Testing experiments on synchronized Petri nets

Marco Pocci[(a)], Isabel Demongodin[(b)], Norbert Giambiasi[(c)], Alessandro Giua[(d)]

[(a)–(d)] LSIS, Aix-Marseille University, France

[(a),(d)] DIEE, University of Cagliari, Italy

{marco.pocci,isabel.demongodin,norbert.giambiasi}@lsis.org, giua@diee.unica.it

July 12, 2014

Abstract

Synchronizing sequences have been proposed in the late 60's to solve testing problems on systems modeled by finite state machines. Such sequences lead a system, seen as a black box, from an unknown current state to a known final one.

This paper presents a first investigation of the computation of synchronizing sequences for systems modeled by synchronized Petri nets. In the first part of the paper, existing techniques for automata are adapted to this new setting. Later on, new approaches, that exploit the net structure to efficiently compute synchronizing sequences without an exhaustive enumeration of the state space, are presented.

Note to practitioners

Driving a system to a known state when its current state it is not known is a very important problem in many practical applications, such as checking communication protocols, part orienteers, digital reset circuit, etc. This topic has received a lot of attention in the past few decades. The classic approach to solve this problem considers systems modeled by automata. In particular a standard technique requires the computation of a synchronizing sequence, i.e., a sequence of inputs that drives the system to a unique final state independently of the initial state and does not require the observation of the system's outputs.

This paper presents a first investigation on the computation of synchronizing sequences for systems modeled by synchronized Petri nets. Petri nets are a very intuitive model that is widely used in automation. Many analysis problems can be efficiently solved using Petri nets by taking into account the net structure, without an exhaustive enumeration of its state space. The techniques we propose for computing synchronizing sequences exploit the net structure and lead to viable algorithms that can be applied to large scale systems.

Published as:

M. Pocci, I. Demongodin, N. Giambiasi, A. Giua, "Testing experiments on synchronized Petri nets," *IEEE Trans. on Automation Science and Engineering*, Vol. 11, No. 1, pp. 125–138, January 2014. DOI: 10.1109/TASE.2013.2290774.

This work has been partially supported by Region Sardinia, LR 7/2007 (call 2010) under project SIAR (CRP-24709).

1 Introduction

Due to increasingly larger size and rising complexity, the need of checking systems' performance increases and *testing* problems periodically resurface. Several testing problems for *discrete event systems* (DES) have been introduced in the pioneering paper of Moore [1] using *automata*, while Lee and Yannakakis [2] have presented a review of different techniques to solve them. Five fundamental classes of testing problems have been defined: i) determining the final state after a test; ii) state identification; iii) state verification; iv) conformance testing; v) machine identification.

We consider the problem of determining the final state after a test. This has many important applications and is of general interest in several areas [3, 4], including robotics and robotic manipulation, industrial automation, etc.

A classical solution to this problem, assuming the system has no measurable outputs, consists in determining a *synchronizing sequence* (SS), i.e, an input sequence that drives the system to a known state no matter what current unknown state is. The main disadvantage of using automata in this context is the fact that the algorithms for computing SSs are polynomial in the number of states [2], and it is well known that in a discrete event system the state space grows exponentially with the number of subsystems that compose it (state space explosion) [5].

Looking for a more efficient way to solve such a problem, we turn to *Petri nets* (PNs), a powerful discrete event model. Petri nets provide several techniques to overcome the state space explosion problem, by means of *structural analysis*, i.e., algorithms that work on the net structure rather that its reachability set [5]. As far as we know, our results are the first concerning the use of Petri nets to compute SSs.

A review of the literature on SSs can be found in the next section, while in the section that follows the main contributions of the paper are described.

1.1 Literature review

Synchronization protocols have been developed to address global resource sharing in hierarchical real-time scheduling frameworks [6–8]. Synchronization experiments have been done also in biocomputing, where Benenson *et al.* [9, 10] have used DNA molecules as both software and hardware for finite automata of nano-scaling size. They have produced a solution of 3×10^{12} identical automata working in parallel. In order to synchronously bring each automaton to its "ready-to-restart" state, they have spiced it with a DNA molecule whose nucleotide sequence encodes a reset word. Jürgensen [11] has surveyed synchronization issues from the point of view of coding theory in real life communication systems. Synchronization is an important issue in network time protocol [12,13], where sharing of time information guaranties the correct internet system functioning. Most of real systems, natural or man-built, have no integrated reset or cannot be equipped with. That is the case of digital circuits, where a reset circuit not only involves human intervention but increases the cost of the device itself reducing its effectiveness.

In this field Cho *et al.* [14] have shown how to generate test cases for synchronous circuits with no reset. When classic procedures fail due to large circuit size or because a synchronizing sequence does not exist, Lu *et al.* [15] propose a technique based on partial reset, i.e., special inputs that reset a subset of the flip-flops in the circuit leaving the other flip-flops at their current values. Hierons [16] has presented a method to produce a test sequence with the minimum number of resets. Nowadays the synchronizing theory is a field of very intensive research, motivated also by the famous Černý conjecture [17]. In 1964 Ján Černý has conjectured that $(n-1)^2$ is the upper bound for the length of the shortest SS for any $n-$state machine. The conjecture is still open except for some special cases [18], [4]. Synchronization allows simple error recovery since, if an error is detected, a SS can be used to initialize the machine into a known state. That is why synchronization plays a key rôle in scientific contexts, without which all system behavior observations may become meaningless. An interesting challenge is represented by the *road coloring problem*, where one is asked whether there exists a coloring, i.e., an edge labeling, such that the resulting automaton can be synchronized. It was first stated by Adler in [19]. It has been investigated in various special cases and finally a positive solution has been presented by Trahtman in [20], for which complexity analysis are provided [21].

At present the problem of determining a synchronizing sequence has not yet been investigated for Petri net models and only few works have addressed the broad area of testing in the PN framework.

The question of automatically testing PNs has been investigated by Jourdan and Bochmann in [22]. They have adapted methods originally developed for *Finite State Machines* (FSMs) and, classifying the possible occurring types of error, identified some cases where *free choice* and 1-*safe* PNs [23] provide more significant results especially in concurrent systems. Later the authors have extended their results also to k-*safe* PNs [24]. Zhu and He have given an interesting classification of testing criteria [25] — without testing algorithms — and presented a theory of testing high-level Petri nets by adapting some of their general results in testing concurrent software systems.

In the PN modeling framework, one of the main supervisory control tasks is to guide the system from a given initial marking to a desired one similarly to the synchronization problem. Yamalidou *et al.* have presented a formulation based on linear optimization [26, 27]. Giua *et al.* have investigated the *state estimation problem*, proposing an algorithm to calculate an estimate — and a corresponding error bound — for the actual marking of a given PN based on the observation of a word. A different state estimation approach has been presented by Corona *et al.* [28], for labelled PNs with silent transitions, i.e., transitions that do not produce any observation. Similar techniques have been proposed by Lingxi *et al.* in [29] to get a minimum estimate of initial markings, aiming to characterize the minimum number of resources required at the initialization for a variety of systems.

1.2 Contribution of our work

This paper deals with the construction of SSs for systems described by bounded *synchronized PNs*. Synchronized PNs, as introduced by Moalla *et al.* [30], are nets where a label is associated with each transition. A label corresponds to an input event whose occurrence causes the firing of all enabled transitions associated with it.

We present several original contributions.

- First, we show that the classic automata approach [2], with minor changes, can be applied to the *reachability graph* (RG) of a net. While this approach is fairly general, working for arbitrary bounded nets, it does not offer any computational advantage.
- Looking for more efficient solutions, a special class of bounded Petri nets called *state machines* (SMs) [23] is considered. This model, albeit simple, allows a more compact description than automata: in fact if such a net has k tokens, its reachability graph may have up to $k^{m-1}/(m-1)!$ states, where m is the number of places in the net. We define k-SS a SS constructed for such a net under the assumption that it contains k tokens.

  For SMs we present several techniques for SS computation that are based on structural analysis and do not require the analysis of the whole RG. We address first the case of strongly connected nets with a single token and present a path based approach, called STS, to compute a 1–SS. We also show how one may construct a k-SS from an arbitrary 1-SS, computed either by STS or by RG analysis. This result offers significant computational advantages with respect to the automata case. The case of non strongly connected SM is also similarly addressed and several results concerning the existence of SS in such a case are given.
- We extended these results to possibly unbounded nets that are not state machines but that contain SM subnets: this family includes several classes of nets that have been used to model *resource allocation systems* [31, 32].
- In several experimental results — including randomly generated nets and a manufacturing example — we compare the performances of the different approaches discussed in the paper, in terms of computation times, sequence lengths and number of times the approach finds a solution. The results show the viability of the structural based approaches that in almost all cases result more efficient that the RG based approaches.

The paper is organized as follows. In Section 2 the background on automata with inputs and PNs is provided. Section 3 presents the classic SS construction method for automata with inputs. Section 4 shows how to obtain SSs by adapting the classic method developed for automata with inputs to bounded synchronized PNs, via RG construction. Section 5 proposes an original technique, based on path analysis, for efficiently determining SSs on strongly connected SMs. Section 6 presents a short discussion of algorithm complexity. The case of non-strongly connected SMs is investigated in Section 7. In Section 8 our approaches are extended to nets

containing state machine subnets. In Section 9 numerical results are presented, applying our tool to randomly generated SMs and to a manufacturing example. Finally, in Section 10, conclusions are drawn and open areas of research are outlined.

2 Background

2.1 Automata with inputs

An *automaton with inputs* Λ is a structure

$$\Lambda = (\chi, E, \delta),$$

where χ and E are finite and nonempty sets of states and input events respectively, and $\delta : \chi \times E \to \chi$ is the state transition function.

When the automaton is in the current state $x \in \chi$ and receives an event $e \in E$, it reaches the next state specified by $\delta(x, e)$.

Note that δ is usually assumed to be a *total function*, i.e., a function defined on each element (x, e) of its domain. In such a case the automaton is called *completely specified*.

The number of states and input events are respectively denoted by $n = |\chi|$, $p = |E|$. One can extend the transition function δ from input events to sequences of input events as follows: a) if ε denotes the empty input sequence, $\delta(x, \varepsilon) = x$ for all $x \in \chi$; b) for all $e \in E$ and for all $w \in E^*$ it holds that $\delta(x, we) = \delta(\delta(x, w), e)$[1].

The transition function δ can also be extended to a set of states as follows: for a set of states $\chi' \subseteq \chi$, an input event $e \in E$ yields the set of states $\chi'' = \delta(\chi', e) = \bigcup_{x \in \chi'} \delta(x, e)$.

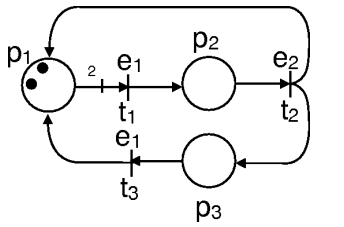
A simple way to represent any automaton is a graph, where states and input events are respectively depicted as nodes and labelled arcs.

An automaton with inputs is said *strongly connected* if there exists a directed path from any node of its graph to any other node.

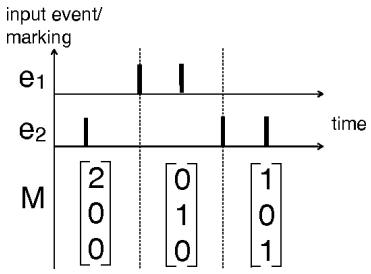
The set of nodes of a non-strongly connected automaton can be partitioned into its maximal strongly connected components. A component is called *ergodic*, if its set of output arcs is included in its set of input arcs, *transient*, otherwise.

An automaton contains at least one ergodic component and a strongly connected automaton consists of a single ergodic component.

[1]Here $*$ denotes the Kleene star operator and E^* represents the set of all sequences on alphabet E.



(a)



(b)

Figure 1: A synchronized PN (a) and a possible behavior (b).

2.2 Place/Transition nets

In this section, it is recalled the PN formalism used in the paper. For more details on PNs the reader is referred to [23, 33].

A Petri net (PN), or more properly a *Place/Transition net*, is a structure

$$N = (P, T, Pre, Post)\,,$$

where P is the set of m places, T is the set of q transitions, $Pre: P \times T \to \mathbb{N}$ and $Post: P \times T \to \mathbb{N}$ are the pre and post incidence functions that specify the weighted arcs.

A *marking* is a vector $M: P \to \mathbb{N}$ that assigns to each place a nonnegative integer number of tokens; the marking of a place p is denoted with $M(p)$. A marked PN is denoted $\langle N, M_0 \rangle$.

A transition t is enabled at M iff $M \geq Pre(\cdot, t)$. An enabled transition may be fired yielding the marking $M' = M + Post(\cdot, t) - Pre(\cdot, t)$. The set of enabled transitions at M is denoted $\mathcal{E}(M)$.

$M[\sigma\rangle$ denotes that the sequence of transitions $\sigma = t_1 \dots t_k$ is enabled at M and $M[\sigma\rangle M'$ denotes that the firing of σ from M yields M'.

A marking M is said to be *reachable* in $\langle N, M_0 \rangle$ iff there exists a firing sequence σ such that $M_0[\sigma\rangle M$. The set of all markings reachable from M_0 defines the *reachability set* of $\langle N, M_0 \rangle$ and is denoted with $R(N, M_0)$.

The *preset* and *postset* of a place p are respectively denoted ${}^\bullet p$ and $p^\bullet$. One can define the set of input transitions for a set of places $\hat{P}$ as the set ${}^\bullet\hat{P} = \{t : \forall p \in \hat{P},\ t \in {}^\bullet p\}$. Analogously the set of output transitions for a set of places $\hat{P}$ is the set $\hat{P}^\bullet = \{t : \forall p \in \hat{P},\ t \in p^\bullet\}$.

2.3 Synchronized Petri nets

A *synchronized PN* [33] is a structure $\langle N, E, f\rangle$ such that: i) N is a P/T net; ii) E is an alphabet of input events; iii) $f : T \to E$ is a labeling function that associates with each transition t an input event $f(t)$.

Given an initial marking M_0, a marked *synchronized PN* is a structure $\langle N, M_0, E, f\rangle$.

One extends the labeling function to sequences of transitions as follows: if $\sigma = t_1 t_2 \dots t_k$ then $f^*(\sigma) = f(t_1)f(t_2)\dots f(t_k)$.

The set T_e of transitions associated with input event e is defined as follows: $T_e = \{t \mid t \in T, f(t) = e\}$. Equivalently all transitions in T_e are said to be receptive to input event e.

The evolution of a synchronized PN is driven by input sequences as it follows. At marking M, transition $t \in T$ is fired iff:

1. it is enabled, i.e., $t \in \mathcal{E}(M)$;
2. the event $e = f(t)$ occurs.

On the contrary, the occurrence of an event associated with a transition $t \notin \mathcal{E}(M)$ does not produce any firing. Note that a single server semantic is here adopted, i.e., when input event e occurs, the enabled transitions in T_e fire only once regardless of their enabling degree.

One writes $M \xrightarrow{w} M'$ to denote that the application of input event sequence $w = e_1 \dots e_k$ from M drives the net to M'.

In Figure 1(a) is shown an example of synchronized PN. Note that labels next to each transition denote its name and the associated input event. In Figure 1(b) the net evolution is presented over a possible input sequence $w = e_2e_1e_1e_2e_2$ starting from marking M_0.

In the rest of the paper, the reader will only deal with the class of bounded synchronized PNs that also satisfy the following structural restriction, that is common in the literature to ensure the determinism of the model:

$$\nexists p \; s.t. \qquad t, t' \in p^\bullet \; and \; f(t) = f(t'). \tag{1}$$

When an event occurs in a deterministic net, all enabled transitions receptive to that event can simultaneously fire. Thus an input sequence $w = e_1e_2\cdots e_k \in E^*$ drives a deterministic net through the sequence of markings M_0, M_1, M_2, $\cdots$, M_k where M_0 is the initial marking and

$$M_{i+1} = M_i + \sum_{t \in T_{e_{i+1}} \cap \mathcal{E}(M_i)} \left(Post(\cdot, t) - Pre(\cdot, t)\right).$$

Example 1 *Consider the PN of Figure 1(a) and let* $M = [2\,0\,1]^T$ *be the current marking. Transitions* t_1 *and* t_3 *are enabled and upon the occurrence of event* e_1 *will simultaneously fire,*

yielding marking $M' = [1\,1\,0]^T$. *Note that markings* $[0\,1\,1]^T$ *and* $[3\,0\,0]^T$, *respectively obtained by the independent firing of* t_1 *and* t_3, *are never reachable.* ■

A marked PN $\langle N, M_0 \rangle$ is said to be bounded if there exists a positive constant k such that for all $M \in R(N, M_0)$, $M(p) \leq k \ \forall p \in P$. Such a net has a finite reachability set. In this case, the behavior of the net can be represented by the *reachability graph* (RG), a directed graph whose vertices correspond to reachable markings and whose edges correspond to the transitions and the associated event causing a change of marking.

The graph in Figure 3(a) (disregarding the dashed edges) is the reachability graph of the PN in Figure 1(a).

2.4 State machine Petri nets

Let first recall the definition of a state machine PN.

Definition 2 (State machine PN) [23] A *state machine* (SM) PN is an ordinary PN such that each transition t has exactly one input place and exactly one output place, i.e.,

$$|{}^\bullet t| = |t^\bullet| = 1 \qquad (\forall t \in T) \qquad \blacksquare$$

Observe that a SM $N = (P, T, Pre, Post)$ may also be represented by an *associated graph* $\mathcal{G}_N = (V, A)$ whose set of vertices $V = P$ coincides with set of places of the net, and whose set of arcs A corresponds to the set of transitions of the net, i.e., $A \subseteq P \times P = \{(p_i, p_j) \mid \exists t \in T, p_i = {}^\bullet t, p_j = t^\bullet\}$.

Such a graph can be partitioned into its maximal strongly connected components, analogously to the automata with inputs. These components induce also a partition of the set of places of the corresponding SM.

Definition 3 (Associated graph) Given a SM $N = (P, T, Pre, Post)$, let $\mathcal{G}_N = (P, A)$ be its associated graph. P can be partitioned into *components* as follows:

$$P = P_1 \cup \cdots \cup P_k$$

such that for all $i = 1, \ldots, k$ and $A_i = A \cap (P_i \times P_i)$ it holds that (P_i, A_i) is a maximal strongly connected sub-graph of $\mathcal{G}_N$. ■

As discussed in Section 2.1, components $P_1, \ldots P_k$ can be classified as transient or ergodic components.

Definition 4 (Condensed graph) Given a SM $N = (P, T, Pre, Post)$, its corresponding *condensed graph* $\mathcal{C}(N)$ is defined as a graph where each node represents a maximal strongly connected component and whose edges represent the transitions connecting these components. ■

In Figure 2(a) it is shown an example of a synchronized SM which is not strongly connected. Transient and ergodic components are respectively identified by dashed and dotted boxes. For

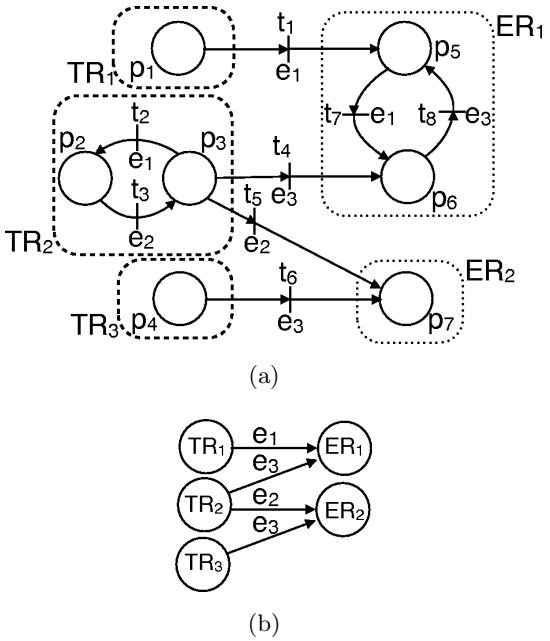


Figure 2: A not strongly connected PN (a) and its condensed graph (b).

such a net the transient components are $TR_1 = \{p_1\}$, $TR_2 = \{p_2,\ p_3\}$, $TR_3 = \{p_4\}$ and the ergodic components are $ER_1 = \{p_5,\ p_6\}$ and $ER_2 = \{p_7\}$. The corresponding $\mathcal{C}(N)$ is shown in Figure 2(b), where subnets induced by each component are represented by single nodes.

3 Synchronizing sequences for automata with inputs

In this section, the SS classic construction is presented by the aid of finite automata.

Definition 5 (SSs on automata) Consider an automaton with inputs $\Lambda = (\chi, E, \delta)$ and a state $\bar{x} \in \chi$. The input sequence $\bar{w}$ is called *synchronizing for state* $\bar{x}$ if it drives the automaton to $\bar{x}$, regardless of the initial state, i.e., $\forall x \in \chi$ it holds that $\delta(x, w) = \bar{x}$. ■

The information about the current state of Λ after applying an input sequence w is defined by the set $\phi(w) = \delta(\chi, w)$, called the *current state uncertainty of* w. In other words w is a synchronizing sequence (SS) that takes the automaton to the final state $\bar{x}$ iff $\phi(w) = \{\bar{x}\}$.

The synchronizing tree method [34, 35] has been proposed to provide shortest SSs. Such a method is suitable only for small size systems, since the memory required to build up the tree is high, and becomes useless when the size grows. As a matter of fact the problem of finding shortest SSs is known to be NP-complete [18].

Two polynomial algorithms have been mainly used to provide SSs that are not necessarily the shortest. The so-called *greedy* and *cycle* algorithms, respectively of Eppstein [18] and Trahtman [36], that have equivalent complexity.

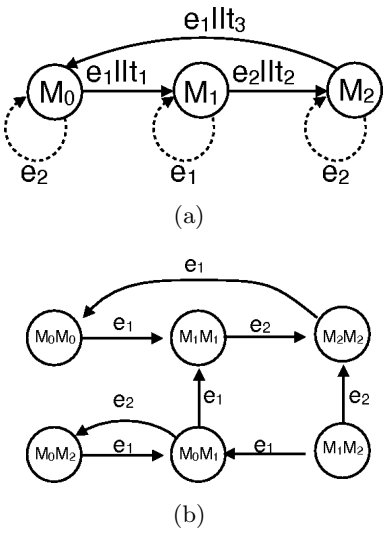


Figure 3: The completely specified RG (a) and the corresponding AG (b) of the PN in Figure 1(a).

The *greedy* algorithm [18] determines an input sequence that takes a given automaton, regardless of its initial state, to a known target state: note that the target state is determined by the algorithm and cannot be specified by the user. Here we propose a slightly different implementation of the greedy algorithm (see Algorithm 7), that takes as input also a state $\bar{x}$ and determines a sequence that synchronizes to that state.

This algorithm is later used as a building block to determine a SS to reach a given marking among those in the reachability set of a bounded PN.

Definition 6 (Auxiliary graph) Given an automaton with inputs Λ with n states, let $\mathcal{A}(\Lambda)$ be its *auxiliary graph* AG. $\mathcal{A}(\Lambda)$ contains $n(n+1)/2$ nodes, one for every unordered pair (x', x'') of states of Λ, including pairs (x, x) of identical states. There exists an edge from node (x', x'') to $(\hat{x}', \hat{x}'')$ labeled with an input event $e \in E$ iff $\delta(x', e) = \hat{x}'$ and $\delta(x'', e) = \hat{x}''$. ■

Algorithm 7 *(Greedy computation of SSs on automata with inputs)*
Input: An AG $\mathcal{A}(\Lambda)$, associated with an automaton with inputs $\Lambda = (\chi, E, \delta)$, and a target state $\bar{x} \in \chi$.
Ouput: A SS $\bar{w}$ for state $\bar{x}$.

1. Let $i = 0$.

2. Let $w_0 = \varepsilon$, the empty initial input sequence.

3. Let $\phi(w_0) = \chi$, the initial current state uncertainty.

4. While $\phi(w_i) \neq \{\bar{x}\}$, do

4.1. $i = i + 1$.

M_0	$[2\,0\,0]^T$
M_1	$[0\,1\,0]^T$
M_2	$[1\,0\,1]^T$

Table 1: Markings of the PN in Figure 1(a).

4.2. Pick two states $x, x' \in \phi(w_{i-1})$ such that $x \neq x'$.

4.3. **If** there does not exist any path in $\mathcal{A}(\Lambda)$ from node (x', x'') to $(\bar{x}, \bar{x})$, stop the computation, there exists no SS for $\bar{x}$.

Else find the shortest path from node (x', x'') to $(\bar{x}, \bar{x})$ and let w be the input sequence along this path, do

4.3.1. $w_i = w_{i-1}w$

4.3.2. $\phi(w_i) = \delta(\phi(w_{i-1}), w)$.

5. $\bar{w} = w_i$. ■

The following theorem provides a necessary and sufficient condition for the existence of a SS for a target final state.

Theorem 8 The following three propositions are equivalent.

1. Given an automaton with inputs $\Lambda = (\chi, E, \delta)$, there exists a SS for state $\bar{x} \in \chi$;
2. $\mathcal{A}(\Lambda)$ contains a path from every node (x', x''), where $x', x'' \in \chi$, to node $(\bar{x}, \bar{x})$;
3. Algorithm 7 determines a SS $\bar{w}$ for state $\bar{x} \in \chi$ at step 5., if there exists any SS.

Proof: [1) implies 2)] If there exists a SS for state $\bar{x} \in \chi$, there exists an input sequence w for $\bar{x}$ s.t. for any $x', x'' \in \chi$ it holds that $\delta(x', w) = \delta(x'', w) = \bar{x}$. Hence there exists a path labeled w from any (x', x'') to $(\bar{x}, \bar{x})$.

[2) implies 3)] Consider iteration i of the while loop of Algorithm 7. If there exists a path labeled w from any (x', x'') to $(\bar{x}, \bar{x})$, then it holds that $\delta(\{x', x''\}, w) = \{\bar{x}\}$. Hence the following inequality holds:

$$|\phi(w_i)| = |\delta(\phi(w_{i-1})\backslash\{x, x'\}, w) \cup \{\bar{x}\}| \leq |\phi(w_i)| - 1.$$

The existence of such a sequence for every couple of states $x', x'' \in \chi$ assures that the current state uncertainty will be reduced to singleton $\{\bar{x}\}$ after no more than n iteration.

[3) implies 1)] Since Algorithm 7 requires the current state uncertainty to be singleton and uses it as a stop criterium, if it terminates at step 5., then the sequence found is clearly a SS. □

One can easily understand that, when the automaton is not strongly connected, the above reachability condition will be verified only when there exists only one ergodic component and there may exist a SS only for those states belonging to this ergodic component.

4 Synchronizing sequences for bounded synchronized PNs

When computing a SS for real systems modeled by automata, it is assumed that a complete description of the model in terms of space-set, input events and transition function is given. The idea is that the test generator knows all possible states in which the system may be.

A similar notion can be given for Petri nets, where equivalently one can say that the test generator knows a "starting state", i.e., a possible state, of the system and the initial uncertainty coincides with the set of states reachable from this starting state.

In a synchronization problem via PNs, it is given a Petri net N and a *starting marking* M_0. The current marking M is unknown, but it is assumed to be reachable from M_0.

This starting marking, together with the firing rules, provides a characterization of the initial state uncertainty, given by $\mathcal{M}_0 = R(N, M_0)$. The goal is to find an input sequence that, regardless of the initial marking, drives the net to a known marking $\bar{M} \in R(N, M_0)$.

Given a synchronized PN $\langle N, E, f \rangle$, a straightforward approach to determine a SS consists in adapting the existing approach for automata to the reachability graph (RG).

It is easy to verify that this direct adaptation presents one shortcoming that makes it not always applicable: the greedy approach requires the graph to be completely specified, while in a RG of a PN this condition is not always true. In fact, from a marking not all transitions are necessarily enabled, causing the RG of the PN to be partially specified. In order to use the aforementioned approach it is necessary to turn its RG $\mathcal{G}$ into a completely specified $\tilde{\mathcal{G}}$.

Example 9 *Consider the PN in Figure 1(a). The current marking $M = [2\ 0\ 0]^T$ enables only transition t_1, then all events not associated with t_1 are not specified. Hence for that marking one adds a self loop labelled e_2 and so on for the rest of the reachable markings.* ■

In Figure 3(a) is shown the RG of the PN in Figure 1(a). Note that dashed edges are added in order to make it completely specified. In Figure 3(b) is shown the AG corresponding to the RG in Figure 3(a).

One can summarize the modified approach for PNs in the following algorithm.

Algorithm 10 *(RG computation of SSs on synchronized PNs)*
Input: A bounded synchronized PN $\langle N, E, f \rangle$, a starting marking M_0 and a target marking $\bar{M}$.
Ouput: A SS $\bar{w}$ for marking $\bar{M} \in R(N, M_0)$.

1. *Let $\mathcal{G}$ be the reachability graph of $\langle N, M_0 \rangle$.*
2. *Let $\tilde{\mathcal{G}}$ be the modified reachability graph obtained by completing $\mathcal{G}$, then by adding a self loop labelled e, i.e., $\forall M \in \mathcal{G}$ and $\forall e \in E$ s.t. $\nexists t \in T_e \cap \mathcal{E}(M)$.*
3. *Construct the corresponding AG $\mathcal{A}(\tilde{\mathcal{G}})$.*

4. *A SS for marking $\bar{M}$, if such a sequence exists, is given by the direct application of Algorithm 7 to $\mathcal{A}(\tilde{\mathcal{G}})$, having $\bar{M}$ as target.* ■

The following proposition can now be stated.

Proposition 11 Given a bounded synchronized PN N and a starting marking M_0, there exists a SS leading to a marking $\bar{M} \in R(N, M_0)$ iff the reachability condition on its AG $\mathcal{A}(\tilde{\mathcal{G}})$ is verified, i.e., there is a path from every node (M_i, M_j), with $M_i, M_j \in R(N, M_0)$, to node $(\bar{M}, \bar{M})$.

Proof: Consider a marked PN net $\langle N, M_0 \rangle$ and its RG $\mathcal{G}$. Given a marking $M \in R(N, M_0)$, a sequence $\sigma = t_{j1} t_{j2} \dots t_{jp}$ generates the trajectory $M[t_{j1}\rangle M_1[t_{j2} \dots t_{jp}\rangle M_p$ iff there exists an oriented path $\gamma = M t_{j1} M_1 t_{j2} \dots t_{jp} M_p$ in $\mathcal{G}$. The same equivalence holds between a synchronized PN and its completely specified RG $\tilde{\mathcal{G}}$. Thus an input sequence $w = e_{j1} e_{j2} \dots e_{jp}$ drives the net from M to M_p iff there exists an oriented path $\gamma = M e_{j1} M_1 e_{j2} \dots e_{jp} M_p$ in $\tilde{\mathcal{G}}$.

Since the completely specified RG $\tilde{\mathcal{G}}$ can be considered as an automaton whose behavior is equivalent to that of the synchronized PN, one can obtain a SS via Algorithm 7. □

5 Synchronizing sequences on strongly connected state machines

Consider a strongly connected SM defined in Section 2.4. Knowing the number of tokens k initially contained in the net — regardless of their initial distribution — is sufficient to exactly determine the reachability set of the net: in fact, the number of tokens will remain constant as the net evolves and any distribution of the k tokens can be reached.

If a SM is not strongly connected, knowing the number of tokens k initially contained in the net — but not their initial distribution — will give a larger approximation of the reachability set that may be used to design a SS. The knowledge of the number of tokens initially contained in each component — but not their initial distribution within each component — will provide an exact characterization of the reachability set.

This new setting aims to determine a SS without constructing the whole state-space. Hence a new formal definition of SS for SMs has to be given.

Definition 12 (SS on state machine PNs) Given a synchronized SM $\langle N, E, f \rangle$, assume that the initial marking M_0 is not given but is known to belong to a set

$$\mathcal{M}_0 = \{M \in \mathbb{N}^m \mid \sum_i M(p_i) = k\}.$$

$\bar{w}$ is called a k-SS if for all $M \in \mathcal{M}_0$ it holds $M \xrightarrow{\bar{w}} \bar{M}$. ■

In this section, we first analyze the problem of determining a 1-SS and then address the more general k-SS, starting from 1-SS.

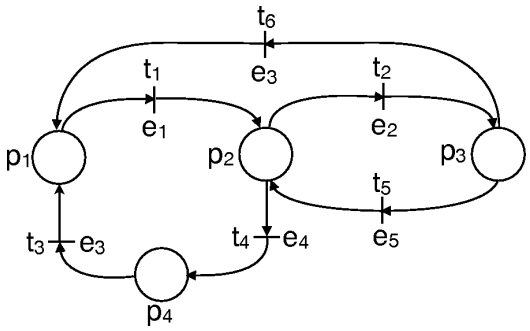


Figure 4: A strongly connected synchronized SM

5.1 1-SS on strongly connected state machines

In this subsection we present a particular technique to determine 1-SSs via sufficient conditions over the net structure. Such a technique can be more efficient than the approach presented in Algorithm 10, as discussed later in Section 6.

Let us first give the definition of directed path.

Definition 13 ***(Directed path)*** *Given a SM PN* $N = (P, T, Pre, Post)$*, an alternated sequence of places and transitions* $\rho = \langle p'_0 t'_1 p'_1 t'_2 \cdots t'_r p'_r \rangle$ *is called a* directed path *if* $\forall i = 0, 1, \ldots r$ *and* $\forall j = 1, \ldots r$ *it holds: i)* $p'_i \in P$ *and* $t'_j \in T$*; ii)* $p'_{j-1} \in {}^\bullet t'_j$ *and* $t'_j \in {}^\bullet p'_j$*. A path non-containing any repeated place is called* elementary*.* ■

The notion of *synchronizing transition sequence* for a set of places $\hat{P}$ and a specific place $\bar{p} \in \hat{P}$ can now be given.

Definition 14 *(Synchronizing transition sequence)* Given a synchronized SM $\langle N, E, f \rangle$, let $\rho(\hat{P}, \bar{p}) = \langle p'_0 t'_1 p'_1 t'_2 \cdots t'_r p'_r \rangle$, with $\bar{p} = p'_r$, be a directed path in $N = (P, T, Pre, Post)$ that visits all places in $\hat{P} \subseteq P$ and ends in $\bar{p} \in \hat{P}$, with $\bar{p} \neq p_j$ for $j = 0, 1, \ldots, r-1$. Let σ be the firing sequence obtained by removing all places from $\rho(\hat{P}, \bar{p})$. Such a sequence is called a *synchronizing transition sequence* for $\hat{P}$ and $\bar{p}$ if

C1) $\nexists t, t' \in \hat{P}^\bullet$ such that $t \in \sigma$, $t' \notin \sigma$, and $f(t) = f(t')$.

C2) $\forall p'_i, p'_k \in \rho(\hat{P}, \bar{p})$: if $p'_i = p'_k$ and $i < k$ it holds that $f(t'_j) \neq f(t'_k)$ for $j = 1, \ldots k-1$. ■

In simple words, condition C1) requires that there is no transitions exiting $\hat{P}$ and sharing the same label of a transition in σ. Condition C2) requires that if a place is visited multiple times, its ingoing transition does not share the same label with any of the transitions in the path.

A first result related to the existence of a SS for SMs with a single token can now be stated.

Proposition 15 Consider a strongly connected synchronized SM $\langle N, E, f \rangle$ containing a single token. Let σ be a synchronizing transition sequence for P and $\bar{p} \in P$. Then $\bar{w} = f^*(\sigma)$ is a 1-SS

for marking $\bar{M}$ that assigns the token to place $\bar{p}$, i.e.,

$$\bar{M}: \bar{M}(p) = \begin{cases} 1 \ if \ \ p = \bar{p}, \\ 0 \ otherwise. \end{cases}$$

Proof: Let $\sigma = t'_1 \cdots t'_r$ be the synchronizing transition sequence found and $\rho(P, \bar{p}) = \langle p'_0 t'_1 p'_1 t'_2 \cdots t'_r p'_r \rangle$, with $\bar{p} = p'_r$, be the corresponding path (not necessarily elementary).

Let $\bar{w}$ be the corresponding input event sequence, i.e., $\bar{w} = f^*(\sigma) = e'_0 \cdots e'_r$.

We first prove that after the occurrence of event e_1 the token can only be in a place p'_k such that $k \geq 1$. Assume, in fact, the token is initially in place p'_i and event e'_1 occurs. Two different cases have to be treated. If $i = 0$, then by definition of σ, the token is certainly driven to place p'_1. If $i \geq 1$, two further sub-cases are possible: a) no output transition of p'_i has label e_1, i.e., $T_{e_1} \cap p'^{\bullet}_i = \varnothing$, and the token will not not move; b) an output transition of p'_i has label e_1 and its firing moves the token to some place p'_j, with $j > i$.

The last result follows from conditions C1) and C2) of Definition 14. In fact, condition C1) assures that only transitions belonging to σ are receptive to e_1; thus the token can only be driven along the chosen path. Besides condition C2) assures that the token cannot go back in the upstream path along the sequence.

By repeating this argument, we can show that after the application of event e_i, for $i = 2, \ldots, r$, the token can only be in a place p'_k such that $k \geq i$, hence this ensures that when all events in the input sequence $\bar{w}$ have been applied the token will be in place $\bar{p}$. □

Next algorithm shows how Proposition 15 can be effectively used to compute a 1-SS. In this, function $\tau : P \times T \to T$ (resp. $\pi : P \times T \to P$) returns the set of transitions (resp. places) visited by directed path ρ. Function $start : P \times T \to P$ determines the last place has been added to ρ. For instance, consider the path $\rho = \langle p_r t_r p_{r-1} t_{r-1} \cdots t_1 p_0 \rangle$. It holds that $\tau(\rho) = \{t_1, t_2, \cdots, t_r\}$, $\pi(\rho) = \{p_0, p_1, \ldots, p_r\}$ and $start(\rho) = p_r$.

Algorithm 16 *(STS computation of 1-SSs on synchronized PNs)*
Input: A SM PN $N = (P, T, Pre, Post)$ and a target place $\bar{p}$.
Ouput: a 1-SS w that drives the token to place $\bar{p}$.

1. $\rho = \bar{p}$, $\mathcal{R} = \{\rho\}$;
2. $flag := false$;
3. ***while*** $flag = false \ \vee \ \mathcal{R} \neq \varnothing$
   a. *pick* $\rho \in \mathcal{R} : |\rho| = \max_{\rho' \in \mathcal{R}} |\rho'|$;
   b. $p := start(\rho)$;
   c. $\mathcal{T} := {}^{\bullet}p \backslash (\tau(\rho) \cup \bar{p}^{\bullet})$;
   d. ***while*** $flag = false \ \vee \ \mathcal{T} \neq \varnothing$,

i. *pick* $t \in \mathcal{T}$, $\rho' := {}^\bullet t t \rho$;

ii. ***if*** ρ' *does not satisfiy C2),* **then goto** *step* **3.d.v.**
end if

iii. ***if*** $\pi(\rho) = P$,
- ***if*** ρ' *satisfies C1),* ***then*** $flag := true$.
  else, **goto** *step* **3.d.v.**
  end if

end if

iv. $\mathcal{R} := \mathcal{R} \cup \{\rho'\}$

v. $\mathcal{T} := \mathcal{T} \backslash \{t\}$;

end while

e. $\mathcal{R} := \mathcal{R} \backslash \{\rho\}$.

end while

4. if $flag = false$,

a then *no STS exists;*

else

b *pick* $\rho \in \mathcal{R} : |\rho| = \max_{\rho' \in \mathcal{R}} |\rho'|$;

c *let* σ *be the firing sequence obtained removing all places from* ρ;

d $w := f^*(\sigma)$.

end if ■

The algorithm computes a synchronizing transition sequence. It starts from desired place $\bar{p}$ (step **1.**) and puts the path of zero length $\rho = \bar{p}$ into $\mathcal{R}$, which contains the set of path to be analyzed. The net is explored using a backward search until either a STS has been found, i.e., the $flag$ is true, or there are no more paths to analyze, i.e., $\mathcal{R} \neq \varnothing$ (step **3.**).

Once a path ρ is selected, we consider the set of transitions $\mathcal{T}$ inputting its start place p that i) have not already been visited in the path; ii) do not output from the final place $\bar{p}$ (step **3.c.**).

For all new paths ρ', obtained adding to ρ one transition in $\mathcal{T}$ and its input place (step **3.d.i.**), we do the following. First, we check condition C2) (step **3.d.ii**), which must hold for all prefixes of the final path. If it does not hold, we discard the path going to step **3.d.v.**. Then we check if ρ' contains all places: in this case, if it satisfies condition C1) (step **3.d.iii**) we stop the algorithm (flag=true), else we discard it, going to step **3.d.v.**. All new not discarded paths, are added to set $\mathcal{R}$ to be later explored (step **3.d.iv**).

When all transitions in $\mathcal{T}$ have been evaluated, path ρ is then removed from $\mathcal{R}$ (step **3.e.**).

In step **4.**, if the flag is set to false, there is no 1-SS constructible via the STS approach. Otherwise the path of maximum length is contained in $\mathcal{R}$ and it defines an STS.

Paths are constructed via a depth-first-search, as ensured by the condition of step **3.a.** that always picks (one of) the longest path(s). We could implement a *breadth-first-search* by picking — at the same step — the shortest $\rho \in \mathcal{R}$, to ensure for the shortest STS solution if found.

Example 17 *Consider the strongly connected SM in Figure 4. The objective is to find a* 1*-SS that leads the system to the marking* $[0\,0\,0\,1]^T$*. Let* $\rho = \langle p_1 t_1 p_2 t_2 p_3 t_5 p_2 t_4 p_4 \rangle$ *be the directed path that contains all the places, ending in* p_4*, and* $\sigma = t_1 t_2 t_5 t_4$ *the synchronizing transition sequence for* P *and place* p_4*. Sequence* $\bar{w} = f^*(\sigma) = e_1 e_2 e_5 e_4$ *is a* 1*-SS.* ■

Note that condition C1) of Definition 14 is sufficient to assure the sequence to be a synchronizing one if ρ is an elementary path.

The conditions given by Proposition 15 for the existence of a 1-SS are sufficient but not necessary.

Although one determines a 1-SS by just analyzing the net structure — avoiding then the RG and the AG construction and consistently reducing the complexity —, the conditions required are very restrictive.

In fact there are SMs for which those conditions do not hold but that still have a 1-SS.

Example 18 *Consider again the strongly connected SM in Figure 4 with one token and suppose* $f(t_5) = f(t_2) = e_2$*. This time one aims to find a* 1*-SS that leads the system to the marking* $[1\,0\,0\,0]^T$*. There clearly exists no synchronizing transition sequence with such a change of the labeling function, hence no* 1*-SS can be determined by Proposition 15. Despite this, one easily finds the* 1*-SS* $\bar{w} = e_4 e_3$ *by the way of Algorithm 10.* ■

Note that, when conditions required by Proposition 15 do not hold, one can always determine a SS using Algorithm 10, obviously with an increased complexity as shown later in Section 6.

5.2 k-SS on strongly connected state machines

We now consider the problem of determining a k-SS for nets with k tokens.

Proposition 19 Consider a strongly connected synchronized SM $\langle N, E, f \rangle$ containing k tokens.

Let σ be a synchronizing transition sequence for P and $\bar{p} \in P$ and $w = f^*(\sigma)$ a 1-SS. w^k is a k-SS that moves all k tokens to place $\bar{p}$, such that:

$$\bar{M} : \bar{M}(p) \;=\; \begin{cases} k \;\; if \;\; p = \bar{p}, \\ 0 \;\; otherwise. \end{cases}$$

Proof: Consider a first application of w, at least one token is driven to $\bar{p}$. Because of condition C2) and of the fact that the directed path does not pass through $\bar{p}$, none of the output transitions of this place is receptive to some event in w. Hence every application of w does not move the token from $\bar{p}$ and takes the k tokens at least one by one to place $\bar{p}$. □

Example 20 Consider the SM of Example 17, where $w = e_1e_2e_5e_4$ is the 1-SS previously found. Let the PN have 2 tokens. It holds that $w^2 = e_1e_2e_5e_4e_1e_2e_5e_4$ is a 2-SS, leading the net to the desired final marking $\bar{M} = [0\,0\,0\,2]^T$. ■

The previous propositions show that determining a synchronizing transition sequence allows to readily construct not only a 1-SS but a k-SS for an arbitrary k. However not all SSs can be obtained in this way.

Thus we consider the following problem: given any arbitrary 1-SS, constructed not from a synchronizing transition sequence but by using Algorithm 10, does Proposition 19 apply so that we can use it to construct a k-SS? Unfortunately this is not the case, as shown by the next example.

Example 21 Consider the SM of Example 17 and let $w = e_3e_1e_2e_5e_4$ be a 1-SS for p_4. Let the PN have 2 tokens. It is easy to see that w^2 is not a 2-SS, since only one token out of two would be driven to p_4. ■

It is however possible to provide a sufficient condition for an arbitrary 1-SS to ensure that, concatenating it k times, a k-SS is obtained.

Proposition 22 Consider a strongly connected synchronized SM $\langle N, E, f\rangle$ containing k tokens. Let w be a 1-SS for a target marking $\bar{M}$, such that $\bar{M}(p) = 1$ if $p = \bar{p}$, otherwise $\bar{M}(p) = 0$.

If for all $t \in \bar{p}^\bullet$ it holds that $f(t) \notin w$, i.e., sequence w does not contain any symbol labeling an output transition of place $\bar{p}$, then w^k is a k-SS for a target marking $\bar{M}_k$, such that $\bar{M}_k(p) = k$ if $p = \bar{p}$, otherwise $\bar{M}_k(p) = 0$.

Proof: After the first application of w, at least one of the tokens is driven to $\bar{p}$. Any further application of w moves to $\bar{p}$ at least one of the tokens not in this place, and does not move the tokens already in $\bar{p}$, as none of its output transitions is receptive to any event in w. Thus w^k takes the k tokens to place $\bar{p}$. □

6 A discussion on computational complexity

In this section we give an estimate of the computational complexity of both RG and STS approaches for 1-SS construction.

6.1 Complexity via reachability graph analysis

The *greedy* and the *cycle* algorithms work both in $O(n^3 + |E|n^2)$ time, where n and $|E|$ are, resp., the number of states and the input alphabet cardinality of the automaton. Proofs can be respectively found in [18] and [36].

Such algorithms are applicable to synchronized PNs by first exhaustively enumerating the state

space of the net, i.e., constructing its RG. Although alternative techniques are proposed to decrease its complexity (e.g. [37, 38]), the RG generation suffers from the problem of exponential space and time complexity. In particular, for a SM the reachability set of markings can significantly increase with the number of tokens under the following expression.

Theorem 23 Given a strongly connected SM $N = (P, T, Pre, Post)$, with k tokens, let m be the number of its places. The RG $\mathcal{G}$ of this net has a number of nodes equal to:

$$\binom{m+k-1}{m-1} \leq \frac{1}{(m-1)!} k^{m-1}$$

Proof: Consider the given net once a new node is added. It can be easily shown that the reachability set cardinality is given by the following formula:

$$|\mathcal{G}(m+1,k)| = \sum_{i=0}^{k} |\mathcal{G}(m,i)| =$$

$$= |\mathcal{G}(m,0)| + |\mathcal{G}(m,1)| + \ldots + |\mathcal{G}(m,k)|,$$

where $|\mathcal{G}(m,i)|$ is either the cardinality of the new obtained PN with $k-i$ tokens in the added place or that one of the initial PN with i tokens. Such results can be reported in a matrix form, obtaining the well known Pascal matrix, that comes out from the Pascal's triangle. The elements of the symmetric Pascal matrix are the binomial coefficients, i.e., it holds that

$$\binom{i+j-2}{i-1}$$

having $i = m$, $j = k+1$. □

Considering the above result, one can state the following lemma.

Lemma 24 Consider a strongly connected SM with k tokens. Let m be the number of its places and E be its input alphabet. For such a net, Algorithm 10 requires a time

$$O(|\mathcal{G}(m,k)|^3 + |E||\mathcal{G}(m,k)|^2)$$

$$\leq O\left(\left[\frac{k^{m-1}}{(m-1)!}\right]^3 + |E|\left[\frac{k^{m-1}}{(m-1)!}\right]^2\right). \qquad \blacksquare$$

6.2 Complexity via synchronizing transition sequences

We have shown in Proposition 15 a technique to compute a 1-SS on a strongly connected net based on synchronizing transitions sequences. Here we discuss the complexity of such a procedure.

To compute a synchronizing transition sequence one can proceed using a backward depth-first search from place $\bar{p}$ and verifying the conditions of Definition 14 over the labeling function.

It is known that a depth first search requires $O(b^d)$ time [39], for explicit graphs traversed with repetition, having a branching factor b and a depth search of d.

Assume that a SM has a backward branching factor (the number of transitions inputting in a place) bounded by $\phi = \max_{p \in P} |{}^\bullet p|$. While exploring the net with possible repetitions of places, an upper bound for the depth search length is $q - 1$, where q is the number of net transitions. Thus a first very rough approximation of the needed time is given by $O(\phi^{q-1})$.

This time only depends on structural net parameters, does not grow with the number of tokens and is typically smaller than the time required by Algorithm 10.

7 Synchronizing sequences on non-strongly connected state machines

Consider now connected — but not necessarily strongly connected — state machines. It can be shown how the existence of a SS depends on the interconnection between ergodic and transient components.

Proposition 25 Consider a synchronized SM $\langle N, E, f \rangle$ with μ transient components and η ergodic components. If $\eta > 1$ there exists no SS for such a net.

Proof: Let the net have two ergodic components ER' and ER''. Consider two initial markings M_0' and M_0'' both with k tokens such that M_0' (resp., M_0'') assigns all tokens to the component ER' (resp., ER''). Clearly there exists no marking $\bar{M}$ reachable from both M_0' and M_0'', hence no SS exists according to Definition 12. □

It is now proposed an algorithm to determine sequences for not strongly connected state machines having a single ergodic component where the interconnection between transient components can be arbitrary.

It is first stated the following result.

Proposition 26 Consider a SM $N = (P, T, Pre, Post)$ with a single component ER and let $\mathcal{C}(N)$ be its condensed graph. For each node v_i of $\mathcal{C}(N)$ associated with a transient component TR_i (with $i > 0$), let l_i be the length of the longest path from v_i to node v_0 associated with the ergodic component ER. Then if there is an edge (v_i, v_j) in $\mathcal{C}(N)$ it holds $l_i > l_j$.

Proof: First observe that $\mathcal{C}(N)$ is acyclic by construction and the node v_0 is reachable from any other node, hence $l_j \in \mathbb{N}$ is well defined for each node v_j (with $j > 0$). By definition, if (v_i, v_j) is an edge of $\mathcal{C}(N)$, then $l_i \geq l_j + 1$. □

The following algorithm for the one-token case allows to obtain a SS, such that a place $\bar{p}$ in the single ergodic component is marked.

Algorithm 27 ***(Computing a SS leading to*** $\bar{p} \in ER$***)***

Input: A synchronized PN $\langle N, E, f\rangle$ containing 1 tokens, with μ transient components and 1 ergodic component.
Ouput: A SS $\bar{w}$ for place $\bar{p}$.

1. Let $\mathcal{C}(N)$ be the condensed graph of N and associate ER with node v_0.
2. Label every other node v_i of $\mathcal{C}(N)$ with l_i, where l_i is the length of the longest path from v_i to v_0.
3. Let Σ_k be the set of nodes such that $\Sigma_k = \{v_i : l_i = k\}$, thus for construction $\Sigma_0 = \{v_0\}$.
4. Let $w = \varepsilon$.
5. For k=l_{max}:1
   - **5.1.** for all $v_i \in \Sigma_k$,
     - **5.1.1.** pick any transition t' connecting v_i to v_j, being $v_j \in \Sigma_{k'}$ and $k' < k$;
     - **5.1.2.** consider the strongly connected subnet associated with node v_i. Determine a 1-SS w' for place p', where $p' \in t'^{\bullet}$;
6. Consider the strongly connected subnet associated with node v_0. Let w be a 1-SS for place $\bar{p}$.
7. Let $w = wf^*(\sigma)$. ■

The algorithm starts taking into account the farthest nodes from ER. By definition of condensed graph, transient nodes with the same label value are not connected. Hence at step 5.1.2. the application of each couple (w', t'), step by step, drives the token always nearer to ER until it reaches it.

We have remarked that a net with more than one ergodic component cannot have a SS, at least according to Proposition 25. However, the knowledge of the initial token distribution among the net components may lead to other interesting characterizations, provided of course the initial state uncertainty is redefined according to this new information.

8 Synchronizing sequence on nets with state machine subnets

In the following we discuss how our approach can be applied in the more general setting of arbitrary nets, possibly unbounded, that contain SM subnets. This family includes several classes of nets used to model *resource allocation systems* (RAS) [32, 40] including S^3PR nets, S^4PR nets, S^*PR nets, $NS-RAP$, $ERCN$–merged nets or PR nets. A survey of the field can be found in [32].

Proposition 28 *Consider a synchronized PN $\langle N, E, f\rangle$. Let $P = P_s \cup P_z$ and $T = T_s \cup T_z$, such that $N_s = (P_s, T_s, Pre_s, Post_s)$ is a strongly connected SM subnet, where Pre_s and $Post_s$ are the restrictions of Pre and $Post$ to $P_s \times T_s$.*

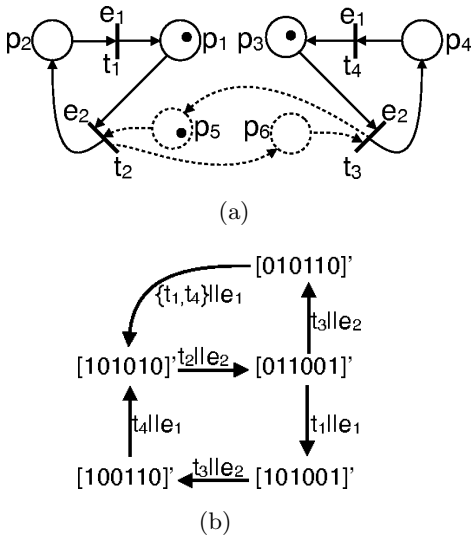


Figure 5: A synchronized PN (a) and its RG

Given an initial marking M_0 of N, let $\bar{w}$ be a SS that drives the subnet N_s to a target marking $\bar{M}_s$. This sequence $\bar{w}$ drives N to a target marking $\bar{M}$ such that:

$$\bar{M}(p) = \bar{M}_s(p) \;\; if \;\; p \in P_s,$$

if the two following conditions hold:

i) $\{T_z^\bullet \cup^\bullet T_z\} \cap P_s = \varnothing$;

ii) $(\forall e \in \bar{w})\ T_e \cap P_z^\bullet \cap T_s = \varnothing$.

Proof: Condition i) states that no transition $t \in T_z$ is connected to any place $p \in P_s$. This ensures that the firing of a transition in T_z cannot affect the marking of places in P_s. Hence, given the special structure of N_s, the following condition holds for any initial marking M_0: $(\forall M \in R(N, M_0))\ \sum_{p \in P_s} M(p) = \sum_{p \in P_s} M_0(p)$, i.e., the token count in the SM component remains constant, even if in the whole net N does not.

Let $\bar{w} = e_1 e_2 \cdots e_k$ be a SS for subnet N_s that yields a known marking $\bar{M}_s$ from any reachable marking M of the subnet. To prove the result, it is sufficient to show that at each step $i = 1, \ldots, k$ the same sequence, applied to N from any marking M', with $M'(p) = M(p)$ if $p \in P_s$, produces exactly the same transition firings that it produces in N_s.

In fact, when a input symbol $e_i \in \bar{w}$ is applied:

- *by condition i), all transitions that can fire in N_s can also fire in N, because the additional places P_z in N cannot disable these transitions since they do not belong to ${}^\bullet T_s$;*
- *by condition ii), no transition in $P_z^\bullet$ can fire, because no transition in $P_z^\bullet$ has label e_i.* □

(a) $|N_{STS}|/|N_{RG}|$

m \ q	3	4	5	6	7	8	9	10	11	12	13	14	15
2	1	1	1	1	1	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1	1	1	1	1	1
4		1	1	1	1	1	1	1	1	1	1	1	1
5			1	0,8	1	1	1	1	1	1	1	0,8	1
6				1	1	1	1	0,8	1	1	1	1	1
7					1	1	0,6	0,66	0,8	1	0,88	1	1

(b) $|\hat{T}_{STS}|/|\hat{T}_{RG}|$

m \ q	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0.22	0.14	0.24	0.17	0.20	0.15	0.13	0.18	0.15	0.18	0.12	0.12	0.10
3	0.23	0.23	0.21	0.26	0.23	0.26	0.36	0.48	0.54	0.30	0.33	0.40	0.74
4		0.43	0.20	0.32	0.37	0.50	0.62	0.54	0.52	0.65	0.61	0.75	0.93
5			0.6	0.30	0.32	0.41	0.53	0.60	0.90	0.59	0.78	0.72	1.52
6				0.45	0.21	0.29	0.45	0.72	0.73	0.54	1.03	0.76	1.08
7					0.62	0.16	0.22	0.27	0.88	0.78	1,15	1,6	2,54

(c) $|\hat{L}_{STS}|/|\hat{L}_{RG}|$

m \ q	3	4	5	6	7	8	9	10	11	12	13	14	15
2	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
3	0.83	0.91	0.91	0.91	0.83	1.00	1.10	0.83	1.00	0.91	0.77	1.10	1.00
4		0.65	0.85	0.89	1.00	0.77	0.85	0.95	0.84	1.00	0.94	0.89	1.06
5			0.8	0.85	0.85	0.70	0.91	0.91	0.96	0.75	1.00	0.87	0.92
6				0.6	0.53	0.90	0.79	0.90	0.80	0.74	0.86	0.81	0.70
7					0.54	0.61	0.57	0.50	0.88	0.59	0,76	0,85	0,69

Table 2: Numerical results for randomly generated SM PNs ($k = 1$)

(a) $|\hat{T}_{STS}|/|\hat{T}_{RG}|$

m \ q	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0.26	0.12	0.23	0.16	0.17	0.13	0.11	0.16	0.14	0.15	0.11	0.11	0.10
3	0.27	0.16	0.14	0.14	0.14	0.21	0.28	0.35	0.42	0.24	0.27	0.24	0.54
4		0.28	0.12	0.20	0.23	0.31	0.41	0.35	0.11	0.39	0.20	0.36	0.40
5			0.17	0.08	0.08	0.19	0.01	0.28	0.44	0.27	0.35	10^{-3}	0.75
6				10^{-3}	10^{-3}	0.09	0.02	10^{-3}	10^{-3}	0.20	10^{-3}	0.29	0.39
7					10^{-3}	10^{-3}	10^{-3}	0.05	0.05	10^{-3}	10^{-3}	10^{-4}	10^{-3}

(b) $|\hat{L}_{STS}|/|\hat{L}_{RG}|$

m \ q	3	4	5	6	7	8	9	10	11	12	13	14	15
2	1.71	1.71	1.71	1.71	1.71	1.71	1.71	1.71	1.71	1.71	1.71	1.71	1.71
3	1.5	1.60	1.41	1.60	1.33	1.71	1.89	1.50	1.71	1.60	1.41	1.89	1.76
4		1.5	1.41	1.42	1.64	1.46	1.57	1.60	1.54	1.54	1.50	1.54	1.77
5			1.1	1.42	1.35	1.26	1.44	1.50	1.55	1.36	1.65	1.37	1.65
6				1.3	0.92	1.52	1.32	1.64	1.27	1.33	1.28	1.39	1.30
7					0.85	0.97	0.97	0.92	0.88	0.77	0,79	0,64	0,79

Table 3: Numerical results for randomly generated SM PNs ($k = 2$)

Such a result can be further generalized to nets containing more than one state machine subnets.

Proposition 29 *Consider a synchronized PN* $\langle N, E, f\rangle$. *Let* $P_s \cup P_z = P$ *and* $T_s \cup T_z = T$, *where* $P_s = \dot{\bigcup_{i=1}^{n}} P_{s,i}$ *and* $T_s = \dot{\bigcup_{i=1}^{n}} T_{s,i}$ *(here* $\dot{\cup}$ *denotes the union of disjoint subsets). These sets are such that for* $i = 1, 2, \ldots n$ $N_{s,i} = (P_{s,i}, T_{s,i}, Pre_{s,i}, Post_{s,i})$ *is a strongly connected SM subnet.* $Pre_{s,i}$ *and* $Post_{s,i}$ *are the restrictions to* Pre *and* $Post$ *to* $P_{s,i} \times T_{s,i}$.

For every subnet $N_{s,i}$, *let* $\bar{w}_i$ *be a SS that drives the subnet* $N_{s,i}$ *to a target marking* $\bar{M}_{s,i}$. *The sequence* $\bar{w} = \bar{w}_1 \bar{w}_2 \ldots \bar{w}_n$ *drives* N *to a target marking* $\bar{M}$ *such that:*

$$\bar{M}(p) = \bar{M}'_{s,i} \text{ with } \bar{M}_{s,i} \xrightarrow{\bar{w}_{i+1}\bar{w}_{i+2}\cdots\bar{w}_n} \bar{M}'_{s,i} \quad if \;\; p \in P_{s,i},$$

if the two following conditions hold:

i) $\{{}^\bullet T_z \cup T_z^\bullet\} \cap P_s = \varnothing$;

ii) $(\forall e \in \bar{w}_i)\ T_e \cap P_z^\bullet \bigcap_{j=1}^{i} T_{s,j} = \varnothing$.

Proof: The proof follows along the same lines of the proof of Proposition 28 with just an additional consideration. First we observe that condition ii) in Proposition 29 is a generalization of condition ii) in Proposition 28: the condition now must hold not only for the transitions of net i *but also for those of all nets* j *with* $j < i$. *In fact, the overall SS is composed by concatenation of the SSs for each state machine subnet. When we apply the SS* w_i *to the net, we assume that the markings of all subnets* j, *for* $j < i$ *are known but may change, as some transitions in the already synchronized subnets may be receptive to an event* $e \in \bar{w}_i$. *However, condition ii) ensures that the enabling condition of these transitions does not depend on the places in* P_z, *whose marking is unknown, and the marking reached after the application of event* e *is computable.* □

Sequence $\bar{w}$ determined in the previous proposition is a SS for the subnet N_s. It also drives the complete model N to a state where the marking of places in P_s is known, while in general nothing can be said about the marking of places in P_z.

Example 30 *Consider the net in Figure 5(a). Let* $P_{s,1} = \{p_1, p_2\}$, $P_{s,2} = \{p_3, p_4\}$, $P_z = \{p_5, p_6\}$, $T_{s,1} = \{t_1, t_2\}$, $T_{s,2} = \{t_3, t_4\}$ *and then* $T_z = \varnothing$. N_s *is then the net depicted in Figure 5(a), without taking into account dashed places and arcs. Let* $\bar{w}_1$ *and* $\bar{w}_2$ *be SSs that drives respectively* $N_{s,1} = (P_{s,1}, T_{s,1}, Pre_{s,1}, Post_{s,1})$ *to* $\bar{M}_{s,1} = [0\,1]^T$ *and* $N_{s,2} = (P_{s,2}, T_{s,2}, Pre_{s,2}, Post_{s,2})$ *to* $\bar{M}_{s,2} = [0\,1]^T$. *By separately analyzing the two subnets,* $\bar{w}_1 = \bar{w}_2 = e_1$ *are obtained.* $\bar{w} = \bar{w}_1 \bar{w}_2$ *respects conditions i) and ii) of Proposition 29 and is therefore a SS for* N_s, *i.e., it drives the net to a marking* $\bar{M}$ *that is either* $[1\,0\,1\,0\,1\,0]^T$ *or* $[1\,0\,1\,0\,0\,1]^T$, *as can be seen by its RG in Figure 5(b).* □

9 Experimental results

This section has two objectives. First, we compare the two algorithms we have presented for SS computation of state machine Petri nets (reachability based versus path based) by applying them to a series of randomly generated nets and analyzing their performance. Second, we show the effectiveness of our approach by applying it to a manufacturing system modeled by a PN which is not a SM. The model data and MATLAB programs can be downloaded from [41].

All simulations have been run on a mini Mac intel core Duo 2, 2.53 GHz processor, with 4 GB 1067 MhZ DDR3 RAM.

9.1 Numerical results for randomly generated PNs

Randomly generated models have been previously adopted as a validation method for synchronizing sequence construction also by Roman [42].

For selected values of m places, q transitions and $k = 1, 2$ tokens, we randomly generate 100 deterministic and strongly connected synchronized SMs having $m = 2 \div 7$ places, $q = m \div 15$ transitions and $k = 1, 2$ tokens. In all cases the input alphabet has cardinality f randomly chosen in $\frac{q}{m} \div q$. Note that $\frac{q}{m}$ is the minimal alphabet cardinality to ensure the determinism for a SM having m places and q transitions. For each net a place is randomly selected and we use both Algorithm 10 (denoted RG) and Algorithm 10 (denoted by STS) to determine a SS to this place. The algorithms are compared by means of three performance indexes:

N_{RG}, N_{STS}: number of times the algorithm successfully terminates returning a SS;

$\hat{T}_{RG}$, $\hat{T}_{STS}$: average time required to compute the sequence;

$\hat{L}_{RG}$, $\hat{L}_{STS}$: average length of the sequences.

Finally the performance of the two approaches is evaluated by computing the ratio of N_{STS} (resp. $\hat{T}_{STS}$ and $\hat{L}_{STS}$) to N_{RG} (resp. $\hat{T}_{RG}$ and $\hat{L}_{RG}$).

Results are shown in Table 2 for nets with one token and in Table 3 for the two token case. Note that the table showing N_{STS}/N_{RG} does not depend on the number of tokens and thus it is shown only for $k = 1$. Black cells denote parameter values for which no strongly connected SM can be generated, i.e., for $m > q$.

Table I(a) shows the ratio N_{STS}/N_{RG} between the number of times a SS has been found using the STS and the RG approach. In the previous sections we have mentioned that while the RG approach always determines a SS if any exists, the STS approach may fail to do so. Hence the value in the table should be contained in the interval $[0, 1]$. We can observe, however, that over 88% of the table entries show a value of 1, hence confirming that the STS approach can find a solution in most cases and thus this result is not too restrictive.

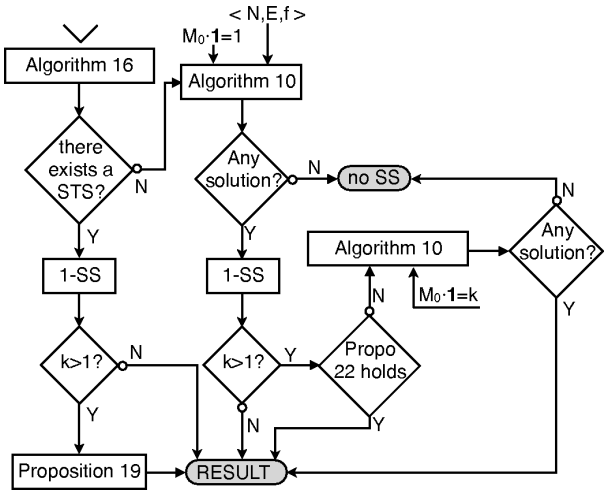


Figure 6: Suggested way to determine a SS for strongly connected SMs.

Table I(b) shows the ratio T_{STS}/T_{RG} between the execution time to compute a SS using the STS and the RG approach for nets with one token. Here we expect the STS approach to be more efficient, as discussed in Section 6, and this is confirmed from the fact that in almost all cases the table entries are smaller than one. Only in a few cases, for very large values of m and q, we have that the RG method is faster than the STS one. This, we believe, it is due to our implementation of the STS approach that uses a brute force depth-first search.

Table II(a) shows the ratio T_{STS}/T_{RG} between the execution time to compute a SS using the STS and the RG approach for nets with two tokens. Here we see the main advantage of the STS method, that can use a 1-SS to determine a k-SS, while the RG method had a complexity that grows polynomially with k (and exponentially with m), as discussed in Section 6. Here the advantage of the STS method is more noticeable for large values of m and q. Table I(c) and Table II(b) show the ratio L_{STS}/L_{RG} between the average length of a SS computed using the STS and the RG approach for nets with one or two tokens. This index is probably less significant than the previous ones, although one may argue that the shortest the SS the less expensive is the synchronization (in terms of costs or of time required). Here we can see that in the case of one token the STS approach in most of the cases produces shorter SS, while the situation is the opposite for 2 tokens. This is due to the fact that the k-SS obtained by STS is always k times longer than the corresponding 1-SS, while shorter solutions may be obtained by the RG approach.

On the base of these results, we can say that to compute a SS for strongly connected Petri nets it is convenient to first search for a STS based solution using Algorithm 16 and then, if this fails, to use Algorithm 10. This is summarized in the flowchart in Figure 6.

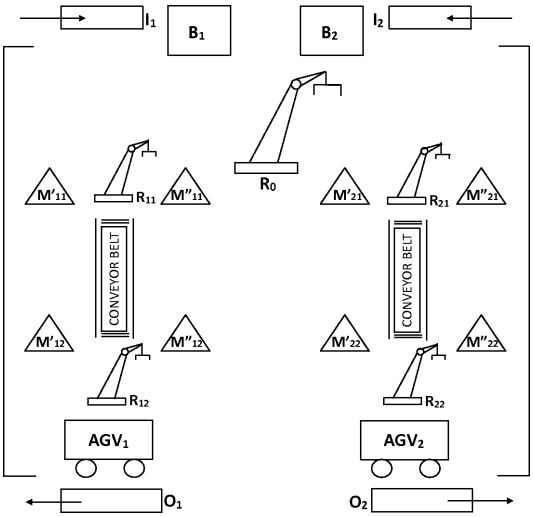


Figure 7: Layout of the manufacturing system.

9.2 A manufacturing example

We consider a manufacturing plant consisting of two production lines, that produce two different kinds of final product. The two lines are symmetric and run in parallel. Each line i has a fixed number of k pallets, that limit the number of parts that can be under processing at a given time. A raw piece entering the system waits in the buffer (not modeled) until a pallet becomes available. Robot R_0 feeds the two lines alternatively, taking one part from the buffer and mounting it on an empty pallet.

In each line there are two workstations, which are composed by two machines and one robot. A workstation can process several pallets at a time.

For instance consider line 1. The pallet entering the system is deposed by R_0 on a conveyor belt and is moved to the first workstation where machines $M'_{1,1}$ and $M"_{1,1}$ perform their operation as requested. Robot R_{11} moves the pallet from the conveyor belt to the machines and viz.

Once the operations required on the first workstation have been completed, the pallet is put again onto the conveyor belt and moved to the second workstation where the processing is repeated. After processing, parts are unloaded from the pallets by robot $R_{1,2}$ and put on an AGV that moves them to the output buffer O_1.

The system's layout is shown in Figure 7: we say that such a plant has production length $l = 2$ because each line is composed by a series of two workstations. We can consider a parameterized family of plants of this type, assuming that the number of pallets k and length l of the production lines may vary. For this family of plants, we obtain the synchronized Petri net depicted in Figure 8.

This Petri net has $4+6l$ places and $2+10l$ transitions. The marking of places p_1 and p_2 represents number of the empty pallets in each line, the marking of place p_3 (resp. p_4) denotes that robot

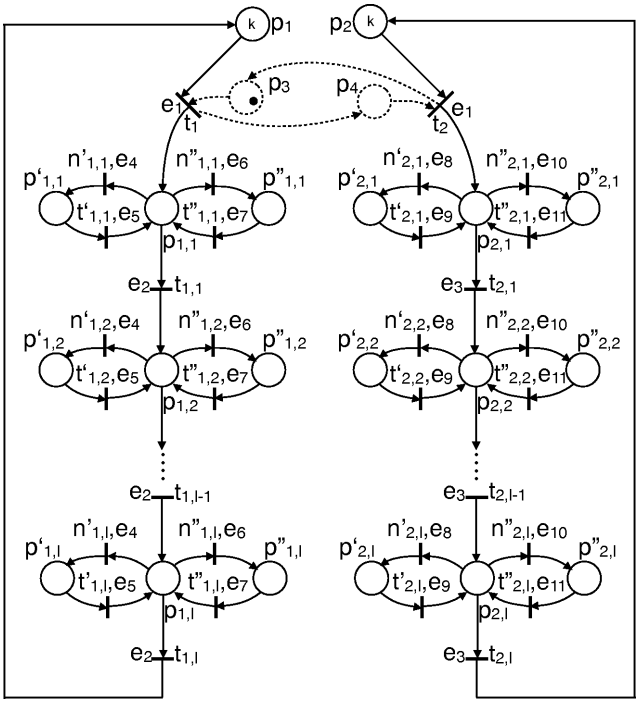


Figure 8: PN model of the actuactors of the manufacturing system in Figure 7

R_0 is ready to move a pallet to the left (resp. right) production line.

Consider machines M'_{ij} and $M"_{ij}$, which compose workstation j in line i. Transition $n'_{i,j}$ (resp. $n"_{i,j}$) represents the loading of a pallet from the conveyor belt queue to machine M'_{ij} (resp. $M"_{ij}$) while transition $t'_{i,j}$ (resp., $t"_{i,j}$) represents the unloading. Tokens in place p'_{ij} (resp. $p"_{ij}$) denote pallets loaded on machine M'_{ij} (resp. $M"_{ij}$).

The firing of transition $t_{i,j}$ denotes the transfer of a pallet to the next workstation.

Although the considered PN is not a state machine, the proposed approach is still applicable.

The PN in Figure 8, without taking into account dashed places and arcs, is composed by two SMs. Hence, according to Proposition 29, $P_s = P_{s,1} \cup P_{s,2} = P \setminus \{p_3, p_4\}$, $P_{s,1} = \{p_1, p_{1,1}, p'_{1,1}, p''_{1,1}, \dots p_{1,l}, p'_{1,l}, p''_{1,l}\}$, $P_{s,2} = \{p_2, p_{2,1}, p'_{2,1}, p''_{2,1}, \dots p_{2,l}, p'_{2,l}, p''_{2,l}\}$ and $T_s = T$.

We look for a SS that from an arbitrary state can empty the system, thus moving all empty pallets to the input buffers. The obtained sequences are $\bar{w}_1 = \{e_4 e_5 e_6 e_7 e_2\}^l$, $\bar{w}_2 = \{e_8 e_9 e_{10} e_{11} e_3\}^l$ and $\bar{w} = \bar{w}_1 \bar{w}_2$. $\bar{w}$ meets conditions of Proposition 29 then is a SS for N_s.

Results obtained with the STS and RG approaches are shown in Table 4, where length (L_{STS} and L_{RG}) and required time (T_{STS} and T_{RG}) of the SS are summarized for different values of m and l. Also, the table shows the cardinality of the reachability graph ($|G|$) and of the AG ($|\mathcal{A}(G)|$). These are important parameters to understand the limits of the RG approach, while exhaustively enumerating the set space of the net.

k	l	$\|G\|$	$\|\mathcal{A}(G)\|$	$T_{RG}[s]$	$T_{STS}[s]$	L_{RG}	L_{STS}
1	1	32	528	3.43	0.26	16	10
1	2	98	4851	*o.t.*	0.43	*n.c.*	20
1	3	200	20100	*o.t.*	2.81	*n.c.*	30
1	4	338	57291	*o.t.*	31.13	*n.c.*	40
2	1	200	20100	*o.t.*	0.26	*n.c.*	20
2	2	1568	*o.t.*	*n.c.*	0.43	*n.c.*	40
2	3	*o.t.*	*n.c.*	*n.c.*	4.27	*n.c.*	60
2	4	*o.t.*	*n.c.*	*n.c.*	56.75	*n.c.*	80
3	1	800	320400	*o.t.*	0.26	*n.c.*	30
3	2	*o.t.*	*n.c.*	*n.c.*	0.43	*n.c.*	60
3	3	*o.t.*	*n.c.*	*n.c.*	4.27	*n.c.*	90
3	4	*o.t.*	*n.c.*	*n.c.*	56.75	*n.c.*	120

Table 4: Numerical results for a manufacturing system.

The table shows also non-numerical values where the corresponding result cannot be provided: i) *out of time* (o.t.), when the corresponding value has not been computed within 6 hours; ii) *not computable* (n.c.), if the corresponding value cannot be computed: e.g., the RG is o.t. and the corresponding AG cannot be evaluated.

Note that for an increasing number of tokens the RG approach goes almost always o.t., due to a significant larger space state. On the contrary, the required time does not change with the STS approach. Also, the length of the constructed SS grows linearly with k, confirming the results of Proposition 19.

10 Conclusions and future works

In this paper, we have shown how automata techniques can be applied with minor changes to the class of bounded synchronized PNs.

Also it has been proposed a method that provides a synchronizing sequence for the class of synchronized SM PNs.

Our approach alleviates the state explosion problem also in the case of multiple tokens, since the construction of the reachability graph is not needed. We have shown by means of several examples how the computational time does not increase as the number of tokens in the net increases

There is an open line for interesting future works. We plan to extend our approach to unbounded PNs, whose behavior can be approximated with a finite *coverability graph* (CG), by introducing an ω component to denote a place whose token content may be arbitrarily large.

Note that classic coverability methods construction cannot be directly applied to this class of PNs, that is why a new algorithmic procedure for the CG construction has to be provided.

References

[1] E. F. Moore, "Gedanken-experiments on sequential machines," *Automata Studies, Annals of Mathematical Studies*, vol. 34, pp. 129 – 153, 1956.

[2] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machine – a survey," *Proceedings of the IEEE*, vol. 84, pp. 1090–1123, August 1996.

[3] B. K. Natarajan, "Some paradigms for the automated design of parts feeders," *The International Journal of Robotics Research*, vol. 8 (6), p. 89–109, 1989.

[4] D. Ananichev and M. V. Volkov, "Synchronizing monotonic automata," *Lecture Notes in Computer Science*, pp. 111–121, 2003.

[5] L. Holloway, B. Krogh, and A. Giua, "A survey of Petri net methods for controlled discrete event systems," *Discrete Event Dynamic Systems*, vol. 7, no. 2, pp. 151–190, 1997.

[6] M. Van den Heuvel, R. Bril, and J. Lukkien, "Transparent synchronization protocols for compositional real-time systems," *Industrial Informatics, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2011.

[7] M. Behnam, I. Shin, T. Nolte, and M. Nolin, "Sirap: A synchronization protocol for hierarchical resource sharing in real-time open systems," in *Proceedings of the 7th ACM & IEEE International Conference on Embedded Software (EMSOFT'07)*, pp. 279–288, ACM, October 2007.

[8] T. Nolte, I. Shin, M. Behnam, and M. Sjodin, "A synchronization protocol for temporal isolation of software components in vehicular systems," *Industrial Informatics, IEEE Transactions on*, vol. 5, pp. 375 –387, nov. 2009.

[9] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro, "Programmable and autonomous computing machine made of biomolecules," *Nature*, vol. 414, pp. 430 – 434, 2001.

[10] Y. Benenson, R. Adar, T. Paz-Elizur, Z. Livneh, and E. Shapiro, "Dna molecule provides a computing machine with both data and fuel," *Proc. National Acad. Sci. USA*, vol. 100, no. 5, pp. 2191–2196, 2003.

[11] H. Jürgensen, "Synchronization," *Information and Computation*, vol. 206, pp. 1033–1044, September 2008.

[12] G. Gaderer, P. Loschmidt, and T. Sauter, "Improving fault tolerance in high-precision clock synchronization," *Industrial Informatics, IEEE Transactions on*, vol. 6, pp. 206 –215, may 2010.

[13] D. Mills, "Internet time synchronization: the network time protocol," *Communications, IEEE Transactions on*, vol. 39, pp. 1482 –1493, oct 1991.

[14] H. Cho, S.-W. Jeong, F. Somenzi, and C. Pixley, "Multiple observation time single reference test generation using synchronizing sequences," in *Design Automation, 1993, with the European Event in ASIC Design. Proceedings. [4th] European Conference on*, pp. 494 –498, feb 1993.

[15] Y. Lu and I. Pomeranz, "Synchronization of large sequential circuits by partial reset," in *VLSI Test Symposium, 1996., Proceedings of 14th*, pp. 93 –98, apr-1 may 1996.

[16] R. M. Hierons, "Using a minimal number of resets when testing from a finite state machine," *Inf. Process. Lett.*, vol. 90, pp. 287–292, June 2004.

[17] J. Černý, "Poznámka k homogénnym experimentom s konečnými automatmi. (slovak) [a note on homogeneous experiments with finite automata]," *Mathematica Slovaca*, vol. 3, pp. 208–216, 1964.

[18] D. Eppstein, "Reset sequences for monotonic automata," *SIAM J. Computing*, vol. 19, pp. 500–510, 1990.

[19] R. Adler, L. Goodwyn, and B. Weiss, "Equivalence of topological markov shifts," *Israel Journal of Mathematics*, vol. 27, pp. 49–63, 1977.

[20] A. N. Trahtman, "The road coloring problem," *Israel Journal of Mathematics*, vol. 172, pp. 51–60, 2009. 10.1007/s11856-009-0062-5.

[21] A. Roman, "The np-completeness of the road coloring problem," *Inf. Process. Lett.*, vol. 111, pp. 342–347, March 2011.

[22] G.-V. Jourdan and G.v. Bochmann, "On testing 1-safe Petri nets," in *Theoretical Aspects of Software Engineering, 2009. TASE 2009. Third IEEE International Symposium on*, pp. 275 –281, 29-31 2009.

[23] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, pp. 541 –580, apr 1989.

[24] G. Bochmann and G.-V. Jourdan, "Testing k-safe Petri nets," in *Proceedings of the 21st IFIP WG 6.1 International Conference on Testing of Software and Communication Systems and 9th International FATES Workshop*, TESTCOM '09/FATES '09, (Berlin, Heidelberg), pp. 33–48, Springer-Verlag, 2009.

[25] H. Zhu and X. He, "A methodology of testing high-level Petri nets," *Information and Software Technology*, vol. 44, no. 8, pp. 473 – 489, 2002.

[26] E. Yamalidou and J. Kantor, "Modeling and optimal control of discrete-event chemical processes using Petri nets," *Computers & Chemical Engineering*, vol. 15, no. 7, pp. 503 – 519, 1991.

[27] E. Yamalidou, E. Adamides, and D. Bovin, "Optimal failure recovery in batch processing using Petri net models," *Proceedings of the 1992 American Control Conference*, vol. 3, pp. 1906–1910, 1992.

[28] A. Giua, C. Seatzu, and D. Corona, "Marking estimation of Petri nets with silent transitions," *Automatic Control, IEEE Transactions on*, vol. 52, pp. 1695 –1699, sept. 2007.

[29] L. Li and C. Hadjicostis, "Minimum initial marking estimation in labeled Petri nets," in *American Control Conference, 2009. ACC '09.*, pp. 5000 –5005, june 2009.

[30] M. Moalla, J. Pulou, and J. Sifakis, "Synchronized Petri nets : A model for the description of non-autonomous systems," in *Mathematical Foundations of Computer Science 1978* (J. Winkowski, ed.), vol. 64 of *Lecture Notes in Computer Science*, pp. 374–384, Springer Berlin / Heidelberg, 1978.

[31] J. Ezpeleta and L. Recalde, "A deadlock avoidance approach for non-sequential resource allocation systems," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 34, no. 1, pp. 93–101, 2004.

[32] Z. W. Li and M. C. Zhou, *Deadlock Resolution in Automated Manufacturing Systems: A Novel Petri Net Approach*. Springer Publishing Company, Incorporated, 1st ed., 2009.

[33] R. David and H. Alla, *Discrete, Continuous and Hybrid Petri Nets*. Springer-Verlag, 2004.

[34] F. Hennie, *Finite-State Models for Logical Machines*. New York: John Wiley, 2 ed., 1968.

[35] Z. Kohavi, *Switching and Finite Automata Theory*. The McGraw-Hill College, 2 ed., 1978.

[36] A. N. Trahtman, "Some results of implemented algorithms of synchronization," in *10th Journees Mentoises d'inform.*, (Liege, Belgium), September 8-11 2004.

[37] P. Buchholz and P. Kemper, "Hierarchical reachability graph generation for Petri nets," in *Universität Dortmund, Fachbereich Informatik, Forschungsbericht Nr. 660*, p. 2002, 1997.

[38] A. Farooq, H. Hejiao, and W. Xiaolong, "A technique for reachability graph generation for the Petri net models of parallel processes," *International Journal of Electrical and Electronics Engineering*, vol. 3:5, pp. 298–302, 2009.

[39] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2 ed., 2001.

[40] J. Colom, "The resource allocation problem in flexible manufacturing systems," in *Applications and Theory of Petri Nets 2003* (W. Aalst and E. Best, eds.), vol. 2679 of *Lecture Notes in Computer Science*, pp. 23–35, Springer Berlin Heidelberg, 2003.

[41] M. Pocci. The MATLAB toolbox is available at the following web address: `http://www.lsis.org/poccim/PN_SYNCHRO.zip`. September, 2012.

[42] A. Roman, "Synchronizing finite automata with short reset words," *Applied Mathematics and Computation*, vol. 209, no. 1, pp. 125 – 136, 2009.