Post, M0, L, E);
>> [ObsB, yb] = ObsBRG(B)
ObsB =
     [3]   '12'   [4×3 double]   [0]   [ ]
yb =
     [0]   [1]   [2]
```

Fig. 14. Observer of the BRG in Example 5.

where $\mathcal{C}_b(w) = \{[0\ 0\ 0\ 1]^T\}$. Analogously, for initial state estimation, we can use function `IniEst_BRG` to compute the set of basis markings generating *aab*, namely $\{[1\ 0\ 0\ 0]^T, [0\ 1\ 0\ 0]^T, [0\ 0\ 0\ 1]^T\}$.

Let us focus on the data structure of the observer obtained by function `obsBRG` (which could be helpful to readers interested in building their own functions). After constructing the BRG $B$ through function `BRG_L`, we input `[ObsB,yb]=ObsBRG(B)` to compute the observer of the BRG (see Fig. 14). *ObsB* is a $1\times 5$ cell, and columns 1 to 5 represent the number of states [8] of the observer, the set of events [9], the transition function [10], the set of initial states, and the set of marked states, respectively. Each element of $y_b$ corresponds to the set of basis markings of a state of the observer. We point out that the estimator of the BRG, and the observer/estimator of the RG obtained by functions in Table 4 have a similar structure of *ObsBRG*. Therefore, they are not discussed here. ◇

[8] It also denotes the set of states of the observer. Suppose the number of states is $n$. Then the set of states is $\{0, 1, \ldots, n-1\}$.

[9] Alphabetical symbols input to the LPN are orderly projected to numerical symbols in the observer. For instance, $E = \{['a']; ['b']\}$ of the LPN becomes $E =' 12'$.

[10] Suppose one row of the matrix is $[x_i\ j\ x_k]$. It means that from state $x_i$ the occurrence of event $k$ leads to state $x_k$.

Table 5. Main Functions in *PN_DIAG*

| Name | Function |
|---|---|
| `BRG_D` | constructs the BRG for fault diagnosis wrt the class of faults |
| `showBRG_D` | shows the BRG in a readable text form |
| `MBRG` | constructs the modified BRG (MBRG) that is used to check whether the LPN system is diagnosable or not. |
| `showMBRG` | shows the BRG in a readable text form |
| `BRD` | constructs the basis reachability diagnoser, a diagnoser construted on the BRG. |
| `showBRD` | shows the BRG in a readable text form |
| `diagnosability` | tests if a bounded LPN system is diagnosable wrt each fault class |
| `diagnosis` | online diagnoses an LPN system for a given observation $w$ |

```
>> [Pre, Post, M0] = getMatrix; F = {[3]; [5]}; L = {[1]}; E =
{['a']};
BG = BRG_D(Pre, Pos, M0, F, L, E);
BD = BRD(BG, Pre,Post, M0, F, L, E);
T = MBRG(Pre, Pos, M0, F, L, E);
>> diagnosability(T, BD)
   All fault classes are not diagnosable since for each of them there
is an indeterminate loop.
Fault class 1:
            path = aa
            cycle = a
Fault class 2:
            path = aa
            cycle = a
```

Fig. 15. Diagnosability test in Example 6

```
>> w = 'a'; diagnosis(Pre, Post, M0, w, L ,F, E)
The following results summarizes the step of the on-line diagnosis
carried out by the observed word: - 'a'.
      SUFFIX        DIAGNOSIS STATE

        eps              0   0
         a               2   2
```

Fig. 16. Online diagnosis of observation $w = a$.

### *4.3 Fault Diagnosis*

Given an LPN system and a set of fault transitions, which have been partitioned into several fault classes $F = \{f_1, f_2, \ldots, f_r\}$, two main problems are investigated in the literature: i) fault diagnosis, which consists in establishing, given an observation, if a fault in a given class has occurred, and ii) diagnosability analysis, which consists in establishing if the occurrence of a fault in a given class could be detected after the occurrence of a finite number of other events. Cabasino *et al.* have proposed efficient algorithms to solve the above problems using LPNs (see (Cabasino, 2009) for technical details). Such algorithms are based on the notion of BRG, and are implemented by the MATLAB functions in directory *PN_DIAG*. Table 5 lists some of the functions.

*Example 6.* Consider again the LPN in Fig. 8. Let $t_3$ and $t_5$ be fault transitions that belong to two different fault classes $f_1$ and $f_2$, respectively, $t_2$ and $t_4$ be regular unobservable transitions, and the label associated to $t_1$ be $a$. To check if $f_1$ and $f_2$ are diagnosable, first the BRG

Table 6. Main Functions in *PN_OPAC*

| Name | Function |
|---|---|
| `BRG_Cur` | constructs the corresponding BRG for current-state opacity wrt a given secret |
| `showBRG_Cur` | shows the BRG in a readable text form |
| `CurOpac` | checks if the LPN system is current-state opaque wrt a given secret |
| `InitOpac` | checks if the LPN system is initial-state opaque wrt a given secret |

```
>> [Pre, Post, M0] = getMatrix; E = {['a']}; L = {[1, 3]};
W = [-1 0 0 -1]; K = -2; S = { W; K};
>> [CSO, Obs, y] = CurOpac(Pre, Post, M0, L, E, S)
CSO =
     1
Obs =
     [4]   '1'   [4×3 double]   [0]   [ ]
y =
     [0]   [1×2 double]   [1×2 double]   [1×3 double]
```

Fig. 17. Check if the LPN system is current-state opaque.

for diagnosis is constructed using funcition `BRG_D`, based on which we construct the basis reachability diagnoser through function `BRD`. Then, the modified BRG is built using `MBRG`, and finally function `diagnosability` can be applied. The above steps and the obtained results are presented in Fig. 15, which shows that neither of $f_1$ and $f_2$ is diagnosable. Given an observation $w = a$, to diagnose if fault $f_1$ or $f_2$ has occurred we use function `diagnosis`, and the result is shown in Fig. 16. When nothing is observed (i.e., suffix equals *eps*) the diagnosis state is [0 0], which means that neither $f_1$ nor $f_2$ has occurred; when $a$ is observed, the diagnosis state is [2 2], which means that $f_1$ and $f_2$ may have occurred and they are contained in one (but not in all) justification of $w$. ◇

### *4.4 State-Based Opacity Verification*

Given an LPN system and a secret, opacity verification consists in checking if the system is opaque wrt the secret. PetriBaR can be used to verify current-state opacity (CSO) and initial state opacity (ISO). It is assumed that the secret is described by a set of GMECs, and the unobservable induced subnet is acyclic. For initial-state opacity verification, it is also assumed that none of the secret markings is weakly exposable, i.e., their unobservable reach is not strictly contained in the secret.

Function for opacity verification are in directory *PN_OPAC*, and are based on the results in (Tong et al., 2017). Some significant functions are listed in Table 6.

*Example 7.* Consider again the PN system in Fig. 8. Let $M_0 = [1\ 1\ 0\ 0]^T$, $T_u = \{t_2, t_4, t_5\}$, the labels assigned to $t_1$ and $t_3$ be $a$, and the secret $S = \{M | M(p_1) + M(p_4) \geq 2\}$. We use function `CurOpac` to check if the LPN is current-state opaque wrt $S$, and the result is presented in Fig. 17. The value of $CSO$ is 1, which means that the LPN is current-state opaque; if the value is 0, the LPN system is not current-state opaque. ◇

## 5. CONCLUSIONS

The presented toolbox PetriBaR provides multiple functions for PN analysis, including the solutions to four problems: reachability, state estimation, fault diagnosis, and opacity verification. In all cases BRG-based techniques are implemented. Compared with other PN tools, PetriBaR has high flexibility, portability and compatibility. As a future work, we plan to integrate all functions with the graphical editor to form a fully functional software. Meanwhile, more functions will be continuously added to enrich the toolbox, such as functions for opacity enforcement.

## REFERENCES


Balduzzi, F., Giua, A., and Menga, G. (2000). First-order hybrid Petri nets: a model for optimization and control. *IEEE transactions on robotics and automation*, 16(4), 382–399.

Cabasino, M.P. (2009). *Diagnosis and identification of discrete event systems using Petri nets*. Ph.D. thesis, Ph. D. Thesis.

Cabasino, M.P., Giua, A., Pocci, M., and Seatzu, C. (2011). Discrete event diagnosis using labeled Petri nets. an application to manufacturing systems. *Control Engineering Practice*, 19(9), 989–1001.

Cabasino, M.P., Giua, A., and Seatzu, C. (2010). Fault detection for discrete event systems using Petri nets with unobservable transitions. *Automatica*, 46(9), 1531–1539.

Giua, A. (1992). Petri nets as discrete event models for supervisory control. *Rensselaer Polytechnic Institute, Troy, NY*.

INA (2003). Integrated net analyzer (version 2.2). `https://www2.informatik.hu-berlin.de/lehrstuehle/automaten/ina/`.

Ma, Z., Tong, Y., Li, Z., and Giua, A. (2017). Basis marking representation of Petri net reachability spaces and its application to the reachability problem. *IEEE Transactions on Automatic Control*, 62(3), 1078–1093.

Moody, J.O. and Antsaklis, P.J. (2000). Petri net supervisors for DES with uncontrollable and unobservable transitions. *IEEE Transactions on Automatic Control*, 45(3), 462–476.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Procedings of the IEEE*, 77(4), 541–580.

Pastravanu, O., Matcovschi, M.H., and Mahulea, C. (2004). Petri net toolbox teaching discrete event systems under MATLAB. In *Advances in Automatic Control*, 247–255. Springer.

PetriBaR (2017). PetriBaR (version 1.0). `https://www.researchgate.net/publication/321431345_MATLAB_Toolbox_for_Petri_Nets`.

Sessego, F., Giua, A., and Seatzu, C. (2008). Hypens: a matlab tool for timed discrete, continuous and hybrid Petri nets. *Lecture Notes in Computer Science*, 5062, 419–428.

TAPAAL (2017). TAPAAL (version 3.3.0). `http://www.tapaal.net/`.

Tong, Y., Ma, Z., Li, Z., Seatzu, C., and Giua, A. (2016). Verification of language-based opacity in Petri nets using verifier. In *American Control Conference (ACC), 2016*, 757–763. IEEE.

Tong, Y., Li, Z., Seatzu, C., and Giua, A. (2017). Verification of state-based opacity using Petri nets. *IEEE Transactions on Automatic Control*, 62(6), 2823–2837.