Supervisory Control of Petri Nets with Language Specifications*

Alessandro Giua
Dip. di Ing. Elettrica ed Elettronica, Università di Cagliari, Italy
Email: giua@diee.unica.it

Abstract

In this chapter we discuss how Petri nets can be used in the framework of supervisory control theory. A discrete event system is defined in such a theory as a language generator: this motivates the need to start the chapter with a short but self-contained introduction to Petri net languages. We consider the monolithic supervisory design that requires to construct the concurrent composition of the plant with the specification, to check this structure for controllability and nonblockingness, and eventually to refine it. We show how Petri nets can be used within this approach and show that while the procedure can always be applied to bounded nets, in the case of unbounded Petri nets it may not be possible to obtain a Petri net supervisor.

NOTE: A correction to the notion of "uncontrollable marking" given in this paper as Definition 7, and a modified proof of Theorem 3 that takes into account the corrected notion, can be found in:
B. Lacerda, P.U. Lima, "On the Notion of Uncontrollable Marking in Supervisory Control of Petri Nets," *IEEE Trans. on Automatic Control*, Vol. 59, No. 11, pp. 3069-3074, Nov. 2014.

*Published as: A. Giua, "Supervisory control of Petri nets with language specifications," in *Control of Discrete-Event Systems: Automata and Petri Net Perspectives*, C. Seatzu, M. Silva, J.H. van Schuppen (Eds), Lecture Notes in Control and Information Science, Vol. 433, pp. 235-256, Springer, 2013.

1 Introduction

In this chapter, we study Petri nets (PNs) as language generators and we show how PNs can be used for supervisory control of discrete event systems under language specifications.

Supervisory control, originated by the work of Ramadge and Wonham [13], is a system theory approach that has been gaining increasing importance because it provides a unifying framework for the control of Discrete Event Systems (DESs). [1]

In the original work of Ramadge and Wonham finite state machines (FSMs) were used to model plants and specifications. FSMs provide a general framework for establishing fundamental properties of DES control problems. They are not convenient models to describe complex systems, however, because of the large number of states that have to be introduced to represent several interacting subsystems, and because of the lack of structure. More efficient models have been proposed in the DES literature. Here the attention will be drawn to Petri net models.

PNs have several advantages over FSMs. Firstly, PNs have a higher language complexity than FSM, since Petri net languages are a proper superset of regular languages. Secondly, the states of a PN are represented by the possible markings and not by the places: thus they give a compact description, i.e., the structure of the net may be maintained small in size even if the number of the markings grows[2]. Thirdly, PNs can be used in modular synthesis, i.e., the net can be considered as composed of interrelated subnets, in the same way as a complex system can be regarded as composed of interacting subsystems.

Although PNs have a greater modeling power than FSMs, computability theory shows that the increase of modeling power often leads to an increase in the computation required to solve problems. This is why a section of this paper focuses on the decidability properties of Petri nets by studying the corresponding languages: note that some of these results are original and will be presented with formal proofs. It will be shown that Petri nets represent a good trade-off between modeling power and analysis capabilities

The chapter is structured as follows. In Section 1 Petri net generators and languages are defined. In Section 2 the concurrent composition operator on languages is defined and extended to an operator on generators. In Section 3 it is shown how the classical monolithic supervisory design can be carried out using Petri net models. Finally, in Section 4 some issues arising from the use of unbounded PNs in supervisory control are discussed.

2 Petri Nets and Formal Languages

This section provides a short but selfstanding introduction to Petri net languages. PN languages represents an interesting topic within the broader domain of formal language theory but there are few books devoted to this topic and the relevant material is scattered in several journal publications. In this section and in the following we focus on the definition of Petri net generators and operators that will later be used to solve a supervisory control problem.

2.1 Petri Net Generators

Definition 1. A *labeled Petri net system* (or *Petri net generator*) [7, 12] is a quadruple $\boldsymbol{G} = (N, \ell, \mathbf{m}_0, F)$ where:

[1] A general overview of Supervisory Control has been presented in Chapter 3 of *Control of Discrete-Event Systems: Automata and Petri Net Perspectives*, C. Seatzu, M. Silva, J.H. van Schuppen (Eds), Lecture Notes in Control and Information Science, Vol. 433, pp. 235-256, Springer, 2013.

[2] However, we should point out that many analysis techniques for Petri nets are based on the construction of the reachability graph, that suffers from the same state explosion problem typical of automata. To take advantage of the compact PN representation, other analysis techniques (e.g. structural) should be used.

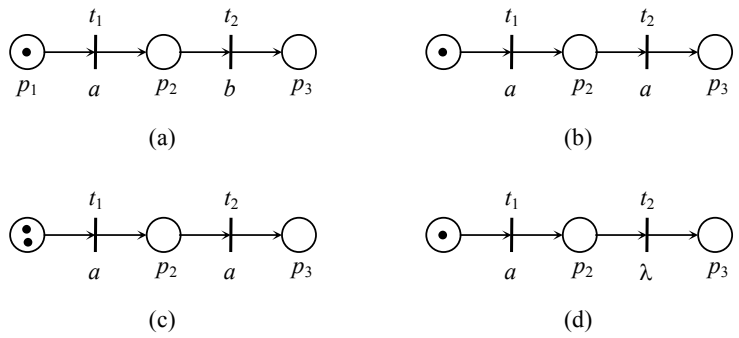

Figure 1: PN generators of Example 1

- $N = (P, T, \mathbf{Pre}, \mathbf{Post})$ is a *Petri net structure* with $|P| = m$ and $|T| = n$;
- $\ell : T \to E \cup \{\lambda\}$ is a *labeling function* that assigns to each transition a label from the alphabet of events E or assigns the empty word[3] λ as a label;
- $\mathbf{m}_0 \in \mathbb{N}^n$ is an *initial marking*;
- $F \subset \mathbb{N}^n$ is a finite set of *final markings.*

Three different types of *labeling functions* are usually considered.

- *Free labeling:* all transitions are labeled distinctly and none is labeled λ, i.e., $(\forall t, t' \in T)\ [t \neq t' \Longrightarrow \ell(t) \neq \ell(t')]$ and $(\forall t \in T)\ [\ell(t) \neq \lambda]$.
- λ*-free labeling:* no transition is labeled λ.
- *Arbitrary labeling:* no restriction is posed on ℓ.

The labeling function may be extended to a function $\ell : T^* \to E^*$ defining: $\ell(\lambda) = \lambda$ and $(\forall t \in T, \forall \sigma \in T^*)\ \ell(\sigma t) = \ell(\sigma)\ell(t)$.

Example 1. Consider the nets in Fig. 1 where the label of each transition is shown below the transition itself. Net (a) is a free-labeled generator on alphabet $E = \{a, b\}$. Nets (b) and (c) are λ-free generators on alphabet $E = \{a\}$. Net (d) is an arbitrary labeled generator on alphabet $E = \{a\}$. ■

Three languages are associated with a generator $\boldsymbol{G}$ depending on the different notions of terminal strings.

- *L-type* or *terminal language:*[4] the set of strings generated by firing sequences that reach a final marking, i.e.,
$$L_L(\boldsymbol{G}) = \{\ell(\sigma) \mid \mathbf{m}_0\ [\sigma\rangle\ \mathbf{m}_f \in F\}.$$
- *G-type* or *covering language* or *weak language:* the set of strings generated by firing sequences that reach a marking $\mathbf{m}$ covering a final marking, i.e.,
$$L_G(\boldsymbol{G}) = \{\ell(\sigma) \mid \mathbf{m}_0\ [\sigma\rangle\ \mathbf{m} \geq \mathbf{m}_f \in F\}.$$
- *P-type* or *prefix language:*[5] the set of strings generated by any firing sequence, i.e.,
$$L_P(\boldsymbol{G}) = \{\ell(\sigma) \mid \mathbf{m}_0\ [\sigma\rangle\ \}.$$

[3]While in other parts of this book the empty string is denoted ε, in this section we have chosen to use the symbol λ for consistency with the literature on PN languages.

[4]This language is called marked behavior in the framework of Supervisory Control and is denoted $L_m(\boldsymbol{G})$.

[5]This language is called closed behavior in the framework of Supervisory Control and is denoted $L(\boldsymbol{G})$.

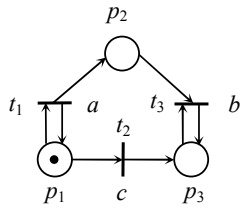

Figure 2: Free-labeled generator $\boldsymbol{G}$ of Example 2

Example 2. Consider the free-labeled generator $\boldsymbol{G}$ in Fig. 2. The initial marking, also shown in the figure, is $\mathbf{m}_0 = [1\ 0\ 0]^T$. Assume the set of final markings is $F = \{[0\ 0\ 1]^T\}$. The languages of this generator are:

$$L_L(\boldsymbol{G}) = \{a^m c b^m \mid m \geq 0\};$$

$$L_G(\boldsymbol{G}) = \{a^m c b^n \mid m \geq n \geq 0\};$$

$$L_P(\boldsymbol{G}) = \{a^m \mid m \geq 0\} \cup \{a^m c b^n \mid m \geq n \geq 0\}.$$

■

2.2 Deterministic generators

A *deterministic* PN generator [7] is such that the word of events generated from the initial marking uniquely determines the marking reached.

Definition 2. A λ-free generator $\boldsymbol{G}$ is *deterministic* iff for all $t, t' \in T$, with $t \neq t'$, and for all $\mathbf{m} \in R(N, \mathbf{m}_0)$: $\mathbf{m}\,[t\rangle \wedge \mathbf{m}\,[t'\rangle \Longrightarrow \ell(t) \neq \ell(t')$.

According to the previous definition, in a deterministic generator two transitions sharing the same label may never be simultaneously enabled and no transition may be labeled by the empty string. Note that a free-labeled generator is also deterministic. On the contrary, a λ-free (but not free labeled) generator may be deterministic or not depending on its structure and also on its initial marking.

Example 3. Consider generators (b) and (c) in Fig. 1: they have the same net structure and the same λ-free labeling, but different initial marking. The first one is deterministic, because transitions t_1 and t_2, sharing label a can never be simultaneously enabled. On the contrary, the second one is not deterministic, because reachable marking $[1\ 1\ 0]^T$ enables both transitions t_1 and t_2: as an example, the observed word aa may be produced by two different sequences yielding two different markings

$$\begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} [t_1\rangle \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} [t_1\rangle \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix} \qquad or \qquad \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} [t_1\rangle \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} [t_2\rangle \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

■

The previous definition of determinism was introduced in [18] and used in [7, 12]. It may be possible to extend it as follows.

Definition 3. A λ-free generator $\boldsymbol{G}$ is *deterministic* iff for all $t, t' \in T$, with $t \neq t'$, and for all $\mathbf{m} \in R(N, \mathbf{m}_0)$: $\mathbf{m}\,[t\rangle \wedge \mathbf{m}\,[t'\rangle \Longrightarrow [\ell(t) \neq \ell(t')] \vee [\mathbf{Post}(\cdot, t) - \mathbf{Pre}(\cdot, t) = \mathbf{Post}(\cdot, t') - \mathbf{Pre}(\cdot, t')]$.

$$\begin{array}{ccccccc}
\mathcal{L}^f & \rightarrow & \mathcal{L}_d & \rightarrow & \mathcal{L} & \rightarrow & \mathcal{L}^\lambda \\
 & & & & \uparrow & & \uparrow \\
\mathcal{G}^f & \rightarrow & \mathcal{G}_d & \rightarrow & \mathcal{G} & \rightarrow & \mathcal{G}^\lambda \\
\uparrow & & \uparrow & & \uparrow & & \uparrow \\
\mathcal{P}^f & \rightarrow & \mathcal{P}_d & \rightarrow & \mathcal{P} & \rightarrow & \mathcal{P}^\lambda
\end{array}$$

Table 1: Known relations among classes of Petri net languages. An arc $\rightarrow$ represents the set inclusion

With this extended definition, we accept as deterministic a generator in which two transitions with the same label may be simultaneously enabled at a marking $\mathbf{m}$, provided that the two markings reached from $\mathbf{m}$ by firing t and t' are the same. Note that with this extended definition, while the word of events generated from the initial marking uniquely determines the marking reached it does not necessarily uniquely determines the sequences that has fired.

2.3 Classes of Petri Net Languages

The classes of Petri net languages are denoted as follows.

- $\mathcal{L}^f$ (resp. $\mathcal{G}^f$, $\mathcal{P}^f$) denotes the class of terminal (resp. covering, prefix) languages generated by free-labeled PN generators.
- $\mathcal{L}_d$ (resp. $\mathcal{G}_d$, $\mathcal{P}_d$) denotes the class of terminal (resp. covering, prefix) languages generated by deterministic PN generators.
- $\mathcal{L}$ (resp. $\mathcal{G}$, $\mathcal{P}$) denotes the class of terminal (resp. covering, prefix) languages generated by λ-free PN generators.
- $\mathcal{L}^\lambda$ (resp. $\mathcal{G}^\lambda$, $\mathcal{D}^\lambda$, $\mathcal{P}^\lambda$) denotes the class of terminal (resp. covering, prefix) languages generated by arbitrary labeled PN generators.

Fig. 1 shows the relationship among these classes. Here $A \rightarrow B$ represents a strict set inclusion $A \subsetneq B$.

While a formal proof of all these relations can be found in [1], we point out that the relations on each line — that compare the same type of languages of nets with different labeling — are rather intuitive. Additionally, one readily understand that any *P-type* language of a generator $\boldsymbol{G}$ may also be obtained as a *G-type* language defining as set of final markings $F = \{\vec{0}\}$.

Parigot and Peltz [10] have defined PN languages as regular languages with the additional capability of determining if a string of parenthesis is well formed.

If we consider the class $\mathcal{L}$ of PN languages, it is possible to prove [12] that $\mathcal{L}$ is a strict superset of regular languages and a strict subset of context-sensitive languages. Furthermore $\mathcal{L}$ and the class of context-free languages are not comparable. An example of a language in $\mathcal{L}$ that is not context-free: $L = \{a^m b^m c^m \mid m \geq 0\}$. An example of a language that is context-free but is not in $\mathcal{L}$: $L = \{ww^R \mid w \in E^*\}$[6] if $|E| > 1$.

All these results are summarized in Fig. 3. Note that the class $\mathcal{L}_d$, although contained in $\mathcal{L}$, occupies the same position of $\mathcal{L}$ in the hierarchy shown in the figure.

In the framework of Supervisory Control, we will assume that the generators considered are deterministic. In particular, class $\mathcal{L}_d$ (or possibly $\mathcal{G}_d$ for unbounded nets) will be used to describe marked languages, while class $\mathcal{P}_d$ will be used to describe closed languages. There are several reasons for this choice.

- Systems of interest in supervisory control theory are deterministic.

[6]The string w^R is the reversal of string w.

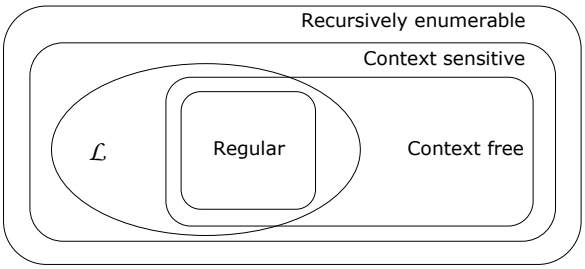

Figure 3: Relations among the class $\mathcal{L}$ and other classes of formal languages

- Although each class of deterministic languages here defined is strictly included in the corresponding class of λ-free languages, it is appropriate to restrict our analysis to deterministic generators. In fact, several properties of interest are decidable for deterministic nets while they are not for λ-free nets [1, 11, 18]
- In [1] it was shown that the classes $\mathcal{G}_d$ and $\mathcal{L}_d$ are incomparable, and furthermore $\mathcal{G}_d \cap \mathcal{L}_d = \mathcal{R}$, where $\mathcal{R}$ is the class of regular languages. Hence taking also into account the G-type language (in addition to the L-type language) one extends the class of control problems that can be modeled by deterministic unbounded PNs.

2.4 Other classes of Petri net languages

Gaubert and Giua [1] have explored the use of infinite sets of final markings in the definition of the marked behavior of a net. With each more or less classical subclass of subsets of $\mathbb{N}^m$ — finite, ideal (or upper), semi-cylindrical, star-free, recognizable, rational (or semilinear) subsets — it is possible to associate the class of Petri net languages whose set of accepting states belongs to the class.

When comparing the related Petri net languages, it was shown that for arbitrary or λ-free PN generators, the above hierarchy collapses: one does not increase the generality by considering semilinear accepting sets instead of the usual finite ones. However, for free-labeled and deterministic PN generators, it is shown that one gets new distinct subclasses of Petri net languages, for which several decidability problems become solvable.

3 Concurrent Composition and System Structure

In this section we recall the definition of the concurrent composition operator on languages and introduce the corresponding operator on nets.

Definition 4 (Concurrent composition of languages)**.** Given two languages $L_1 \subseteq E_1^*$ and $L_2 \subseteq E_2^*$, their *concurrent composition* is the language L on alphabet $E = E_1 \cup E_2$ defined as follows:

$$L = L_1 \parallel L_2 = \{\ w \in E^*\ \mid\ w \uparrow_{E_1} \in L_1,\ w \uparrow_{E_2} \in L_2\ \}$$

where $w \uparrow_{E_i}$ denotes the projection of word w on alphabet E_i, for $i = 1, 2$.

We now consider the counterpart of this language operator on a net structure.

Definition 5 (Concurrent composition of PN generators)**.** Let $\boldsymbol{G}_1 = (N_1, \ell_1, \mathbf{m}_{0,1}, F_1)$ and $\boldsymbol{G}_2 = (N_2, \ell_2, \mathbf{m}_{0,2}, F_2)$ be two PN generators. Their *concurrent composition*, denoted also $\boldsymbol{G} = \boldsymbol{G}_1 \parallel \boldsymbol{G}_2$, is the generator $\boldsymbol{G} = (N, \ell, \mathbf{m}_0, F)$ that generates $L_L(\boldsymbol{G}) = L_L(\boldsymbol{G}_1) \parallel L_L(\boldsymbol{G}_2)$ and $L_P(\boldsymbol{G}) = L_P(\boldsymbol{G}_1) \parallel L_P(\boldsymbol{G}_2)$.

The structure of $\boldsymbol{G}$ may be determined with the following procedure.

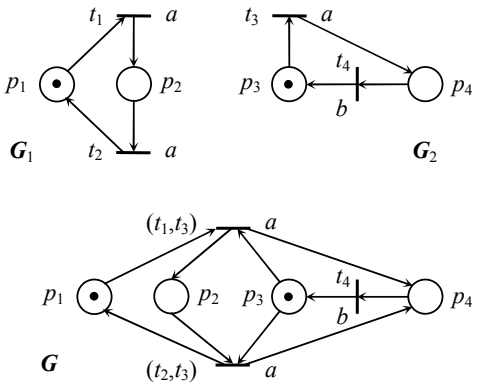

Figure 4: Two generators $\boldsymbol{G}_1$, $\boldsymbol{G}_2$ and their concurrent composition $\boldsymbol{G}$ of Example 4

Algorithm 1. *Let P_i, T_i and E_i $(i = 1, 2)$ be the place set, transition set, and the alphabet of $\boldsymbol{G}_i$.*

- *The place set P of N is the union of the place sets of N_1 and N_2, i.e., $P = P_1 \cup P_2$.*
- *The transition set T of N and the corresponding labels are computed as follows.*
  - *For each transition $t \in T_1 \cup T_2$ labeled λ, a transition with the same input and output bag of t and labeled λ belongs to T.*
  - *For each transition $t \in T_1 \cup T_2$ labeled $e \in (E_1 \setminus E_2) \cup (E_2 \setminus E_1)$, a transition with the same input and output bag of t and labeled e belongs to T.*
  - *Consider a symbol $e \in E_1 \cap E_2$ and assume it labels m_1 transitions $T_{e,1} \subseteq T_1$ and m_2 transitions $T_{e,2} \subseteq T_2$. Then $m_1 \times m_2$ transitions labeled e belong to T. The input (output) bag of each of these transitions is the sum of the input (output) bags of one transition in $T_{e,1}$ and of one transition in $T_{e,2}$.*
- $\mathbf{m}_0 = [\mathbf{m}_{0,1}^T \ \mathbf{m}_{0,2}^T]^T$.
- *F is the cartesian product of F_1 and F_2, i.e., $F = \{[\mathbf{m}_1^T \ \mathbf{m}_2^T]^T \mid \mathbf{m}_1 \in F_1, \mathbf{m}_2 \in F_2\}$.*

The composition of more than two generators can be computed by repeated application of the procedure. Note that while the set of places grows linearly with the number of composed systems, the set of transitions and of final markings may grow faster.

Example 4. Let $\boldsymbol{G}_1 = (N_1, \ell_1, \mathbf{m}_{0,1}, F_1)$ and $\mathbf{G_2} = (N_2, \ell_2, \mathbf{m}_{0,2}, F_2)$ be the two generators shown in Fig. 4. Here $F_1 = \{[1\ 0]^T\}$ and $F_2 = \{[1\ 0]^T, [0\ 1]^T\}$. Their concurrent composition $\boldsymbol{G} = \boldsymbol{G}_1 \parallel \boldsymbol{G}_2$ is also shown in Fig. 4. The initial marking of $\boldsymbol{G}$ is $\mathbf{m}_0 = [1\ 0\ 1\ 0]^T$ and its set of final markings is $F = \{[1\ 0\ 1\ 0]^T, [1\ 0\ 0\ 1]^T\}$.

■

4 Supervisory Design Using Petri Nets

In this section we discuss how Petri net models may be used to design supervisors for language specifications within the framework of Supervisory Control. The design of a supervisor in the framework of automata was presented in Chapter **??** and we assume the reader is already familiar with this material.

4.1 Plant, specification and supervisor

Here we comment some of the assumptions that are peculiar to the PN setting.

- The **plant** is described by a deterministic PN generator $\boldsymbol{G}$ on alphabet E. Its closed language is $L(\boldsymbol{G}) = L_P(\boldsymbol{G})$ while its marked[7] language is $L_m(\boldsymbol{G}) = L_L(\boldsymbol{G})$. We assume such a generator is *nonblocking*, i.e., $\overline{L_L(\boldsymbol{G})} = L_P(\boldsymbol{G})$.

  The transition set of $\boldsymbol{G}$ is partitioned as follows $T = T_c \cup T_{uc}$, where T_c are the *controllable transitions* that can be disabled by a control agent, while T_{uc} are the *uncontrollable transitions*. Note that this allows a generalization of the automata settings where the notion of controllability and uncontrollability is associated to the events. In fact, it is possible that two transitions, say t' and t'', have the same event label $\ell(t') = \ell(t'') = e \in E$ but one of them is controllable while the other one is not. In the rest of the chapter, however, we will not consider this case and assume that the event alphabet may be partitioned as $E = E_c \cup E_{uc}$ where

$$E_c = \bigcup_{t \in T_c} \ell(t), \quad E_{uc} = \bigcup_{t \in T_{uc}} \ell(t) \quad \text{and} \quad E_c \cap E_{uc} = \emptyset.$$

  It is also common to consider plants composed by m PN generators $\boldsymbol{G}_1, \ldots, \boldsymbol{G}_m$ working concurrently. The alphabets of these generators are $E_1, \ldots, E_m$. The overall plant is a PN generator $\boldsymbol{G} = \boldsymbol{G}_1 \parallel \cdots \parallel \boldsymbol{G}_m$ on alphabet $E = E_1 \cup \cdots \cup E_m$.

- The **specification** is a language $K \subset \hat{E}^*$, where $\hat{E} \subset E$ is a subset of the plant alphabet. Such a specification defines a set of *legal words* on E given by $\{w \in E^* \mid w \uparrow_{\hat{E}} \in \text{prefix}(K)\}$.

  The specification K is represented by a deterministic nonblocking PN generator $\boldsymbol{H}$ on alphabet $\hat{E}$ whose marked language is $L_m(\boldsymbol{H}) = L_L(\boldsymbol{H}) = K$. As for the plant, other choices for the marked language are possible.

- The **supervisor** is described by a nonblocking PN generator $\boldsymbol{S}$ on alphabet E. It runs in parallel with the plant, i.e., each time the plant generates an event e a transition with the same label is executed on the supervisor. The control law computed by $\boldsymbol{S}$ when its marking is $\mathbf{m}$ is given by $g(\mathbf{m}) = E_{uc} \cup \{e \in E_c \mid (\exists t \in T_c)\mathbf{m}[t\rangle, \ell(t) = e\}$.

4.2 Monolithic Supervisor Design

The *monolithic supervisory design* requires three steps. In the first step, a coarse structure for a supervisor is synthesized by means of concurrent composition of the plant and specification. In the second step, the structure is analyzed to check if properties of interest (namely, the absence of uncontrollable and blocking states) hold. In the third step, if the properties do not hold, this structure is trimmed to avoid reaching undesirable states.

Algorithm 2 (Monolithic supervisory design)**.** *We are given a plant $\boldsymbol{G}$ and a specification $\boldsymbol{H}$.*

1. *Construct by concurrent composition the generator $\boldsymbol{J} = \boldsymbol{G} \parallel \boldsymbol{H}$.*
2. *Determine if the generator $\boldsymbol{J}$ satisfies the following properties:*
   - nonblockingness*, i.e., it does not contain blocking markings from which a final marking cannot be reached;*
   - controllability*, i.e., it does not contain uncontrollable markings such that when $\boldsymbol{G}$ and $\boldsymbol{H}$ run in parallel an uncontrollable event is enabled in $\boldsymbol{G}$ but is not enabled in $\boldsymbol{H}$.*

   If $\boldsymbol{J}$ satisfies both properties, then both $\boldsymbol{H}$ and $\boldsymbol{J}$ are suitable supervisors.

[7] While in the case of bounded nets the L-type language can describe any marked language, in the case of unbounded generators other choices for the marked language are possible considering the G-type language of the generator or even any other type of terminal languages as mentioned in § 2.4. This will be discussed in Section 5.

3. *If $\boldsymbol{J}$ contains blocking or uncontrollable markings, we have to trim it to obtain a nonblocking and controllable generator $\boldsymbol{S}$. The generator $\boldsymbol{S}$ obtained through this procedure is at the same time a suitable maximally permissive supervisor and the corresponding closed-loop system.*

In the previous algorithm, the generator $\boldsymbol{J}$ constructed in step 1 represents the largest behavior of the plant that satisfies all the constraints imposed by the specifications. More precisely, its closed language

$$L(\boldsymbol{J}) = \{w \in E \mid w \in L(\boldsymbol{G}), w \uparrow_{\hat{E}} \in L(\boldsymbol{H})\}$$

represents the behavior of the plant restricted to the set of legal words, while its marked behavior

$$L_m(\boldsymbol{J}) = \{w \in E \mid w \in L_m(\boldsymbol{G}), w \uparrow_{\hat{E}} \in L_m(\boldsymbol{H})\}$$

represents the marked behavior of the plant restricted to the set of legal words marked by the specification.

In step 2 we have used informally the term "blocking marking" and "uncontrollable marking". We will formally define these notions in the following.

We first define some useful notation. The structure of the generators is $\boldsymbol{J} = (N, \ell, \mathbf{m}_0, F)$, $\boldsymbol{G} = (N_1, \ell_1, \mathbf{m}_{0,1}, F_1)$, and $\boldsymbol{H} = (N_2, \ell_2, \mathbf{m}_{0,2}, F_2)$, where $N = (P, T, \mathbf{Pre}, \mathbf{Post})$ and $N_i = (P_i, T_i, \mathbf{Pre}_i, \mathbf{Post}_i)$, $(i = 1, 2)$. We define the *projection of a marking* $\mathbf{m}$ *of* N *on net* N_i, $(i = 1, 2)$, denoted $\mathbf{m} \uparrow_i$, is the vector obtained from $\mathbf{m}$ by removing all the components associated to places not present in N_i.

We first present the notion of a blocking marking.

Definition 6. A marking $\mathbf{m} \in R(N, \mathbf{m}_0)$ of generator $\boldsymbol{J}$ is a *blocking marking* if no final marking may be reached from it, i.e., $R(N, \mathbf{m}) \cap F = \emptyset$. The generator $\boldsymbol{J}$ is *nonblocking* if no blocking marking is reachable.

We now present the notion of an uncontrollable marking.

Definition 7. Let $T_u \subseteq T$ be the set of uncontrollable transitions of $\boldsymbol{J}$. A marking $\mathbf{m} \in R(N, \mathbf{m}_0)$ of generator $\boldsymbol{J}$ is *uncontrollable* if there exists an uncontrollable transition $t \in T_u$ that is enabled by $\mathbf{m} \uparrow_1$ in $\boldsymbol{G}$ but that is not enabled by $\mathbf{m} \uparrow_2$ in $\boldsymbol{H}$. The generator $\boldsymbol{J}$ is *controllable* if no uncontrollable marking is reachable.

Determining if a generator $\boldsymbol{J}$ is nonblocking and controllable is always possible, as we will show in the next section. We also point out that for bounded nets this test can be done by construction of the reachability graph[8] as in the following example of supervisory design.

Example 5. Consider the generators $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$, and the specification $\boldsymbol{H}$ in Fig. 5 (left). Note that all nets are free-labeled, hence we have an isomorphism between the set of transitions T and the set of events E: in the following each transitions will be denoted by the corresponding event.

$\boldsymbol{G}_1$ describes a conveyor that brings in a manufacturing cell a raw part (event a) that is eventually picked-up by a robot (event b) so that a new part can enter. $\boldsymbol{G}_2$ describes a machine that is loaded with a raw part (event c) and, depending on the operation it performs, may produce parts of type A or type B (events d or e) before returning to the idle state. The set of final states of both generators consists of the initial marking shown in the figure.

The specification we consider, represented by the generator $\boldsymbol{H}$, describes a cyclic operation process where a robot picks-up a raw part from the conveyor, loads it on the machine

[8] As we have already pointed out, the construction of the reachability graph suffers from the state explosion problem. An open area for future research is the use of more efficient analysis techniques (e.g., structural) to check nonblockingness and controllability for language specification.

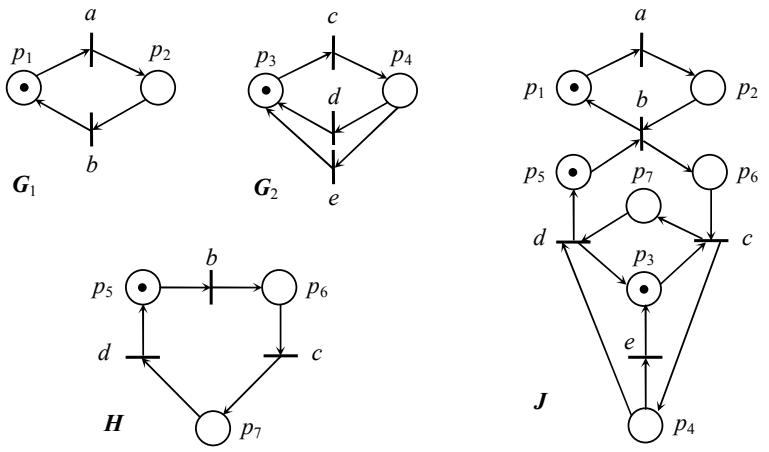

Figure 5: Left: Systems $\boldsymbol{G}_1, \boldsymbol{G}_2$ and specification $\boldsymbol{H}$ for the control problem of Example 5. Right: System $\boldsymbol{J} = \boldsymbol{G}_1 \parallel \boldsymbol{G}_2 \parallel \boldsymbol{H}$

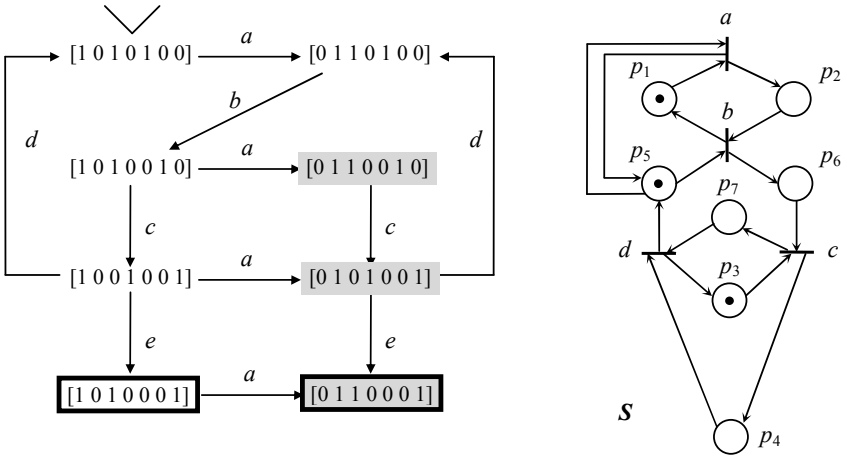

Figure 6: Left: Reachability graph of generator $\boldsymbol{J}$ of Example 5. Right: the structure of the trim generator **S** of Example 6

and after recognizing that a part of type A has been produced repeats the process. The set of final states consists of the initial marking shown in the figure.

The overall process is $\boldsymbol{G} = \boldsymbol{G}_1 \parallel \boldsymbol{G}_2$ and the generator $\boldsymbol{J} = \boldsymbol{G} \parallel \boldsymbol{H}$, is shown in Fig. 5 (right). Its set of final states consists of the initial marking shown in the figure.

Assume now that the controllable transition/event set is $E_c = \{a, c, d, e\}$ and the uncontrollable transition/event set is $E_u = \{b\}$.

It is immediate to show that generator $\boldsymbol{J}$ is blocking and uncontrollable. To show this we have constructed the reachability graph of $\boldsymbol{J}$ in Fig. 6. The two markings shown in thick boxes are blocking because from them it is impossible to reach the initial marking (that is also the unique final marking). The three markings shaded in grey are uncontrollable: in fact, in all these markings $m(p_2) = 1$, i.e., uncontrollable transition b is enabled in the plant $\boldsymbol{G}$, while $m(p_5) = 0$, i.e., b is not enabled in $\boldsymbol{H}$. ■

4.3 Trimming

Once the coarse structure of a candidate supervisor is constructed by means of concurrent composition, we need to trim it to obtain a nonblocking and controllable generator.

The next example shows the problems involved in the trimming of a net.

Example 6. Let us consider the generator $\boldsymbol{J}$ constructed in Example 5.

Refining the PN to avoid reaching the undesirable markings shown in Fig. 6 is complex. First, we could certainly remove the transition labeled by e since its firing always leads to an undesirable state and it is controllable. After removal of this transition, the transition labeled by a will be enabled by the following reachable markings: $\mathbf{m}' = [1\ 0\ 1\ 0\ 1\ 0\ 0]^T, \mathbf{m}'' = [1\ 0\ 1\ 0\ 0\ 1\ 0]^T, \mathbf{m}''' = [1\ 0\ 0\ 1\ 0\ 0\ 1]^T$. We want to block the transition labeled a when the markings $\mathbf{m}''$ and $\mathbf{m}'''$ are reached. Since

$$m'(p_5) = 1 > m''(p_5) = m'''(p_5) = 0,$$

we can add an arc from p_5 to a and from a to p_5 as in Fig. 6. ■

The following algorithm can be given for the trimming of a net.

Algorithm 3. *Let t be a transition to be controlled, i.e., a transition leading from an admissible marking to an undesirable marking. Let e be its label.*

1. *Determine the set of admissible reachable markings that enable t, and partition this set into the disjoint subsets $\mathcal{M}_a$ (the markings from which t should be allowed to fire), and $\mathcal{M}_{na}$ (the markings from which t should not be allowed to fire, to avoid reaching an undesirable marking). If $\mathcal{M}_a = \emptyset$ remove t and stop, else continue.*
2. *Determine a construct in the form:*

$$\begin{aligned}\mathcal{U}(\mathbf{m}) = \quad & [(m(p_1^1) \geq n_1^1) \wedge \ldots \wedge (m(p_{k1}^1) \geq n_{k1}^1)] \vee \\ & \ldots \\ & \vee [(m(p_1^l) \geq n_1^l) \wedge \ldots \wedge (m(p_{kl}^l) \geq n_{kl}^l)],\end{aligned}$$

   such that $\mathcal{U}(\mathbf{m}) =$ TRUE if $\mathbf{m} \in \mathcal{M}_a$, and $\mathcal{U}(\mathbf{m}) =$ FALSE if $\mathbf{m} \in \mathcal{M}_{na}$.
3. *Replace transition t with l transitions $t^1, \ldots, t^l$ labeled a. The input (output) arcs of transition t^j, $j = 1, \ldots, l$, will be those of transition t plus n_i^j arcs inputting from (outputting to) place p_i^j, $i = 1, \ldots, k_j$.*

It is clear that following this construction there is an enabled transition labeled e for any marking in $\mathcal{M}_a$, while none of these transitions are enabled by a marking in $\mathcal{M}_{na}$. We also note that in general several constructs of this form may be determined. The one which requires the minimal number of transitions, i.e., the one with the smallest l, is preferable.

The following theorem gives a sufficient condition for the applicability of the algorithm.

Theorem 1. *The construct of Algorithm 3 can always be determined if the net is bounded.*

Proof. For sake of brevity, we prove this result for the more restricted class of conservative nets. One should keep in mind, however, that given a bounded non conservative net, one can make the net conservative adding dummy sink places that do not modify its behavior.

A net is conservative if there exists an integer vector $Y > \vec{0}$ such that for any two markings $\mathbf{m}$ and $\mathbf{m}'$ reachable from the initial marking $Y^T\mathbf{m} = Y^T\mathbf{m}'$. Hence if $\mathbf{m} \neq \mathbf{m}'$ there exists a place p such that $m(p) > m'(p)$. Also the set of reachable markings is finite.

On a conservative net, consider $\mathbf{m}_i \in \mathcal{M}_a$, $\mathbf{m}_j \in \mathcal{M}_{na}$. We have that $\mathcal{M}_a$ and $\mathcal{M}_{na}$ are finite sets and also there exists a place p_{ij} such that $m_i(p_{ij}) = n_{ij} > m_j(p_{ij})$. Hence

$$\mathcal{U}(\mathbf{m}) = \bigvee_{i \in \mathcal{M}_a} \left[\bigwedge_{j \in \mathcal{M}_{na}} (m(p_{ij}) \geq n_{ij}) \right]$$

is a construct for Algorithm 3. □

Unfortunately, the construct may contain up to $|\mathcal{M}_a|$ *OR* clauses, i.e., up to $|\mathcal{M}_a|$ transitions may be substituted for a single transition to control. Note, however, that it is often possible to determine a simpler construct as in Example 6, where the construct for the transition labeled a was $\mathcal{U}(\mathbf{m}) = [m(p_5) \geq 1]$.

5 Supervisory control of unbounded PN generators

As we have seen in the previous section, the monolithic supervisory design presented in Algorithm 2 can always be applied when the plant $\boldsymbol{G}$ and the specification $\boldsymbol{H}$ are bounded PN generators. Here we consider the case of general, possibly unbounded, generators.

In step 1 of the monolithic supervisory design algorithm the unboundedness of the $\boldsymbol{G}$ or $\boldsymbol{H}$ does not require any special consideration, since the procedure to construct the concurrent composition $\boldsymbol{J} = \boldsymbol{G} \parallel \boldsymbol{H}$ is purely structural in the PN setting. Thus we need to focus on the last two steps, and discuss how it is possible to check if an unbounded generator $\boldsymbol{G}$ is nonblocking and controllable, and eventually how it can be trimmed.

We have previously remarked that in the case of bounded nets the L-type language can describe any marked language. In the case of unbounded generators other choices for the marked language are possible considering the G-type language of the generator or even any other type of terminal language mentioned in § 2.4.

In the rest of this section we will only consider two types of marked languages for a PN generator $\boldsymbol{G}$.

- *L-type language*, i.e., $L_m(\boldsymbol{G}) = L_L(\boldsymbol{G})$. This implies that the set of *marked markings* reached by words in $L_m(\boldsymbol{G})$ is $\mathcal{F} = F$, i.e., it coincides with the finite set of final markings associated to the generator.
- *G-type marked language*, i.e., $L_m(\boldsymbol{G}) = L_G(\boldsymbol{G})$. This implies that the set of *marked markings* reached by words in $L_m(\boldsymbol{G})$ is

$$\mathcal{F} = \bigcup_{\mathbf{m}_f \in F} \{\mathbf{m} \in \mathbb{N}^m \mid \mathbf{m} \geq \mathbf{m}_f\},$$

  i.e., it is the infinite covering set of F.

5.1 Checking nonblockingness

We will show in this subsection that checking a generator for nonblockingness is always possible.

Let us first recall the notion of home space.

Definition 8. A marking $\mathbf{m} \in \mathbb{N}^m$ of a Petri net is a *home-marking* if it is reachable from all reachable markings.

A set of markings $\mathcal{M} \subseteq \mathbb{N}^m$ of a Petri net is a *home space* if for all reachable marking $\mathbf{m}$ a marking in $\mathcal{M}$ is reachable from $\mathbf{m}$.

The following result is due to Johnen and Frutos Escrig.

Proposition 1 ([8])**.** *The property of being a home space for finite unions of linear sets*[9] *having the same periods is decidable.*

We can finally state the following original result.

Theorem 2. *Given a generator $\boldsymbol{J}$ constructed as in step 1 of Algorithm 2 it is decidable if it is nonblocking when its marked language is the L-type or G-type language.*

[9] We say that $\mathcal{E} \subseteq \mathbb{N}^m$ is a *linear set* if there exists some $\boldsymbol{v} \in \mathbb{N}^m$ and a finite set $\{\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n\} \subseteq \mathbb{N}^m$ such that $\mathcal{E} = \{\boldsymbol{v}' \in \mathbb{N}^m \mid \boldsymbol{v}' = \boldsymbol{v} + \sum_{i=1}^{n} k_i \boldsymbol{v}_i \text{ with } k_i \in \mathbb{N}\}$. The vector $\boldsymbol{v}$ is called the *base* of $\mathcal{E}$, and $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n$ are called its *periods*.

Proof. Let $\mathcal{F}$ be the set of marked markings of the generator. According to Definition 6 generator $\boldsymbol{J}$ is nonblocking iff from every reachable markings $\mathbf{m}$ a marked marking in $\mathcal{F}$ is reachable. Thus checking for nonblockingness is equivalent to checking if the set of marked markings $\mathcal{F}$ is a home space.

When the marked language is the L-type language, $\mathcal{F} = F$ and we observe that each marking $\mathbf{m}_f$ can be considered as a linear set with base $\mathbf{m}_f$ and empty set of generators.

When the marked language is the G-type language,

$$\mathcal{F} = \bigcup_{\mathbf{m}_f \in F} \{\mathbf{m} \in \mathbb{N}^m \mid \mathbf{m} \geq \mathbf{m}_f\} = \{\mathbf{m}_f + \sum_{i=1}^{m} k_i \boldsymbol{e}_i \mid \ k_i \in \mathbb{N}\}$$

where vectors $\boldsymbol{e}_i$ are the canonical basis vectors, i.e., $\boldsymbol{e}_i \in \{0,1\}^n$, with $e_i(i) = 1$ and $e_i(j) = 0$ if $i \neq j$.

In both cases $\mathcal{F}$ is the finite unions of linear sets having the same periods, hence checking if it is a home space is decidable by Proposition 1. □

5.2 Checking controllability

We will show in this subsection that checking a generator for controllability is always possible. The material presented in this subsection is original and proofs of all results will be given.

We first present some intermediate result.

Lemma 1. *Let $\langle N, \mathbf{m}_0 \rangle$ be a marked net with $N = (P, T, \mathbf{Pre}, \mathbf{Post})$ and $|P| = m$. Given a marking $\bar{\mathbf{m}} \in \mathbb{N}^m$ and a place $\bar{p} \in P$, we define the set*

$$\mathcal{S}(\bar{\mathbf{m}}, \bar{p}) = \{\mathbf{m} \in \mathbb{N}^m \mid m(\bar{p}) = \bar{m}(\bar{p}), \quad (\forall p \in P \setminus \{\bar{p}\})\ m(p) \geq \bar{m}(p)\}$$

of those markings that are equal to $\bar{\mathbf{m}}$ in component $\bar{p}$ and greater than or equal to $\bar{\mathbf{m}}$ in all other components.

Checking if a marking in this set is reachable in $\langle N, \mathbf{m}_0 \rangle$ is decidable.

Proof. To prove this result, we reduce the problem of determining if a marking in $\mathcal{S}(\bar{\mathbf{m}}, \bar{p})$ is reachable to the standard marking reachability problem of a modified net.

Consider in fact net $N' = (P', T', \mathbf{Pre}', \mathbf{Post}')$ obtained from N as follows. $P' = P \cup \{p_s, p_f\}$; $T' = T \cup \{t_f\} \cup \{t_p \mid \forall p \in P \setminus \{\bar{p}\}\}$. For $p \in P$ and $t \in T$ it holds $\mathbf{Pre}'(p,t) = \mathbf{Pre}(p,t)$ and $\mathbf{Post}'(p,t) = \mathbf{Post}(p,t)$, while the arcs incident on the newly added places and transitions are described in the following. Place p_s is self-looped with all transitions in T, i.e., $\mathbf{Pre}'(p_s,t) = \mathbf{Post}'(p_s,t) = 1$ for all $t \in T$. Place p_f is self-looped with all new transitions t_p, for all $p \in P \setminus \{\bar{p}\}$. Transition t_f has an input arc from place p_s and an output arc to place p_f; furthermore it has $\bar{m}(p)$ input arcs from any place $p \in P \setminus \{\bar{p}\}$. Finally, for all $p \in P \setminus \{\bar{p}\}$ transition t_p is a sink transition with a single input arc from place p.

We associate to N' an initial marking $\mathbf{m}'_0$ defined as follows: for all $p \in P$, $m'_0(p) = m_0(p)$, while $m'_0(p_s) = 1$ and $m'_0(p_f) = 0$. Such a construction is shown in Fig. 7 where the original net N with set of places $P = \{\bar{p}, p', \ldots, p''\}$ and set of transitions $T = \{t_1, ldots, t_n\}$ is shown in a dashed box. Arcs with starting and ending arrows represent self-loops.

We claim that a marking in the set $\mathcal{S}(\bar{\mathbf{m}}, \bar{p})$ is reachable in the original net if and only if marking $\mathbf{m}'_f$ is reachable in $\langle N', \mathbf{m}'_0 \rangle$, where $m'_f(\bar{p}) = \bar{m}(\bar{p})$, $m'_f(p_f) = 1$ and $m'_f(\bar{p}) = 0$ for $p \in P' \setminus \{\bar{p}, p_f\}$.

This can be proved by the following reasoning. The evolution of net N' before the firing of t_f mimics that of N. Transition t_f may only fire from a marking greater than or equal to $\bar{\mathbf{m}}$ in all components but eventually $\bar{p}$. After the firing of t_f, the transitions of the original net are blocked (p_s is empty) and only the sink transitions t_p, for all $p \in P \setminus \{\bar{p}\}$, may fire thus emptying the corresponding places. The only place whose markings cannot change after the firing of t_f is $\bar{p}$. □

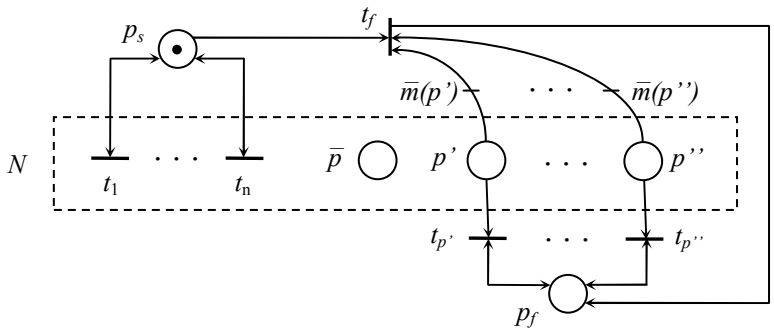

Figure 7: Construction of Lemma 1

Theorem 3. *Given a generator $\boldsymbol{J} = \boldsymbol{G} \parallel \boldsymbol{H}$ constructed as in step 1 of Algorithm 2 it is decidable if it is controllable.*

Proof. We will show that the set of uncontrollable markings to be checked can be written as the finite union of sets of the form $\mathcal{S}(\bar{\mathbf{m}}, \bar{p})$.

Given an uncontrollable transition $t \in T_{uc}$ let $P_G(t)$ (resp., $P_H(t)$) be the set of input places of t that belong to generator $\boldsymbol{G}$ (resp., $\boldsymbol{H}$). Consider now a place $p \in P_H(t)$ and an integer $k \in \{0, 1, \ldots, Pre(p,t) - 1\}$ and define the following marking $\mathbf{m}_{t,p,k}$ such that $m_{t,p,k}(p) = k$, $m_{t,p,k}(p') = Pre(p', t)$ if $p' \in P_G(t)$, else $m_{t,p,k}(p') = 0$. Clearly such a marking is uncontrollable because the places in $\boldsymbol{G}$ contain enough tokens to enable uncontrollable transition t while place p in $\boldsymbol{H}$ does not contain enough tokens to enable it. All other markings in $\mathcal{S}(\mathbf{m}_{t,p,k}, p)$ are equally uncontrollable.

Thus the overall set of uncontrollable markings to be checked can be written as the finite union

$$\bigcup_{t \in T_{uc}} \bigcup_{p \in P_H(t)} \bigcup_{k \in \{0,1,\ldots,Pre(p,t)-1\}} \mathcal{S}(\mathbf{m}_{t,p,k}, p)$$

and by Lemma 1 checking if an uncontrollable marking is reachable is decidable. □

5.3 Trimming a blocking generator

The problem of trimming a blocking net is the following: given a deterministic PN generator $\boldsymbol{G}$ with languages $L_m(G)$ and $L(G) \supset \overline{L_m(G)}$ one wants to modify the structure of the net to obtain a new DES G' such that $L_m(G') = L_m(G)$ and $L(G') = \overline{L_m(G')} = \overline{L_m(G)}$.

On a simple model such as a state machine this may be done, trivially, by removing all states that are reachable but not coreachable (i.e., no final state may be reached from them) and all their input and output edges.

On Petri net models the trimming may be more complex. If the Petri net is bounded, it was shown in the previous section how the trimming may be done without major changes of the net structure, in the sense that one has to add new arcs and eventually duplicate transitions without introducing new places. Here we discuss the general case of possibly unbounded nets.

When the marked language of a net is its L-type Petri net language, the trimming of the net is not always possible as will be shown by means of the following example.

Example 7. Let $\boldsymbol{G}$ be the deterministic PN generator in Fig. 8 (left), with $\mathbf{m}_0 = [1\ 0\ 0\ 0]^T$ and set of final markings $F = \{[0\ 0\ 0\ 1]^T\}$. The marked (L-type) and closed behaviors of this net are: $L_m(G) = \{a^m b a^m b \mid m \geq 0\}$ and $L(G) = \{\overline{a^m b a^n b} \mid m \geq n \geq 0\}$. The infinite reachability graph of this net is partially shown in Fig. 8 (right): here the unique final marking is shown in a box.

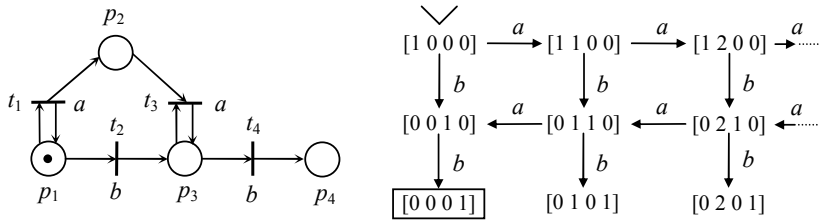

Figure 8: Left: Blocking net of Example 7: Right: Its labeled reachability graph

One sees that all markings of the form $[0\ k\ 0\ 1]^T$ with $k \geq 1$ are blocking. To avoid reaching a blocking marking one requires that p_2 be empty before firing the transition inputting into p_4. However, since p_2 is unbounded this cannot be done with a simple place/transition structure. ■

It is possible to prove formally that the prefix closure of the marked language of the net discussed in Example 7 is not a P-type Petri net language. The proof is based on the pumping lemma for P-type PN languages, given in [7].

Lemma 2. *(*Pumping lemma*). Consider a PN language $L \in \mathcal{P}$. Then there exist numbers k, l such that any word $w \in L$, with $\mid w \mid \geq k$, has a decomposition $w = xyz$ with $1 \leq \mid y \mid \leq l$ such that $xy^iz \in L, \forall i \geq 1$.*

Proposition 2. *Consider the L-type PN language $L' = \{a^m b a^m b \mid m \geq 0\}$. Its prefix closure $L = \overline{L'}$ is not a P-type Petri net language.*

Proof. Given k according to the pumping lemma, consider the word $w = a^k b a^k b \in L$. Obviously, there is no decomposition of this word that can satisfy the pumping lemma. □

When the marked language of a net is its G-type Petri net language, the trimming of the net is always possible because the prefix closure of such a language is a deterministic P-type Petri net language. This follows from next theorem, that provides an even stronger result.

Theorem 4. *[4] Given a deterministic PN generator $\boldsymbol{G} = (N, \ell, \mathbf{m}_0, F)$ with $\overline{L_G(\boldsymbol{G})} \subsetneq L_P(\boldsymbol{G})$, there exists a finite procedure to construct a new deterministic PN generator $\boldsymbol{G}'$ such that $L_G(\boldsymbol{G}') = L_G(\boldsymbol{G})$ and $L_P(\boldsymbol{G}') = \overline{L_G(\boldsymbol{G}')}$.*

5.4 Trimming an uncontrollable generator

In this section we show by means of an example that given a PN generator $\boldsymbol{J} = \boldsymbol{G} \parallel \boldsymbol{H}$ obtained by concurrent composition of a plant and of a specification, it is not always possible to trim it removing the uncontrollable markings.

Example 8. Consider a plant $\boldsymbol{G}$ described by the PN generator on the left of Fig. 8 (including the dashed transition and arcs). We are interested in the closed language of the net, so we will no specify a set of final markings F: all reachable markings are also final. We assume $T_{uc} = \{t_1, t_3, t_5\}$, i.e., $E_{uc} = \{a\}$.

Consider a specification $\boldsymbol{H}$ described by the PN generator on the left of Fig. 9 (excluding the dashed transition and arcs).

On the right of Fig. 9 we have represented the labeled reachability graph of $\boldsymbol{G}$ (including the dashed arcs labeled a on the bottom of the graph) and the labeled reachability graph of $\boldsymbol{H}$ (excluding the dashed arcs labeled a on the bottom of the graph). Now if we consider the concurrent composition $\boldsymbol{J} = \boldsymbol{G} \parallel \boldsymbol{H}$ and construct its labeled reachability graph, we obtain a graph isomorphic to the labeled graph of generator $\boldsymbol{H}$ (only the labeling of the nodes changes).

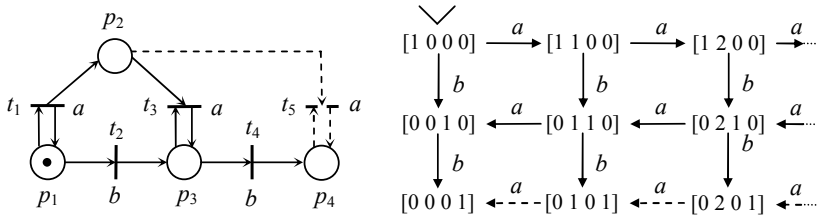

Figure 9: Left: Generators of Example 8. Right: Their labeled rechability graphs

All markings of the form $[0\ k\ 0\ 1]^T$ with $k \geq 1$ are uncontrollable: in fact, when the plant is in such a marking the uncontrollable transition t_5 labeled a is enabled, while no event labeled a is enabled on $\boldsymbol{J}$. If we remove all uncontrollable markings, we have a generator whose closed language is $L = \{\overline{a^m b a^m b} \mid m \geq 0\}$ that, however, as shown in Proposition 2, is not a P-type language. ■

Based on these results in [3] the following result was proven.

Theorem 5. The classes $\mathcal{P}_d$, $\mathcal{G}_d$ and $\mathcal{L}_d$ of PN languages are not closed under the supremal controllable sublanguage operator[10].

5.5 Final remarks

The results we have presented in this section showed that in the case of unbounded PN generators a supervisor may not always be represented as a PN. In fact, while it is always possible to check a given specification for nonblockingness and controllability — even in the case of generators with an infinite state space — when these properties are not satisfied the trim behavior of the closed loop system may not be represented as a net. A characterization of those supervisory control problems that admit PN supervisors is an area still open to future research.

6 Further readings

The book by Peterson [12] contains a good introduction to PN languages, while other relevant results can be found in [18, 10, 11, 7, 1, 16].

Many issues related to PNs as discrete event models for supervisory control have been discussed in the survey by Holloway *et al.* [5] and in the works of Giua and DiCesare [3, 4]. The existence of supervisory control policies that enforce liveness have been discussed by Sreenivas in [15, 17].

Finally, an interesting topic that has received much attention in recent years is the supervisory control of PNs under a special class of state specifications called *Generalized Mutual Exclusion Constraints* (GMECs) that can be enforced by controllers called *monitor places* [2]. Several monitor based techniques have been developed for the control of Petri nets with uncontrollable and unobservable transitions and good surveys can be found in [9, 6].

References

[1] Gaubert, S., Giua, A.: Petri net languages and infinite subsets of $\mathbb{N}^m$. In: Journal of Computer and System Sciences 59:373-391 (1999)

[10] See Chapter 3 for a formal definition of this operator.

[2] Giua, A., DiCesare, F., Silva, M.: Generalized Mutual Exclusion Constraints for Nets with Uncontrollable Transitions. In: Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (Chicago, USA), 974-799 (1992)

[3] Giua, A., DiCesare, F.: Blocking and controllability of Petri nets in supervisory control. In: IEEE Transactions on Automatic Control 39(4): 818-823 (1994)

[4] Giua, A., DiCesare, F.: Decidability and closure properties of weak Petri net languages in supervisory control. In: IEEE Transactions on Automatic Control 40(5):906-910 (1995)

[5] Holloway, L.E., Krogh, B.H., Giua, A.: A Survey of Petri Net Methods for Controlled Discrete Event Systems. In: Discrete Event Dynamic Systems, 7:151-190 (1997).

[6] Iordache, M.V., Antsaklis, P.J.: Supervision Based on Place Invariants: A Survey. In: Discrete Event Dynamic Systems, 16:451–492 (2006)

[7] Jantzen, M.: Language theory of Petri nets. In: Petri Nets: Central Models and Their Properties, Advances in Petri Nets, Lecture Notes in Computer Science 254-I, Brauer, W., Reisig, W., Rozenberg, G. (Editors), Springer Verlag, New York, pp: 397-412 (1987)

[8] Johnen, C., Frutos Escrig, D.: Decidability of home space property. In: LRI 503, Univ. d'Orsay (1989)

[9] Moody, J.O.. Antsaklis, P.J.: Supervisory Control of Discrete Event Systems Using Petri Nets. Kluwer (1998)

[10] Parigot, M., Pelz, E.: A logical formalism for the study of finite behaviour of Petri nets. In: Advances in Petri Nets, Lecture Notes in Computer Sciences 222, Springer Verlag, New York, pp: 346-361 (1985)

[11] Pelz, E.: Closure properties of deterministic Petri net languages. In: Proc. STACS 1987, Lecture Notes in Computer Sciences 247, Springer Verlag, New York, pp: 373-382 (1985)

[12] Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice-Hall, Englewood Cliffs, NJ (1981)

[13] Ramadge, P.J., Wonham, W.M.: The control of discrete event systems. In: Proceedings of the IEEE 77(1):81-98 (1989)

[14] Reutenauer, C.: The Mathematics of Petri Nets. Masson and Prentice-Hall (1990)

[15] Sreenivas, R.S.: On the existence of supervisory policies that enforce liveness in discrete-event dynamic systems modeled by controlled Petri nets. In: IEEE Transactions on Automatic Control 42(7):928-945 (1997)

[16] Sreenivas, R.S.: On minimal representations of Petri net languages. In: IEEE Transactions on Automatic Control 51(5):799- 804 (2006)

[17] Sreenivas, R.S.: On the Existence of Supervisory Policies That Enforce Liveness in Partially Controlled Free-Choice Petri Nets. In: IEEE Transactions on Automatic Control 57(2):435-449 (2012)

[18] Vidal-Naquet, G.: Deterministic Petri net languages. In: Application and Theory of Petri Net, Girault, C., Reisig, W. (Editors), Informatick-Fachberichte 52, Springer Verlag, New York (1982)