Non-interference analysis of bounded labeled Petri nets

Ning Ran[a], Zhengguang Wu[b], Shaokang Zhang[c], Zhou He[d], Carla Seatzu[e]

[a]*Laboratory of Energy-Saving Technology, College of Electronic & Information Engineering, Hebei University, Baoding 071002, China.* ranning87@hotmail.com
[b]*College of Control Science and Engineering, Zhejiang University, Hangzhou 310027, China.* nashwzhg@126.com
[c]*School of Cyber Security and Computer, Hebei University, Baoding 071002, China.* zhangshaokang@hbu.edu.cn
[d]*College of Mechanical & Electrical Engineering, Shaanxi University of Science & Technology, Xi'an 710021, China.* hezhou@sust.edu.cn
[e]*Department of Electrical and Electronic Engineering, University of Cagliari, Cagliari 09124, Italy.* carla.seatzu@unica.it

Abstract

This paper focuses on a fundamental problem on information security of bounded labeled Petri nets: non-interference analysis. As in hierarchical control, we assume that a system is observed by users at different levels, namely high-level users and low-level users. The output events produced by the firing of transitions are also partitioned into high-level output events and low-level output events. In general, high-level users can observe the occurrence of all the output events, while low-level users can only observe the occurrence of low-level output events. A system is said to be non-interferent if low-level users cannot infer the firing of transitions labeled with high-level output events by looking at low-level outputs. In this paper, we study a particular non-interference property, namely strong non-deterministic non-interference (SNNI), using a special automaton called SNNI Verifier, and propose a necessary and sufficient condition for SNNI.

Keywords: Discrete event systems, labeled Petri nets, information security, non-interference

1. Introduction

Information security is quite important for nowadays control systems. In the past few decades, some problems related to information security of

discrete event systems (DESs), such as *opacity* [1–9], *non-interference* [10–19], *anonymity* [20; 21], were extensively studied and a series of methods have been developed.

Information leak is a typical security problem and has received a lot of attention. In such a scenario, an intruder may speculate sensitive information based on the released data. In the DES framework, this problem can be formalized as a *non-interference problem* such as *strong nondeterministic non-interference (SNNI)* and *bisimulation strong non-deterministic non-interference (BSNNI)*.

In [10], the formal definition of SNNI was first proposed for automata, where the system is assumed to be monitored by high-level users (e.g. the administrators) and low-level users (e.g. the customers) from different points of view. Although both classes of users know the structure of the system, the high-level users can observe all the events, while the low-level users can only observe a subset of events, namely the set of low-level events. A system is said to be SNNI if for any enabled sequence that contains high-level events, its projection on the set of low-level events is still enabled. In other words, if a system is SNNI, then low-level users cannot infer the occurrence of high-level events no matter which low-level events are observed. In addition, some related security properties of automata are also studied and compared with SNNI in [10].

In [11], the problem of SNNI was first rephrased in the Petri net framework. Two new non-interference properties are defined taking advantage from the structural feature of safe Petri nets. Finally, reference [11] proves that the absence of causal places and conflict places is a sufficient condition for the structural non-interference. Best *et al.* [12] prove that the property of SNNI and the property of *nondeducibility on composition (NDC)* are equivalent, and also prove that the latter is decidable for unbounded Petri nets. Basile *et al.* [14] investigate SNNI of Petri nets based on the solution of some integer linear programming (ILP) problems, and propose a necessary and sufficient condition for SNNI. The property of BSNNI is also studied in this paper. This method is then improved by removing the assumption of acyclicity of Petri nets [15]. To enforce SNNI to Petri nets, an online supervisory control policy is proposed in [16] that dynamically disables a subset of transitions whose occurrence may violate the conditions for SNNI. In [17], Basile *et al.* extend such ILP-based methods to the SNNI and BSNNI analysis of labeled Petri nets. Although the assumption that the low-level subnet is acyclic is not required in these papers, authors use the linear characterizations of the

firing vectors enabled at a certain marking with a number of constraints that linearly increases with the length of the corresponding sequence.

To avoid exhaustive enumeration of the whole state space, Ran *et al.* [18] recently propose new methods for SNNI analysis and enforcement of bounded Petri nets based on the notion of basis marking. They prove that the property of SNNI can be analyzed checking the set of arcs in the basis reachability graph (BRG). To enforce SNNI, a supervisory control policy is designed that disables a subset of low-level transitions as a response to each transition sequence generated by the system. Such methods have also been extended to the analysis and enforcement of BSNNI of Petri nets in [19].

The method proposed in [18] cannot be directly applied to labeled Petri nets, in which two or more transitions that are simultaneously enabled may produce the same output event. In such a case, the problems of SNNI analysis and enforcement become more complicated since to a low-level transition sequence whose enabling is due to the occurrence of high-level transitions, there may correspond another low-level transition sequence sharing the same output event sequence. This paper extends the analysis method in [18] to the case of labeled Petri nets. More precisely, we first construct an unfolded version of the BRG to identify all the low-level transition sequences whose enabling may only be due to the occurrence of high-level transitions. Then a special automaton, called *SNNI Verifier*, is introduced to check whether every such sequence is indistinguishable from another low-level transition sequence whose enabling is not due to the occurrence of high-level transitions. Based on such a Verifier, a necessary and sufficient condition is given for SNNI. Overall, the main contributions of this paper may be summarized as follows:

- We formalize the relationship between the language of the BRG and the SNNI property of labeled Petri nets. Then, the notion of *unfolded basis reachability graph (UBRG)* is proposed, which facilitates us to identify the set of sequences that are related to SNNI.

- We define a special automaton called SNNI Verifier, and give a necessary and sufficient condition for SNNI of labeled Petri nets.

The rest of the paper is organized as follows. Section 2 provides the required preliminaries on labeled Petri nets. In Section 3, the notions of basis marking and BRG are first recalled, then the notion of UBRG is presented. We introduce the notion of SNNI Verifier and propose a necessary and sufficient condition for SNNI in Section 4. Finally, conclusions are drawn in

Section 5, where our future research directions in this framework are also illustrated.

2. Preliminaries

2.1. Petri Nets

A Petri net (PN) is a 4-tuple $N = (P, T, F, W)$, where P is the set of places, T is the set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs, W is a mapping that assigns a non-negative integer weight to an arc: $W(x, y) > 0$ if $(x, y) \in F$, and $W(x, y) = 0$ if $(x, y) \notin F$, where $x, y \in P \cup T$. The incidence matrix $[N]$ is a $|P| \times |T|$ integer matrix with $[N](p, t) = W(t, p) - W(p, t)$, where $p \in P$ and $t \in T$. Given a place p (transition t), its incidence vector, a row (column) in $[N]$, is denoted by $[N](p, \cdot)$ ($[N](\cdot, t)$). Let $x \in P \cup T$ be a node of net N. The preset of x is ${}^\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$ while the postset is $x^\bullet = \{y \in P \cup T \mid (x, y) \in F\}$.

A marking m of a PN N is a mapping from P to $\mathbb{N} = \{0, 1, 2, ...\}$; $m(p)$ denotes the number of tokens in p. (N, m_0) denotes a PN system with initial marking m_0.

A transition t is enabled at marking m if $\forall p \in {}^\bullet t, m(p) \geq W(p, t)$, which is denoted by $m[t\rangle$. Firing t at m yields a new marking m' such that $\forall p \in P, m'(p) = m(p) + [N](p, t)$, which is denoted by $m[t\rangle m'$. The notation $m[s\rangle$ denotes that the transition sequence $s \in T^*$ is enabled at m. Marking m'' is said to be reachable from m if there exists a transition sequence s such that $m[s\rangle m''$. The set of markings reachable from m in N is called the reachability set of (N, m) and is denoted by $R(N, m) = \{m' \in \mathbb{N}^{|p|} \mid \exists s \in T^*, m[s\rangle m'\}$. The set of all transition sequences that are enabled at the initial marking m_0 is $\mathcal{L}(N, m_0) = \{s \in T^* \mid m_0[s\rangle\}$.

Given a transition sequence $s \in T^*$, we denote by $\pi(s)$ the Parikh vector of s. Let $y = \pi(s)$, we write $y(t) = k$ if transition t appears k times in s.

A PN is said to be *bounded* if there exists a positive constant k such that $\forall p \in P$, $\forall m \in R(N, m_0)$, $m(p) \leq k$. It is *unbounded* if it is not bounded. A sequence of nodes $x_1 x_2 \ldots x_n$ is called a path if $x_{i+1} \in x_i^\bullet$, where $x_i \in P \cup T$. If the first node is the same as the last node, i.e., $x_1 = x_n$, then the path is called a *circuit*. A PN with no circuits is said to be *acyclic*.

Let $T' \subseteq T$ be a subset of transitions, the subnet $N' = (P, T', F', W')$ is called the *T'-induced subnet* of N, where F' is the restriction of F to $(P \times T') \cup (T' \times P)$ and W' is the restriction of W to $(P \times T') \cup (T' \times P)$.

2.2. Labeled Petri Nets

A labeled PN system (LPNS) is a 4-tuple (N, m_0, l, A), where A is the alphabet of labels and $l : T^* \to A^*$ is a labeling function defined as follows:

- $l(t) \in A$, if $t \in T$;
- $l(\varepsilon) = \varepsilon$, where ε is the empty string;
- $l(st) = l(s)l(t)$, if $s \in T^*$ and $t \in T$.

When a transition fires it produces an output event which corresponds to the label associated with it. In simple words, the label can be seen as the output event produced by a sensor associated with the event modeled by the corresponding transition.

Transitions are called *indistinguishable* if they share the same label (or equivalently, the same output event); alternatively, they are called *distinguishable*. Given a label (or output) sequence $w \in A^*$, we denote by $l^{-1}(w)$ the set of transition sequences that are consistent with w, namely

$$l^{-1}(w) = \{s \in \mathcal{L}(N, m_0) \mid l(s) = w\}.$$

The language of labels is denoted by

$$l(\mathcal{L}(N, m_0)) = \{l(s) \in A^* \mid s \in \mathcal{L}(N, m_0)\}.$$

The alphabet A can be partitioned as $A = A_E \cup A_I$, where A_E denotes the set of *explicit* labels, A_I denotes the set of *implicit* labels and $A_E \cap A_I = \emptyset$. Accordingly, the set T of transitions is partitioned as $T = T_E \cup T_I$, where T_E is the set of explicit transitions, i.e.,

$$T_E = \{t \in T \mid l(t) \in A_E\};$$

T_I is the set of implicit transitions, i.e.,

$$T_I = \{t \in T \mid l(t) \in A_I\}.$$

We denote by $[N]_E$ ($[N]_I$) the restriction of $[N]$ to T_E (T_I), namely, $[N]_E$ is a $|P| \times |T_E|$ integer matrix with $[N]_E(p, t) = W(t, p) - W(p, t)$, where $p \in P$ and $t \in T_E$.

The projection $P_E : T^* \to T_E^*$ is defined as:

- $P_E(\varepsilon) = \varepsilon$;
- for all $s \in T^*$ and $t \in T$, $P_E(st) = P_E(s)t$ if $t \in T_E$, and $P_E(st) = P_E(s)$ otherwise.

Analogously, we define the projection $P_I : T^* \to T_I^*$. The projections of $\mathcal{L}(N, m_0)$ over T_E and T_I are denoted by $P_E(\mathcal{L}(N, m_0))$ and $P_I(\mathcal{L}(N, m_0))$, respectively.

2.3. Nondeterministic Finite Automata

A nondeterministic finite automaton (NFA) is a 4-tuple $G = (\Theta, E, \Delta, X_0)$, where Θ is a finite set of states, E is a set of events, $\Delta \subseteq \Theta \times E \times \Theta$ is the transition relation, $X_0 \subseteq \Theta$ is a set of initial states. An NFA with labels is a 6-tuple $(\Theta, E, \Delta, X_0, l, A)$, where A is the alphabet of labels and $l : E^* \to A^*$ is a labeling function that is similar to the labeling function defined in LPNs.

3. Unfolded Basis Reachability Graph

In this section, we first recall the notions of basis marking and basis reachability graph (BRG), then introduce the notion of *unfolded BRG* (UBRG).

3.1. Basis Reachability Graph

The notion of basis marking is first proposed in [22] for estimating the markings of PNs in the presence of silent transitions. It provides a compact way to describe the set of reachable markings consistent with a given observation. Then this method has been efficiently extended to study some problems related to partial observation, such as fault diagnosis [23–25], fault prognosis [26], reachability [27], etc. We adopt the following two assumptions, which are necessary to apply the method based on the notion of basis marking.

A1) The PN system is bounded.

A2) The T_I-induced subnet is acyclic.

Note that T_I-induced subnet is a subnet (P, T_I, F_I, W_I), where F_I is the restriction of F to $(P \times T_I) \cup (T_I \times P)$ and W_I is the restriction of W to $(P \times W_I) \cup (W_I \times P)$. Although Assumption A2 limits the application scope of the proposed method, it is reasonable for a real system since otherwise

the system may fall into an infinite loop without generating any detectable events.

Given a marking $m \in R(N, m_0)$ and a transition $t \in T_E$, the set of *explanations* of t at m is defined as

$$\Sigma(m, t) = \{s \in T_I^* \mid m[s\rangle m', m'[t\rangle\},$$

and the set of *e-vectors* is defined as

$$Y(m, t) = \{\pi(s) \mid s \in \Sigma(m, t))\}.$$

The set of *minimal explanations* of t at m is defined as

$$\Sigma_{min}(m, t) = \{s \in \Sigma(m, t) \mid \nexists s' \in \Sigma(m, t) : \pi(s') \lneq \pi(s)\}.$$

The set of *minimal e-vectors* is denoted by

$$Y_{min}(m, t) = \{\pi(s) \mid s \in \Sigma_{min}(m, t))\}.$$

In particular, $Y_{min}(m, t)$ can be computed by Algorithm 4.4 in [24].

In simple words, $\Sigma(m, t)$ is the set of implicit sequences whose firing at M enables t. Among the above sequences, $\Sigma_{min}(m, t)$ selects those whose Parikh vector is minimal. The Parikh vectors of these sequences are called minimal e-vectors.

Let (N, m_0, l, A) be an LPNS and $w \in A_E^*$ be a sequence of explicit labels. The set of pairs ($s_E \in T_E^*$ with $l(s_E) = w$ and the *justification*) is indicated as

$$\hat{\mathcal{J}}(w) = \{(s_E, s_I), s_E \in T_E^*, l(s_E) = w, s_I \in T_I^* \mid [\exists s \in l^{-1}(w) : s_E = P_E(s), s_I = P_I(s)]$$

$$\wedge[\nexists s' \in l^{-1}(w) : s_E = P_E(s'), s'_I = P_I(s') \wedge \pi(s'_I) \lneq \pi(s_I)]\},$$

and the set of pairs ($s_E \in T_E^*$ with $l(s_E) = w$ and the j-vector) is denoted by

$$\hat{Y}_{min}(m_0, w) = \{(s_E, y), s_E \in T_E^*, l(s_E) = w, y \in \mathbb{N}^{|T_I|} \mid \exists (s_E, s_I) \in \hat{\mathcal{J}}(w) : \pi(s_I) = y\}.$$

In other words, $\hat{\mathcal{J}}(w)$ is the set of pairs whose first element is the sequence s_E labeled w and whose second element is the corresponding sequence of implicit transitions interleaved with s_E whose firing enables s_E and whose Parikh vector is minimal. The Parikh vectors of these implicit sequences are called j-vectors (justification vector). $\hat{Y}_{min}(m_0, w)$ is the set of pairs whose

first element is the sequence s_E labeled w and whose second element is the corresponding j-vector.

The set of *basis markings* of w is indicated as

$$M_b(w) = \{m \in \mathbb{N}^{|P|} \mid m = m_0 + [N]_I \cdot \pi(s_I) + [N]_E \cdot \pi(s_E), (s_E, s_I) \in \hat{\mathcal{J}}(w)\},$$

and $\mathcal{M}_b = \bigcup_{w \in l(\mathcal{L}(N,m_0))} M_b(w)$.

Therefore, a basis marking is a marking that can be reached from the initial marking firing a sequence of transitions that is consistent with the sequence of explicit labels and a sequence of implicit transitions, interleaved with the previous sequence, whose firing is strictly necessary to enable it (in the sense that its Parikh vector is minimal) [24]. The set of basis markings is a subset (usually a strict subset) of the set of reachable markings. Therefore, if the net is bounded, the set of basis markings is finite.

Definition 1 ([23]). *Let (N, m_0, l, A) be an LPNS, the BRG $G = (\Theta, E, \Delta, m_0)$ is an NFA, where:*

- $\Theta = \mathcal{M}_b$*;*
- $E \subseteq (T_E \times \mathbb{N}^{|T_I|})$ *is the set of events;*
- $\Delta \subseteq \Theta \times E \times \Theta$ *is the transition relation;*
- m_0 *is the initial state.*

The BRG is constructed by Algorithm 1.

From Algorithm 1, we know that for each transition relation $(m, (t, y), m') \in \Delta$, y is a minimal e-vector of t at m and $m' = m + [N]_I \cdot y + [N](\cdot, t)$.

Note that the BRG constructed by Algorithm 1 is slightly different from the BRG proposed in [23]. In the BRG constructed in [23], each node contains two entries, where the second entry is a binary value that allows us to keep track of some information which is not relevant now, so it is omitted.

Example 1. *Consider the LPNS in Fig. 1, where $A_E = \{a, b, c, d, e\}$, $A_I = \{f, g\}$, $T_E = \{l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8, l_9\}$, $T_I = \{h_1, h_2\}$. It holds that $\Sigma(m_0, l_1) = \{h_1\}$ and $\Sigma(m_0, l_5) = \{\varepsilon\}$. It follows that $\Sigma_{min}(m_0, l_1) = \{h_1\}$, $Y_{min}(m_0, l_1) = \{[1\ 0]^T\}$, and $\Sigma_{min}(m_0, l_5) = \{\varepsilon\}$, $Y_{min}(m_0, l_5) = \{[0\ 0]^T\}$. Let $w = ab$. It holds that $\hat{\mathcal{J}}(w) = \{(l_1 l_2, h1), (l_8 l_9, \varepsilon)\}$, and $\hat{Y}_{min}(m_0, w) = \{(l_1 l_2, [1\ 0]^T), (l_8 l_9, [0\ 0]^T)\}$. The set of basis markings is detailed in Table 1. The BRG G is shown in Fig. 2.*

Algorithm 1: BRG Construction

Input: An LPNS (N, m_0, l, A), where $A = A_E \cup A_I$.

Output: The BRG $G = (\Theta, E, \Delta, m_0)$.

Let m_0 be the root node, $\Theta = \{m_0\}$, $E = \emptyset$, $\Delta = \emptyset$.
while *nodes with no tag exist,* **do**
  choose a node $m \in \Theta$ with no tag,
  for *all* $t \in T_E$, **do**
    if $Y_{min}(m, t) \neq \emptyset$, **then**
      for *all* $y \in Y_{min}(m, t)$, **do**
        let $m' = m + [N]_I \cdot y + [N](\cdot, t)$,
        let $\Theta = \Theta \cup \{m'\}$, $E = E \cup \{(t, y)\}$ and $\Delta = \Delta \cup \{(m, (t, y), m')\}$.
      end
    end
  end
  tag node m "old".
end
Remove all tags.

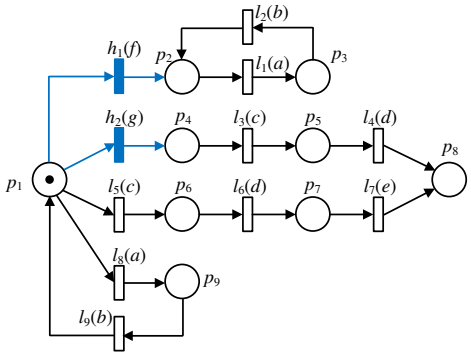

Figure 1: A labeled Petri net system.

Table 1: The set of basis markings.

Node	Basis marking
m_0	$[1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$
m_1	$[0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0]^T$
m_2	$[0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$
m_3	$[0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]^T$
m_4	$[0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0]^T$
m_5	$[0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0]^T$
m_6	$[0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0]^T$
m_7	$[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1]^T$

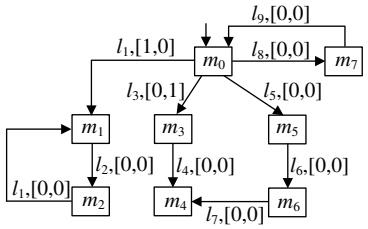

Figure 2: BRG G of the LPNS in Fig. 1.

Let $G = (\Theta, E, \Delta, m_0)$ be a BRG and $\mathcal{L}(G)$ be the language generated by G. Let $\sigma = (\lambda_1, y_1)(\lambda_2, y_2)...(\lambda_k, y_k) \in \mathcal{L}(G)$. We denote by $\varphi : E^* \to T_E^*$ and $\varphi' : E^* \to \mathbb{N}^{|T_I|}$ the sequence and the vector, respectively:

$$\varphi(\sigma) = \lambda_1\lambda_2...\lambda_k$$

and

$$\varphi'(\sigma) = \sum_{i=1}^{k} y_i.$$

In other words, $\varphi(\sigma)$ is the concatenation of the first entry in each arc in σ, and $\varphi'(\sigma)$ is the summation of the second entry in each arc in σ, namely a j-vector of $\varphi(\sigma)$. For example, consider the BRG in Fig. 2, given a sequence $\sigma = (l_1, [1,0])(l_2, [0,0])(l_1, [0,0])$, it holds that $\varphi(\sigma) = l_1 l_2 l_1$ and $\varphi'(\sigma) = [1,0] + [0,0] + [0,0] = [1,0]$.

Proposition 1 ([25]). *Let (N, m_0, l, A) be an LPNS and $G = (\Theta, E, \Delta, m_0)$ be the BRG constructed by Algorithm 1. It holds that*

$$\varphi(\mathcal{L}(G)) = P_E(\mathcal{L}(N, m_0)).$$

In other words, the set of sequences in $\varphi(\mathcal{L}(G))$ coincides with the projection of $\mathcal{L}(N, m_0)$ over the set T_E.

3.2. Unfolded Basis Reachability Graph

Definition 2. *Let (N, m_0, l, A) be an LPNS, the unfolded basis reachability graph (UBRG) $G_U = (\Theta_U, E_U, \Delta_U, m_0)$ is an NFA, where:*

- $\Theta_U = \mathcal{M}_b \cup (\mathcal{M}_b \times \{\alpha_i, \beta_j\})$*, where* $i, j \in \{1, 2, 3, \ldots\}$*;*
- $E_U \subseteq (T_E \times \mathbb{N}^{|T_I|})$ *is the set of events;*
- $\Delta_U \subseteq \Theta_U \times E_U \times \Theta_U$ *is the transition relation;*
- m_0 *is the initial state.*

The UBRG can be constructed by Algorithm 2, which also returns two sets (Φ_α and Φ_β) that are defined later on.

Hereinafter, we use $m \xrightarrow[G_U]{\sigma} m'$ to denote that m' is reached from m in G_U through a sequence σ.

Algorithm 2: UBRG construction and computation of sets Φ_α and Φ_β

Input: An LPNS (N, m_0, l, A), where $A = A_E \cup A_I$.
Output: The UBRG $G_U = (\Theta_U, E_U, \Delta_U, m_0)$, and two sets Φ_α, Φ_β.

Let m_0 be the root node, $\Theta_U = \{m_0\}$, $E_U = \emptyset$, $\Delta_U = \emptyset$, $M_{dup} = \emptyset$.
while *nodes with no tag exist,* **do**
  choose a node $m \in \Theta_U$ with no tag,
  if *m is identical to a node in the path from the root node m_0 to m,* **then**
    let $M_{dup} = M_{dup} \cup \{m\}$ and goto step 2.
  end
  for *all $t \in T_E$,* **do**
    if $Y_{min}(m, t) \neq \emptyset$, **then**
      for *all $y \in Y_{min}(m, t)$,* **do**
        let $m' = m + [N]_I \cdot y + [N](\cdot, t)$,
        let $\Theta_U = \Theta_U \cup \{m'\}$, $E_U = E_U \cup \{(t, y)\}$ and $\Delta_U = \Delta_U \cup \{(m, (t, y), m')\}$.
      end
    end
  end
  tag node m "old".
end
Let $i = j = 0$ and $\Phi_\alpha = \Phi_\beta = \emptyset$.
for *all leaf nodes $m \in \Theta_U$,* **do**
  if *in the path from m_0 to m there exists at least one arc whose second entry is not $\overrightarrow{0}$,* **then**
    if $m \notin M_{dup}$, **then**
      $i = i + 1$,
      replace m with (m, α_i) and $\Phi_\alpha = \Phi_\alpha \cup \{\alpha_i\}$,
    else
      $j = j + 1$,
      replace m with (m, β_j) and $\Phi_\beta = \Phi_\beta \cup \{\beta_j\}$.
    end
  end
end
Remove all tags.

Remark 1. *The UBRG can be viewed as the "unfolded version" of the BRG*[1]*. The differences between Algorithm 1 and Algorithm 2 are twofold. The first difference consists in the way repeated nodes are handled: the former fuses repeated nodes, while the latter does not fuse repeated nodes and records a node in the set* M_{dup} *if it is identical to another node in the path from the root node to the considered one (Steps 4 to 6 in Algorithm 2), where the subscript "dup" stands for "duplicated". Therefore, a path of an infinite length in the BRG (i.e., a path ending in a cycle) coincides with a path in the UBRG whose leaf node belongs to* M_{dup}*, and vice versa. The second difference is that a leaf node (say* m*) in the UBRG may be marked "*α_i*" or "*β_j*"* ($i, j \in \{1, 2, 3, ...\}$) *if*

$$m_0 \xrightarrow[G_U]{\sigma} m \quad and \quad \varphi'(\sigma) \neq \overrightarrow{0},$$

where "α_i*" denotes that* m *is not a duplicated node (i.e.,* $m \notin M_{dup}$*), while "*β_j*" denotes that* m *is a duplicated node (Steps 17 to 28). In other words, the set of paths in the UBRG whose leaf nodes are marked "*α_i*" or "*β_j*" can identify all the explicit transition sequences whose enabling may only be due to the occurrence of implicit transitions. The subscripts* i *and* j *record the numbers of such two kinds of paths, respectively (Steps 21 and 24). Finally,* Φ_α *and* Φ_β *denote the sets of "*α_i*" and "*β_j*" in the leaf nodes, respectively (Steps 22 and 25).*

Example 2. *Reconsider the LPNS in Fig. 1, where* $T_E = \{l_1, l_2, l_3, l_4, l_5, l_6, l_7, l_8, l_9\}$ *and* $T_I = \{h_1, h_2\}$*. The UBRG* G_U *is shown in Fig. 3, and* $M_{dup} = \{m_0, m_1\}$*. We observe that there are two paths in* G_U *containing implicit transitions, and their leaf nodes are tagged* α_1 *and* β_1*, respectively. It holds that* $\Phi_\alpha = \{\alpha_1\}$ *and* $\Phi_\beta = \{\beta_1\}$*. Indeed, the enabling of both sequences* l_3 *and* $l_3 l_4$ *are due to* h_2*, and the enabling of each sequence in* $\overline{(l_1 l_2)^*} \setminus \{\varepsilon\}$ *is due to* h_1*, where* $\overline{(l_1 l_2)^*}$ *is the prefix closure of* $(l_1 l_2)^*$*.*

[1]Unfolding a graph means transforming a graph - especially one that may have cycles - into a tree-like or acyclic structure that explicitly represents all possible paths starting from an initial node, without looping back. Since we are doing the unfolding of the BRG, we believe that the most appropriate and intuitive name to give to the resulting automaton is unfolded BRG (UBRG), even if it is a tree. This implies that the UBRG may have two identical nodes, which do not have to be merged.

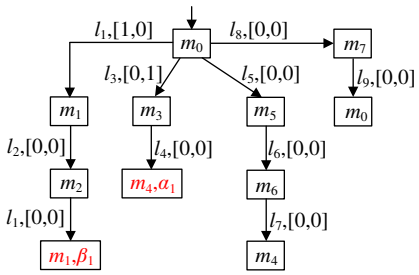

Figure 3: UBRG G_U of the LPNS in Fig. 1.

4. Strong Non-Deterministic Non-Interference Analysis

In this section, we recall the notion of SNNI and propose a method for the SNNI analysis.

4.1. The Notion of Strong Non-Deterministic Non-Interference

In the framework of hierarchical control, a system is observed by users at different levels. Although all users know the structure of the system, high-level users can observe all the output events (or labels), while low-level users can only observe a subset of output events. It is possible that there exist some malicious low-level users (e.g., intruders) intending to infer sensitive information. Therefore, it is necessary to verify if such information leaks may occur under the current hierarchical architecture.

Hereinafter, we denote by A_L the set of labels observable by the low-level users, and by A_H the set of labels that may only be observed by high-level users, i.e., $A_H = A \setminus A_L$. Accordingly, T_L denotes the set of low-level transitions, i.e.,

$$T_L = \{t \in T \mid l(t) \in A_L\},$$

and T_H denotes the set of high-level transitions, i.e.,

$$T_H = \{t \in T \mid l(t) \in A_H\}.$$

We also denote by $[N]_L$ ($[N]_H$) the restriction of $[N]$ to T_L (T_H). The T_L-induced subnet is denoted by N_L.

To use BRG/UBRG for the SNNI analysis, we view the sets A_L, A_H, T_L and T_H as the sets A_E, A_I, T_E and T_I in Subsection 2.1, respectively. Analogously to the projections P_E (P_I) defined in Section 2, we define the projections $P_L : T^* \to T_L^*$ ($P_H : T^* \to T_H^*$). Accordingly, the projection of $\mathcal{L}(N, m_0)$ over T_L (T_H) is denoted by $P_L(\mathcal{L}(N, m_0))$ ($P_H(\mathcal{L}(N, m_0))$).

Given an LPNS (N, m_0, l, A), the *low-level subnet system* is denoted by (N_L, m_0, l_L, A_L), where N_L is the T_L-induced subnet, l_L is equal to l restricted to T_L. For example, the low-level subnet system of the LPNS in Fig. 1 can be obtained removing the transitions and the arcs that are highlighted in blue color.

Here we recall the definition of SNNI of LPNSs.

Definition 3 ([17]). *Let (N, m_0, l, A) be an LPNS and (N_L, m_0, l_L, A_L) be the low-level subnet system. The PN system is SNNI if*

$$l(P_L(\mathcal{L}(N, m_0))) = l_L(\mathcal{L}(N_L, m_0)).$$

In plain words, an LPNS is SNNI if the projection of its language over T_L is indistinguishable from the language of its low-level subnet system. This implies that low-level users cannot infer the occurrence of high-level transitions after any sequence containing high-level transitions has occurred.

Proposition 2. *Let (N, m_0, l, A) be an LPNS and (N_L, m_0, l_L, A_L) be the low-level subnet system. Let $G = (\Theta, E, \Delta, m_0)$ be the BRG constructed using Algorithm 1, where low-level transitions and high-level transitions are viewed as the explicit transitions and the implicit transitions, respectively. The LPNS is SNNI if and only if*

$$l(\varphi(\mathcal{L}(G))) = l_L(\mathcal{L}(N_L, m_0)).$$

Proof. Straightforward from Proposition 1 and Definition 3. □

This proposition reveals the relation between the language of the BRG and the property of SNNI.

Proposition 3. *Let (N, m_0, l, A) be an LPNS and (N_L, m_0, l_L, A_L) be the low-level subnet system. Let $G = (\Theta, E, \Delta, m_0)$ be the BRG constructed using Algorithm 1, where low-level transitions and high-level transitions are viewed as the explicit transitions and the implicit transitions, respectively. The LPNS is SNNI if and only if*

$$\forall s \in \varphi(\mathcal{L}(G)),\ \exists s' \in \mathcal{L}(N_L, m_0) :\ l(s) = l_L(s').$$

Proof. By Proposition 2, a system is SNNI if and only if

$$l_L(\mathcal{L}(N_L, m_0)) \subseteq l(\varphi(\mathcal{L}(G)))$$

and

$$l_L(\mathcal{L}(N_L, m_0)) \supseteq l(\varphi(\mathcal{L}(G)))$$

We first observe that

$$l_L(\mathcal{L}(N_L, m_0)) \subseteq l(P_L(\mathcal{L}(N, m_0)))$$

since (N_L, m_0, l_L, A) is the low-level subnet system of (N, m_0, l, A_L). Therefore, by Proposition 1, it always holds that

$$l_L(\mathcal{L}(N_L, m_0)) \subseteq l(\varphi(\mathcal{L}(G))).$$

It means that the LPNS is SNNI if and only if

$$l_L(\mathcal{L}(N_L, m_0)) \supseteq l(\varphi(\mathcal{L}(G))),$$

i.e., for any transition sequence $s \in \varphi(\mathcal{L}(G))$, there exists a sequence $s' \in \mathcal{L}(N_L, m_0)$ such that $l(s) = l_L(s')$. Hence, the result holds. □

By Proposition 3, we can analyze SNNI only focusing on the set of sequences in $\varphi(\mathcal{L}(G)$. Compared with Proposition 2, Proposition 3 provides a more intuitive condition for SNNI. It should be noted that this proposition may not be directly used for the analysis of SNNI since it is difficult to check all the sequences in $\varphi(\mathcal{L}(G)$ and in $\mathcal{L}(N_L, m_0)$, even if it gives a necessary and sufficient condition for SNNI. Therefore, in the following subsection, we define a special automaton, called SNNI Verifier (SV), and present a systematic method for SNNI analysis.

4.2. Strong Non-Deterministic Non-Interference Verifier

We first give the definition of *parallel composition*, which is inspired by the one in [28].

Definition 4. *Given two NFA with labels $G_1 = (\Theta_1, E_1, \Delta_1, m_{1,0}, l_L, A_L)$ and $G_2 = (\Theta_2, E_2, \Delta_2, m_{2,0}, l_L, A_L)$, the parallel composition of G_1 and G_2 is the NFA $G_1||G_2 = (\Theta_1 \times \Theta_2, E_1 \times E_2, \Delta, (m_{1,0}; m_{2,0}))$, where*

- *$((x_1; x_2), (e_1, e_2), (x_1'; x_2')) \in \Delta$ if $(x_1, e_1, x_1') \in \Delta_1$, $(x_2, e_2, x_2') \in \Delta_2$, e_1 and e_2 share the same label;*

- $((x_1; x_2), (e_1, \varepsilon), (x_1'; x_2)) \in \Delta$ *if* $(x_1, e_1, x_1') \in \Delta_1$*, and the label of* e_1 *is the empty string* ε*;*

- $((x_1; x_2), (\varepsilon, e_2), (x_1; x_2')) \in \Delta$ *if* $(x_2, e_2, x_2') \in \Delta_2$*, and the label of* e_2 *is the empty string* ε*.*

Now we introduce the notion of SV.

Definition 5. *Let* (N, m_0, l, A) *be an LPNS and* (N_L, m_0, l_L, A_L) *be the low-level subnet system. Let* $G_U = (\Theta_U, E_U, \Delta_U, m_0)$ *be the UBRG constructed using Algorithm 2, where low-level transitions and high-level transitions are viewed as the explicit transitions and the implicit transitions, respectively. The SNNI Verifier* $G_V = (\Theta_V, E_V, \Delta_V, v_0)$ *is an NFA, where*

- $\Theta_V = \Theta_U \times R(N_L, m_0)$*;*

- $E_V \subseteq T_L \times T_L$ *is the set of events;*

- $\Delta_V \subseteq \Theta_V \times E_V \times \Theta_V$ *is the transition relation;*

- $v_0 \in \Theta_V$ *is the initial state.*

The SV can be constructed by Algorithm 3, which also computes two sets (Ψ_α *and* Ψ_β*) that are defined later on.*

According to Algorithm 3, the SV G_V is constructed as the parallel composition of G_U and the reachability tree of (N_L, m_0, l_L, A_L), where synchronization is performed on the set A_L. In particular, repeated nodes are not fused and recorded in the set M'_{dup}. In Steps 15 to 23, all the leaf nodes are checked and the information on "α_i" and "β_j" ($i, j \in \{1, 2, 3, ...\}$) are recorded by the sets Ψ_α and Ψ_β, respectively.

Example 3. *Reconsider the LPNS in Fig. 1. The reachability tree of its low-level subnet system is shown in Fig. 4. The SV is computed using Algorithm 3 and is shown in Fig. 5. Note that the transition pair over each arc must share the same label, e.g.,* $l(l_1) = l(l_8) = a$ *and* $l(l_3) = l(l_5) = c$*. Furthermore, it is* $M'_{dup} = \{(m_0; m_0), (m_1, \beta_1; m_7)\}$*,* $\Psi_\alpha = \{\alpha_1\}$ *and* $\Psi_\beta = \{\beta_1\}$*.*

Algorithm 3: SV construction and computation of sets Ψ_α and Ψ_β

Input: An LPNS (N, m_0, l, A).

Output: The SV $G_V = (\Theta_V, E_V, \Delta_V, v_0)$, and two sets Ψ_α, Ψ_β.

Construct the UBRG $G_U = (\Theta_U, E_U, \Delta_U, m_0)$ using Algorithm 2, where low-level transitions and high-level transitions are viewed as the explicit transitions and the implicit transitions, respectively.

Let $v_0 = (m_0; m_0)$, $\Theta_V = \{v_0\}$, $E_V = \emptyset$, $\Delta_V = \emptyset$, $M'_{dup} = \emptyset$.

while *nodes with no tag exist,* **do**

  choose a node $(x_1; x_2) \in \Theta_V$ with no tag,

  if x_1 *is in the form of* (m, β_j)*, and the node* $(m; x_2)$ *is in the path from the root node* v_0 *to* $(x_1; x_2)$*,* **then**

    let $M'_{dup} = M'_{dup} \cup (x_1; x_2)$ and goto step 3.

  end

  for *all* $e_1 \in E_U$ *and all* $e_2 \in T_L$*,* **do**

    if $(x_1, e_1, x'_1) \in \Delta_U$*,* $x_2[e_2\rangle x'_2$ *holds in* (N_L, m_0, l_L, A_L) *and* $l(\varphi(e_1)) = l_L(e_2)$*,* **then**

      $\Theta_V = \Theta_V \cup \{(x'_1; x'_2)\}$, $E_V = E_V \cup (\varphi(e_1), e_2)$,
      $\Delta_V = \Delta_V \cup \{(x_1; x_2), (\varphi(e_1), e_2)), (x'_1; x'_2)\}$.

    end

  end

  tag node $(x_1; x_2)$ "old".

end

Let $\Psi_\alpha = \Psi_\beta = \emptyset$.

for *all the leaf nodes* $(x; x') \in \Theta_V$*,* **do**

  if x *is in the form of* (m, α_i)*,* **then**

    $\Psi_\alpha = \Psi_\alpha \cup \{\alpha_i\}$,

  end

  if x *is in the form of* (m, β_j)*, and* $(x; x') \in M'_{dup}$*,* **then**

    $\Psi_\beta = \Psi_\beta \cup \{\beta_j\}$,

  end

end

Remove all tags.

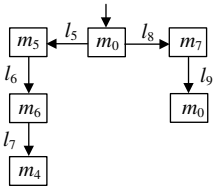

Figure 4: The reachability tree of the low-level subnet system of the PN system in Fig. 1.

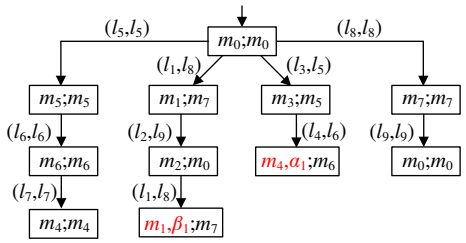

Figure 5: SV G_V of the PN system in Fig. 1.

Theorem 1. *Let $G_V = (\Theta_V, E_V, \Delta_V, v_0)$ be the SV constructed using Algorithm 3. The LPNS is SNNI if and only if*

$$\Phi_\alpha = \Psi_\alpha \quad \textit{and} \quad \Phi_\beta = \Psi_\beta,$$

where Φ_α and Φ_β are computed in Algorithm 2, Ψ_α and Ψ_β are computed in Algorithm 3.

Proof. Given a sequence $\sigma \in \mathcal{L}(G)$, if $\varphi'(\sigma) = \overrightarrow{0}$, it clearly holds that $\varphi(\sigma) \in \mathcal{L}(N_L, m_0)$. Therefore, by Proposition 3, to analyze SNNI we only need to check the set of sequences in $\mathcal{L}(G)$ whose second entry is not $\overrightarrow{0}$, i.e., the enabling of $\varphi(\sigma)$ may only be due to the occurrence of high-level transitions. According to Algorithm 2, such sequences can be identified by the set of paths in G_U whose leaf nodes are marked "α_i" or "β_j", where $i, j \in \{1, 2, 3, ...\}$. In particular, Φ_α and Φ_β denote the sets of all "α_i" and "β_j" in G_U, respectively. According to the two sets Ψ_α and Ψ_β, one is able to know for which sequences in $\varphi(\mathcal{L}(G_U))$ whose leaf nodes are marked "α_i" or "β_j", there exist indistinguishable transition sequences in $\mathcal{L}(N_L, m_0)$. More precisely, by Algorithm 3, $\alpha_i \in \Psi_\alpha$ means that $\varphi(\sigma)$ is of finite length and there exists an indistinguishable low-level transition sequence in $\mathcal{L}(N_L, m_0)$; $\beta_i \in \Psi_\beta$ means that $\varphi(\sigma)$ is of infinite length and there exists an indistinguishable low-level transition sequence of infinite length in $\mathcal{L}(N_L, m_0)$. Therefore, the equalities $\Phi_\alpha = \Psi_\alpha$ and $\Phi_\beta = \Psi_\beta$ hold if and only if for any low-

level transition sequence whose enabling may only be due to the occurrence of high-level transitions, there exists an indistinguishable transition sequence in $\mathcal{L}(N_L, m_0)$. By Proposition 3, the LPNS is SNNI if and only if $\Phi_\alpha = \Psi_\alpha$ and $\Phi_\beta = \Psi_\beta$. This completes the proof. □

Example 4. *According to Theorem 1, the LPNS in Fig. 1 is SNNI since it holds that $\Phi_\alpha = \Psi_\alpha = \{\alpha_1\}$ and $\Phi_\beta = \Psi_\beta = \{\beta_1\}$. Indeed, for each transition sequence whose enabling may only be due to the occurrence of high-level transitions, there exists at least one indistinguishable low-level transition sequence in $\mathcal{L}(N_L, m_0)$. In more detail,*

- *for sequences l_3 and l_3l_4 in $\varphi(\mathcal{L}(G))$, there exist indistinguishable sequences l_5 and l_5l_6 in $\mathcal{L}(N_L, m_0)$, respectively;*
- *for each sequence in $\overline{(l_1l_2)^*} \setminus \{\varepsilon\} \subseteq \varphi(\mathcal{L}(G))$, there exists an indistinguishable sequence in $\overline{(l_8l_9)^*} \setminus \{\varepsilon\} \subseteq \mathcal{L}(N_L, m_0)$.*

Therefore, low-level users cannot infer the occurrence of high-level transitions.

Example 5. *Consider the LPNS in Fig. 6. Note that the only difference between this LPNS and the one in Fig. 1 consists only in the label of transition l_2. Their basis marking sets and their UBRGs are the same (see Table 1 and Fig. 3). The SV is computed using Algorithm 3 and is shown in Fig. 7. By Theorem 1, the LPNS is not SNNI since $\Phi_\alpha = \Psi_\alpha = \{\alpha_1\}$, $\Phi_\beta = \{\beta_1\}$ and $\Psi_\beta = \emptyset$, i.e., $\Phi_\beta \neq \Psi_\beta$. Indeed, a low-level user may infer the occurrence of $f \in A_H$ after fac has occurred.*

Here we briefly discuss the complexity of the proposed SNNI analysis method. Let x be the number of reachable markings of the LPNS. The cardinality of the set of nodes in the BRG and the number of reachable markings of the low-level subnet system are both at most x. It has been proved that the number of basis markings (i.e., the number of nodes in the BRG), in most cases, is much less than the number of reachable markings (see e.g. [25]). By Algorithms 2 and 3, the number of nodes in the UBRG and the SV are at most equal to $2x$ and $2x^2$, respectively. According to Algorithm 3 and Theorem 1, to verify the property of SNNI, we need to check all the leaf nodes in the SV, whose number is at most equal to $2x^2$. Therefore, the overall complexity of the proposed method is $\mathcal{O}(x^2)$.

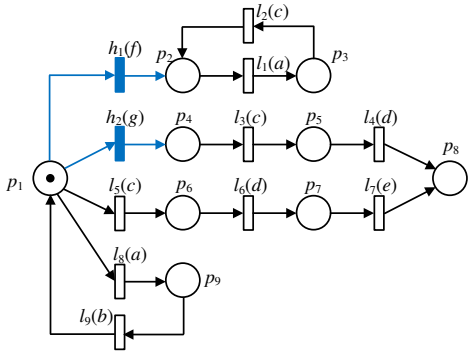

Figure 6: A labeled Petri net system.

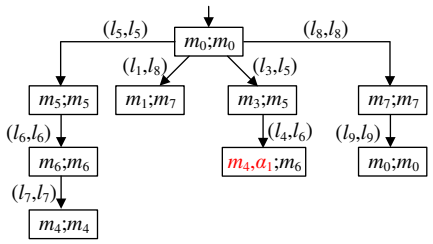

Figure 7: SV G_V of the PN system in Fig. 6.

Remark 2. *The problem of SNNI analysis of LPNSs is solved also in [17]. To avoid computing the reachability graph, authors in [18] solve some integer linear programming (ILP) problems. However, the number of constraints may be quite high and cannot be a priori estimated based on the net structure and the number of tokens. In more detail, for a bounded but non-live system, the number of constraints depends on the dimension of the reachability graph. For a bounded and live system, such a number depends on the initial marking and on the number of T-invariants. As a result, the computational efficiency of our method is in general higher than the ILP-based methods.*

5. Conclusions

This paper focuses on the problem of strong non-deterministic non-interference (SNNI) analysis of labeled Petri nets. We propose a necessary and sufficient condition for SNNI based on a special automaton called SNNI Verifier. Our future work will be twofold. First, we will deal with the problem of SNNI enforcement, i.e., design a supervisory control policy to enforce SNNI to a system. Second, we plan to extend the proposed methods to more properties related to information security of Petri nets such as bisimulation strong non-deterministic non-interference (BSNNI).

References

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

[2] X. Yin, Z. Li, W. Wang, and S. Li. Infinite-step opacity and K-step opacity of stochastic discrete-event systems. *Automatica*, 99:266–274, 2019.

[3] X. Yin and S. Lafortune. A new approach for the verification of infinite-step and K-step opacity using two-way observers. *Automatica*, 80:162–171, 2017.

[4] Y. Ji, X. Yin, and S. Lafortune. Enforcing opacity by insertion functions under multiple energy constraints. *Automatica*, 108:108476, 2019.

[5] K. Zhang, X. Yin, and M. Zamani. Opacity of nondeterministic transition systems: a (bi)Simulation relation approach. *IEEE Transactions on Automatic Control*, 64(12):5116–5123, 2019.

[6] Y. Ji, X. Yin, and S. Lafortune. Opacity enforcement using nondeterministic publicly known edit functions. *IEEE Transactions on Automatic Control*, 64(10):4369–4376, 2019.

[7] Z. Ma, X. Yin, and Z. Li. Verification and enforcement of strong infinite- and k-step opacity using state recognizers. *Automatica*, 133:109838, 2021.

[8] S. Lafortune, F. Lin, and C.N. Hadjicostis. On the history of diagnosability and opacity in discrete event systems. *Annual Reviews in Control*, 45:257–266, 2018. ISSN 1367-5788.

[9] X. Cong, M.P. Fanti, A.M. Mangini, and Z. Li. On-line algorithm for current state opacity enforcement in a Petri net framework. *IFAC-PapersOnLine*, 51(7):349–354, 2018.

[10] R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer security*, 3(1):331–396, 1995.

[11] N. Busi and R. Gorrieri. A survey on non-interference with Petri nets. In *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*, pages 328–344. Springer, 2004.

[12] E. Best, P. Darondeau, and R. Gorrieri. On the decidability of non-interference over unbounded Petri nets. In *SecCo : International Workshop on Security Issues in Concurrency*, pages 16–33, Paris, France, 2010.

[13] L. Bernardinello, G. Kilinc, and L. Pomello. Non-interference notions based on reveals and excludes relations for Petri nets. In *Transactions on Petri Nets and Other Models of Concurrency XI*, pages 49–70. Springer, 2016.

[14] F. Basile and G. De Tommasi. Non-interference assessment in bounded Petri nets via Integer Linear Programming. In *2018 Annual American Control Conference (ACC)*, pages 3056–3061, 2018.

[15] F. Basile and G. De Tommasi. Assessment of bisimulation non-interference in discrete event systems modelled with bounded Petri nets. *IEEE Control Systems Letters*, 5(4):1151–1156, 2021.

[16] F. Basile, G. De Tommasi, and C. Sterle. Noninterference enforcement via supervisory control in bounded Petri nets. *IEEE Transactions on Automatic Control*, 66(8):3653–3666, 2021.

[17] F. Basile, M. Boccia, G. De Tommasi, C. Motta, and C. Sterle. An optimization-based approach to assess non-interference in labeled and bounded Petri net systems. *Nonlinear Analysis: Hybrid Systems*, 44: 101153, 2022.

[18] N. Ran, J. Nie, A. Meng, and C. Seatzu. Non-Interference analysis of bounded Petri nets using basis reachability graph. *IEEE Transactions on Automatic Control*, pages 1–7, 2024.

[19] N. Ran, J. Hao, Z. He, M. Franceschelli, and C. Seatzu. Bisimulation non-interference analysis of bounded Petri nets. In *2024 International Conference on Control, Decision and Information Technologies (CoDIT)*, 2024.

[20] M.K. Reiter and A.D. Rubin. Crowds: anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, nov 1998.

[21] V. Shmatikov. Probabilistic analysis of an anonymity system. *Journal of Computer Security*, 12(3):355–377, 2004.

[22] A. Giua, C. Seatzu, and D. Corona. Marking estimation of Petri nets with silent transitions. *IEEE Transactions on Automatic Control*, 52 (9):1695–1699, 2007.

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

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

[25] N. Ran, H. Su, A. Giua, and C. Seatzu. Codiagnosability analysis of bounded Petri nets. *IEEE Transactions on Automatic Control*, 63(4): 1192–1199, 2018.

[26] N. Ran, J. Hao, and C. Seatzu. Prognosability analysis and enforcement of bounded labeled Petri nets. *IEEE Transactions on Automatic Control*, 67(10):5541–5547, 2022.

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

[28] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems, 2nd edition*. Springer New York, NY, 2008.