Maejo Int. J. Sci. Technol. **2010**, *4*(03), 360-396

Maejo International Journal of Science and Technology

ISSN 1905-7873
Available online at www.mijst.mju.ac.th

Full Paper

Novel analysis of Petri-net-based controllers by means of TCT implementation tool of supervisory control theory

Gökhan Gelen and Murat Uzam*

Niğde Üniversitesi, Mühendislik-Mimarlık Fakültesi, Elektrik-Elektronik Mühendisliği Bölümü, Kampüs, 51200, Niğde, Turkey

*Corresponding author, E-mail: murat_uzam@hotmail.com or murat_uzam@nigde.edu.tr ;
Tel: + 90 388 225 22 82; Fax: + 90 388 225 01 12; URL: http://host.nigde.edu.tr/muzam/

Received: 24 March 2010 / Accepted: 2 August 2010 / Published: 7 September 2010

Abstract: The control of discrete event systems (DES) has been widely studied in the past two decades. Finite-state automata (FSA) and Petri nets (PN) are the two principal modelling formalisms for this study. Supervisory control theory (SCT), based on language and FSA concepts, is a well established framework for the study of discrete event control systems (DECS). PN-based approaches to the control design have been considered as an alternative framework. In the PN-based control of DES, given an uncontrolled PN model of a system and a set of specifications, a PN-based controller consisting of monitors (control places) is synthesised to solve the problem. In general, forbidden-state specifications are considered. Another heavily studied specification is to obtain the live system behaviour (non-blockingness in SCT terminology) for a given PN model by computing a PN-based controller. Unfortunately, PN-based analysis tools cannot deal with uncontrollable transitions. Therefore, to date there is no general technique for the correctness analysis of the computed PN-based controllers. This paper proposes a novel and general methodology to carry out the correctness analysis for the computed PN-based controllers by using the TCT implementation tool of SCT. Three examples are considered for illustration.

Keywords: discrete event systems (DES), Petri nets (PN), finite state automata (FSA), supervisory control theory (SCT)

1. Introduction

The control of discrete event systems (DES) has been widely studied in the past two decades. There are mainly two modelling formalisms for this study: finite-state automata (FSA) and Petri nets (PN). Supervisory control theory (SCT) is based on language and FSA concepts and was originally introduced to extend control theory concepts for continuous systems to the discrete event environment [1-4]. SCT is a well established framework for the study of discrete event control systems (DECS). On the other hand, PN-based approaches to the control design have been considered as an alternative

framework [5-9]. This is mainly because the state-space representation of PN as a vector addition system can result in a compact system description, thus keeping the net structure small even though the number of possible markings may become large. PN models lend themselves not only to the systematic construction of supervisory controllers, but also to the analysis of various qualitative properties and quantitative performance evaluation. The disadvantage of the PN models is that, in general, optimal supervisors need not exist within the class of PN even though the counterpart SCT problem may be solvable in the framework of automata.

In the PN-based control of DES, an uncontrolled PN model of a system represents the open-loop system. Without any means of control, the open-loop system would not behave as requested. Given an uncontrolled PN model of a system and a set of specifications, a PN-based controller consisting of monitors or control places is synthesised to solve the problem. When the uncontrolled PN model and the computed PN-based controller are augmented, the controlled model is obtained, which also represents the closed-loop system. In general, the specifications considered are of forbidden-state ones. Another heavily studied specification is to obtain live system behaviour (non-blockingness in SCT terminology) for a given PN model by computing a PN-based controller. Unfortunately, to date there is no general technique for the correctness analysis of the computed PN-based controllers. Heuristically, this analysis can be carried out by considering the reachability graph of a given PN model [10], but when dealing with complex models this method becomes impractical due to huge reachability graphs. In addition, it is not possible to deal with uncontrollable transitions with the currently available PN analysis tools. Therefore, it is crucial to obtain a general methodology to carry out the correctness analysis for the computed PN-based controllers. This paper proposes a general methodology to carry out the correctness analysis for the computed PN-based controllers by using the TCT implementation tool [11] of SCT. TCT (Toy Control Theory) is a standard software package developed under the supervision of Professor W. N. Wonham, the founder of SCT, for carrying out all necessary computations of SCT.

In brief, the correctness analysis is carried out as follows. To start with, it is assumed that an uncontrolled PN model, the specifications, and the PN-based controller, i.e. a set of monitors (control places), are given. Then the correctness analysis is carried out based on the following three main steps: (1) Transform the given problem from the PN domain to the automata domain and solve the problem by using TCT tool based on SCT. This transformation involves the representation of both uncontrolled PN model and the specifications as automata models. The proper supervisor obtained in this step is called RWSUPER and represents the maximally permissive and non-blocking behaviour for the considered problem; (2) Represent the PN-based controller as an automaton and obtain the controlled model of the system in the automata domain by using the synchronous product of automaton representation of the PN-based controller and automaton representation of the uncontrolled PN model. The supervisor obtained in this step is called PNSUPER and represents the controlled behaviour for the considered problem that survives under the supervision of the PN-based controller; and (3) Compare RWSUPER and PNSUPER. If these two are identical, then it can be concluded that the PN-based controller is maximally permissive and non-blocking.

The transformation from the PN domain to the automata domain has already been used [12-13]. The detailed treatment of this transformation involving more general classes of PN is proposed in this paper. It is well known that PN can be transformed into automata by means of the reachability graph [14]. Some recent work can also be seen [15-18] dealing with transformation from PN to automata. Transformation from automata to PN is also possible [19]. This transformation has already been applied to the control of DES [18, 20-22].

The purpose of this paper is to propose a general methodology for carrying out the correctness analysis for the computed PN-based controllers by using the TCT implementation tool of SCT. To show the applicability of the proposed method, three examples are considered in detail.

The remainder of the paper is organised as follows. PN with weighted input, output, inhibitor and enabling arcs are defined in Section 2. FSA and SCT are briefly reviewed in Section 3. Section 4 defines and explains the proposed mappings from PN to their automata representations. In Section 5, the proposed analysis technique of PN-based controllers by using TCT is explained. In Section 6, three

examples are considered in detail to show the applicability of the proposed analysis technique. Finally, conclusions are given in Section 7.

2. PN with Weighted Input, Output, Inhibitor and Enabling Arcs

PN are widely used as a formal tool for the design, analysis and control of DES. PN were named after Carl A. Petri, a contemporary German mathematician and computer scientist, who introduced a net-like mathematical tool for the study of communication with automata [23]. For PN basics the reader is referred to Peterson [24]. A weighted-arc PN is the one in which weights are associated with arcs. The arc mappings may take values over a set of all non-negative integers. In this case, each arc is said to have *multiplicity w*, where *w* represents the weight of arcs. *Ordinary* PN have a multiplicity of 1. The weight of an arc is indicated by a non-negative integer assigned to the arc. Unlike a standard PN, a PN with *inhibitor arcs* has the possibility of testing whether a place is void of tokens (*zero-testing ability*). Inhibitor arcs are a well known extension for standard PN. Inhibitor arcs are well suited to model a specific condition testing as opposed to producing and consuming tokens. The addition of inhibitor arcs gives PN the same modelling power as a Turing machine [25]. In this paper, weighted inhibitor arcs are considered in addition to ordinary inhibitor arcs. The former can be used for testing whether the number of tokens in a place is less than a certain threshold number [25]. As a complement of inhibitor arc, enabling arc was introduced by Uzam and Jones [26-27]. The introduction of enabling arcs causes PN to have *one testing ability*, i.e. the ability to test whether a place has a token(s). As with inhibitor arcs, enabling arcs are also well suited to model a specific condition testing. In this paper, weighted enabling arcs are considered in addition to ordinary enabling arcs. The former can be used for testing whether the number of tokens in a place is larger than or equal to a certain threshold number [26-27].

Formally a PN with weighted input, output, inhibitor and enabling arcs can be defined as follows:

$$PN = (P, T, Pre, Post, In, En, M_0)$$

where:

- $P = \{p_1, p_2, ..., p_n\}$ is a finite, non-empty set of places ;
- $T = \{t_1, t_2, ..., t_m\}$ is a finite, non-empty set of transitions, $P \cup T \neq \varnothing$ and $P \cap T = \varnothing$;
- Pre: $(P \times T) \rightarrow N$ is an input function that defines weighted input arcs from places to transitions, where N is a set of non-negative integers ;
- Post: $(T \times P) \rightarrow N$ is an output function that defines weighted output arcs from transitions to places ;
- In: $(P \times T) \rightarrow N$ is an inhibitor function that defines weighted inhibitor arcs from places to transitions ;
- En: $(P \times T) \rightarrow N$ is an enabling function that defines weighted enabling arcs from places to transitions ;
- $M_0 : P \rightarrow N$ is the initial marking.

PN with weighted input, output, inhibitor and enabling arcs consist of two types of nodes called places, represented by circles (○), and transitions, represented by bars (▬). There are three types of arcs used, namely ordinary (input or output) arc, represented by a directed arrow (→), inhibitor arc, represented by an arrow whose end is a circle (—○), and finally enabling arc, represented by a directed arrow whose end is empty (—▷). Weighted and directed ordinary arcs connect places to transitions and vice versa, whereas weighted enabling arcs and inhibitor arcs connect only places to transitions. The number of tokens in places represents the current state of the system and transitions represent events. Each transition has a set of input and output places, which represent the pre-condition and post-condition of the transition respectively. The marking of the PN is represented by the number of tokens in each place. Tokens are represented by black dots (•). Movement of tokens between places describes the evolution of the PN and is accomplished by the firing of the enabled transitions. "Enabling rules" and "firing rules" are associated with transitions. The enabling rules state the conditions under which transitions are allowed to fire. The firing rules define the marking modification induced by the transition firing. The enabling rules define the conditions that allow a transition *t* to fire,

and the firing rules specify the change of state produced by the transition. Both the enabling and the firing rules are specified through arcs. In particular, the enabling rules involve input arcs, inhibitor arcs and enabling arcs, while the firing rules depend on input arcs and output arcs. Note that input arcs play a double role as they are involved both in enabling and in firing. The following enabling and firing rules are used to govern the flow of tokens.

Enabling Rules: In a PN with weighted input, output, inhibitor and enabling arcs, there are three rules which define whether a transition is enabled to fire. These pre-conditions must be satisfied for a transition to fire:

1. If an input place p of a transition t is connected to t with a weighted input arc $Pre(p,t)$, then t is said to be enabled when p contains at least the number of tokens equal to the weight of the directed input arc, i.e. $M(p) \geq Pre(p,t)$. This is shown in Figure 1(a), where t1 is enabled if there are two or more tokens in p1. Therefore, t1 is enabled as p1 contains three tokens, i.e. $M(p1) = 3$ and Pre(p1,t1) = 2.
2. If an input place p of a transition t is connected to t with a weighted inhibitor arc $In(p,t)$, then t is said to be enabled when p contains fewer tokens than the weight of the inhibitor arc, i.e. $M(p) < In(p,t)$. This is shown in Figure 1(b), where t1 is enabled provided that there are two or more tokens in p1 and there are fewer than three tokens in p3. Therefore, t1 is enabled as $M(p1) = 3$ with Pre(p1,t1) = 2, and $M(p3) = 2$ with In(p3,t1) = 3.
3. If an input place p of a transition t is connected to t with a weighted enabling arc $En(p,t)$, then t is said to be enabled when p contains at least a number of tokens equal to the weight of the enabling arc, i.e. $M(p) \geq En(p,t)$. This is shown in Figure 1(c), where t1 is enabled provided that there are two or more tokens in p1 and there are at least four tokens in p3. Therefore, t1 is enabled as $M(p1) = 3$ with Pre(p1,t1) = 2, and $M(p3) = 4$ with En(p3,t1) = 4.

Transition t1 in Figure 1(d) represents all enabling rules at the same time. It is enabled if $M(p1) \geq 2$ due to Pre(p1,t1) = 2, and if $M(p2) < 3$ due to In(p2,t1) = 3, and if $M(p3) \geq 4$ due to En(p3,t1) = 4. It can be seen that t1 is enabled with the initial marking given in Figure 1(d).

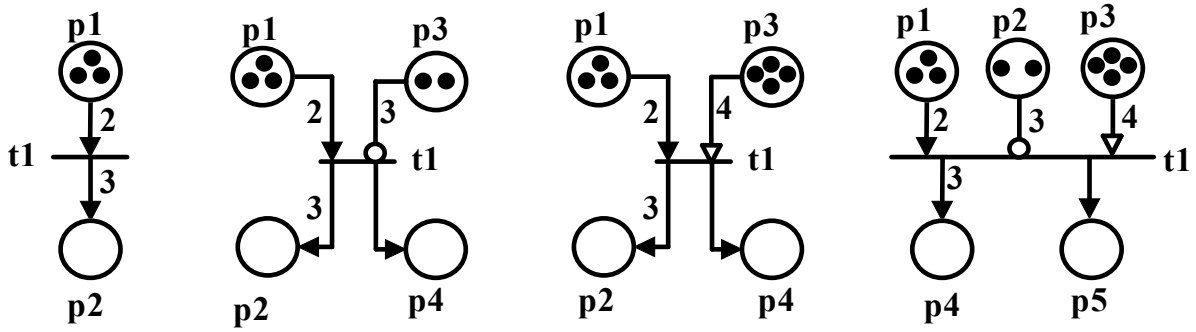
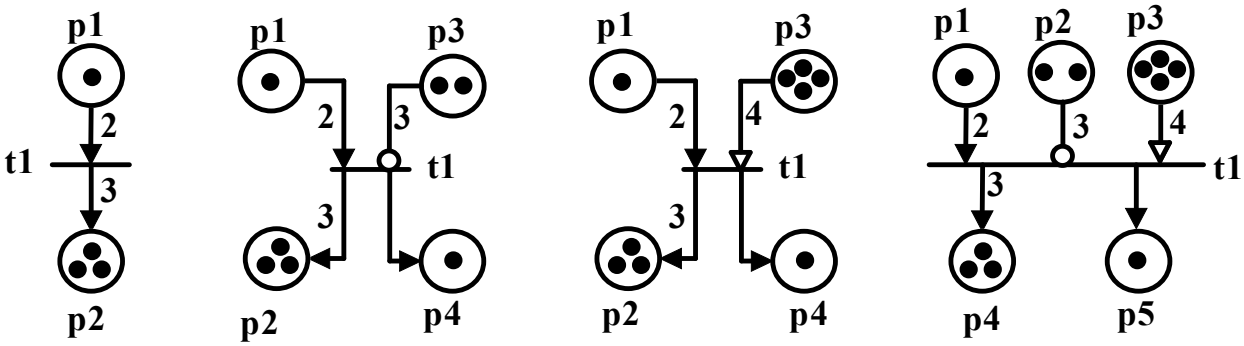


Figure 1. Illustration of enabling rules

Firing Rules: In a PN with weighted input, output, inhibitor and enabling arcs, when an enabled transition t fires, it removes $Pre(p_i,t)$ tokens from each input place p_i and deposits, at the same time, $Post(t,p_o)$ tokens in each output place p_o. The firing of an enabled transition t does not change the marking of the input places that are connected to t only by weighted enabling arcs or weighted inhibitor arcs. Firing rules can be followed from Figure 1 and Figure 2. As explained, all transitions shown in Figure 1 are enabled. When these transitions are fired the new markings are obtained as shown in Figure 2, based on the firing rules.



(a) (b) (c) (d)

Figure 2. Illustration of firing rules

The firing of an enabled transition changes the marking, i.e. the token distribution of a PN. A marking M_i is said to be *reachable* from an initial marking M_0 if there exists a sequence of firings that can transform M_0 to M_i. A firing sequence is represented by $\sigma = t_1, t_2, t_3.....t_m$. To show M_i is reachable from M_0 by σ, the following representation is used: $M_0 [\sigma > M_i$. A PN is said to be *ordinary* if the arcs are not weighted (by default the weight of an arc is 1). A PN is said to be *k-bounded* or *bounded* if the number of tokens in each place does not exceed a finite number 'k' for every marking reachable from the initial marking M_0. A PN is said to be *safe* if all its places are safe. A place 'p' is safe if it contains no more than one token. In other words, a PN is called *safe* if it is 1-bounded. A PN is said to be *conservative* if the total number of tokens in all its places for all reachable markings is constant. A transition is said to be *live* if for all markings of the PN there is a firing sequence which takes the net to a marking, in which the transition is enabled. A PN is *live* if all its transitions are live. A live PN indicates the absence of *deadlocks* in the operation of the modelled system. A PN is said to be *reversible* if the initial marking M_0 is reachable from each marking. In this paper, bounded PN are considered. In later parts of this paper, safe and ordinary PN (SOPN) with inhibitor and enabling arcs are considered.

3. FSA and SCT

FSA are a classical tool used for many years for the study of DES. A finite automaton incorporates both principal system features, i.e. system states and system transitions in an abstract form. A FSA (or simply automaton), denoted by G, is a six-tuple $G=(Q, \Sigma, f, \Gamma, q_0, Q_m)$, where Q is the set of states, Σ is the finite event set, $f: Q\times\Sigma\rightarrow Q$ is the partial transition function, $\Gamma: q \rightarrow 2^{\Sigma}$ is the active event function, $\Gamma(q)$ is the set defined for every state of G and represents the feasible events of q, q_0 is the initial state and $Q_m \subseteq Q$ is the set of marked states representing the completion of a given task or operation. A simple automaton model with two states is shown in Figure 3. This automaton has two states labelled with q0 and q1; q0 with a double arrow is the initial (and marked) state while q1 with an exiting arrow is the marked state of the automaton. A directed arrow represents the transition functions of the automaton. Labels of transitions (e1, e2) correspond to events.

The language generated by G is denoted by L(G) and is defined as $L(G) = \{s\in \Sigma^*: f(q_0,s)$ is defined$\}$. The language marked by G is denoted by $L_m(G)$ and is defined as $L_m(G) = \{s\in\Sigma^*: f(q_0,s)\in Q_m\}$. The DES modelled as automaton G is said to be non-blocking if $L(G)=\overline{L_m(G)}$. DES named as A and B can be composed of a synchronous product (parallel composition) operation. The synchronous product of two automata is denoted by A||B and represents the synchronous behaviour of two automata. In the resulting automaton, common events occur synchronously while the other events occur asynchronously [28-29].

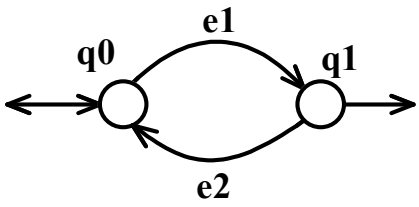


Figure 3. A simple automaton model

SCT was introduced to extend control theory concepts for continuous systems to the discrete-event environment [2, 4]. In SCT, a *forbidden state problem* [2] specifies the conditions that must be avoided, typically the simultaneous utilisation of some resource by two or more users. In addition, SCT generally requires the controlled system to be non-blocking (namely that specified target states, often just the initial state, be maintained reachable), and to be *maximally permissive*, i.e. to permit the occurrence of all events not leading to violation of the foregoing requirements. DES evolve on spontaneously occurring events. Let $\sum$ be a finite set of events; the set of all finite concatenations of events in $\sum$ is denoted by $\sum^*$. An element of this set is called a string. The number of events gives the length of the string. A string with no element is denoted by ε and is called empty string. A subset $L \subseteq \sum^*$ is called a language over $\sum$. For a string $s \in \sum^*$, $\overline{s}$ denotes the prefix of s and is defined as $\overline{s} = \{s_p \in \sum^* \mid \exists t \in \sum^* (s_p t = s)\}$. Extension of this definition to language prefix closure of a language L is denoted by $\overline{L}$. A language L satisfying the condition $L = \overline{L}$ is said to be prefix-closed [28-29].

The SCT makes use of formal languages to model the uncontrolled behaviour of DES (plant) and specifications for the controlled behaviour. The objective is to restrict the behaviour of the system to a desired one, which is represented by the specifications. This is done by disabling some events to prevent the occurrence of some undesired strings in the system. The disabling action is accomplished by another simultaneously executing automaton called the supervisor. The system cannot be forced by the supervisor to generate new events. In SCT, events are divided into two disjoint sets, i.e. controllable events and uncontrollable events. These sets are denoted by $\sum_c$ and $\sum_{uc}$ respectively. The supervisor has no effect on uncontrollable events, which means that the supervisor cannot disable uncontrollable events. The existence of a supervisor is guaranteed if the desired language satisfies controllability conditions. This condition is defined as $\overline{K} \sum_{uc} \cap M \subseteq \overline{K}$, where $\overline{K}$ is the language that will be generated under the control of supervisor and M is the language generated by the uncontrolled system.

The behaviour of a DES under the control of supervisor S is denoted by L(S/G). A language K is called $L_m(G)$-closed when the language satisfies condition $K = \overline{K} \cap L_m(G)$. If the desired behaviour $\overline{K} \cap L_m(G)$ is controllable and $L_m(G)$-closed, then $\overline{K} = L(S/G) = \overline{L_m(S/G)}$, and this means that the controlled behaviour is non-blocking [29].

4. Mappings from PN to FSA

4.1 Mappings from SOPN to FSA

In this section, a set of mappings from SOPN to FSA is proposed. The mapping strategy is based on the representation of a SOPN place by an automaton with two states. The transitions between these states or self-looped transition from a state define the behaviour of the modelled SOPN. Each transition of a SOPN is represented by a transition with an associated event within its automaton counterpart. Note that the transitions considered in this section are all shown as uncontrollable, but it is also possible to consider them as controllable. In this respect there is no difference. Also note that the concurrent firing of enabled transitions is not considered and therefore at any time only one transition can fire. The conversion from a set of fundamental SOPN modules utilised in the modelling of DES to FSA is presented as follows.

4.1.1. Mapping for a single place with an input and an output transition

A single place with one input transition and one output transition is shown in Figure 4, together with its automaton representation. In Figures 4(a) and 4(b), the initial markings of p1 are $M_0(p1) = 0$ and 1 respectively. In this SOPN, t1 and t2 are labelled with 'a' and 'b' respectively to represent the respective events. In Figure 4(a), initially t1 is enabled. If it is fired, a token is deposited in p1. If p1 contains a token, then t2 is enabled to fire. When p1 contains a token and t2 is fired, the token is removed from p1. In the automaton representation of the SOPN shown in Figure 4(a), there are two states: s0 is the initial and marker state and represents the absence of a token in p1, while s1 represents the presence of a token in p1. At the initial state, if event 'a' occurs, then automaton changes its state from s0 to s1. When the automaton is in s1 and event 'b' occurs, then automaton changes its state from s1 to s0. In Figure 4(b), initially t2 is enabled. If it is fired, the token is removed from p1. When p1 is void of token, t1 is enabled to fire. When t1 is fired, a token is deposited in p1. In the automaton representation of the SOPN shown in Figure 4(b), there are two states: s0 is the initial and marker state and represents the presence of a token in p1, while s1 represents the absence of a token in p1. At the initial state, if event 'b' occurs, then automaton changes its state from s0 to s1. When the automaton is in s1 and event 'a' occurs, then automaton changes its state from s1 to s0.

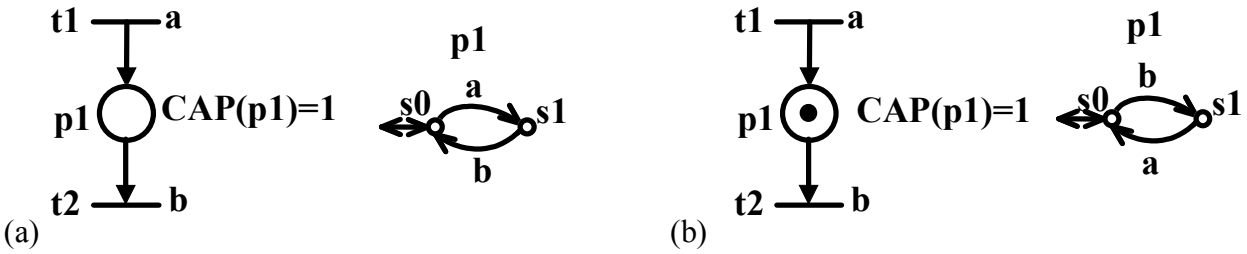


Figure 4. A single place with one input transition and one output transition and its automaton representation: (a) with the initial marking $M_0(p1) = 0$; (b) with the initial marking $M_0(p1) = 1$

4.1.2. Mapping for choice

A choice can be made between two or more activities (processes) in order to let just one of them start. To model this property in PN, a place with more than one output transition is used. A choice example is shown in Figure 5, together with its automaton representation. In Figures 5(a) and 5(b), the initial markings of p1 are $M_0(p1) = 0$ and 1 respectively. In this SOPN, t1, t2 and t3 are labelled with 'a', 'b' and 'c' respectively to represent the respective events. The mapping for choice is obvious from both figures and the state change from one state to another takes place based on the occurrence of the respective events as explained in the previous mappings.

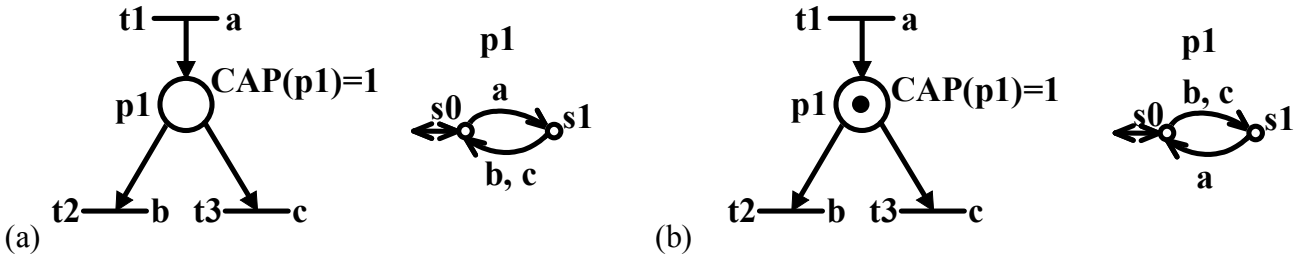


Figure 5. A choice example between transitions t2 and t3 and its automaton representation: (a) with the initial marking $M_0(p1) = 0$; (b) with the initial marking $M_0(p1) = 1$

4.1.3. Mapping for merge

Two or more activities (processes) can be merged. To model this property in PN, a place with more than one input transition is used. A merge example is shown in Figure 6, together with its automaton representation. In Figures 6(a) and 6(b), the initial markings of p1 are $M_0(p1) = 0$ and 1 respectively. In this SOPN, t1, t2 and t3 are labelled with 'a', 'b' and 'c' respectively to represent the respective events. The mapping for merge is obvious from both figures and the state change from one state to another takes place based on the occurrence of the respective events as explained in the previous mappings.

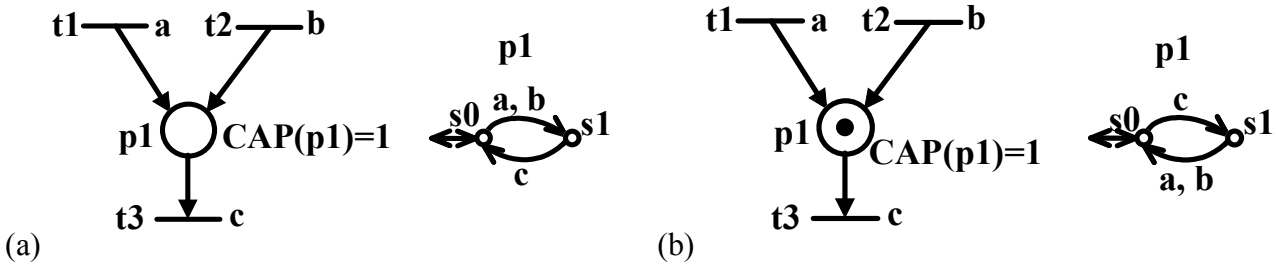


Figure 6. A merge example between transitions t2 and t3 and its automaton representation: (a) with the initial marking $M_0(p1) = 0$; (b) with the initial marking $M_0(p1) = 1$

4.1.4. Mapping for fork (concurrency)

Fork (concurrency) represents two or more concurrent processes initiated at the same time. To model this property in PN, a transition with more than one output place is used. A fork example is shown in Figure 7, together with its automaton representation. Initially, it is assumed that there are no tokens in p1 and p2. In this SOPN, t1, t2 and t3 are labelled with 'a', 'b' and 'c' respectively to represent the respective events. Initially only t1 is enabled. When there is no token in p1 and p2 and t1 is fired, then a token each is put in p1 and p2. This means that both activities represented by p1 and p2 are initiated at the same time. When there is a token in p1 (or p2), t2 (respectively t3) is enabled. When p1 (or p2) contains a token and t2 (respectively t3) is fired, the token is removed from p1 (respectively p2). Each place is represented by a two-state automaton as shown in Figure 7. Then the automaton representation of the SOPN, shown in Figure 7, is obtained by the synchronous product p1||p2 of automata models p1 and p2. In the automaton model p1||p2, there are four states, $Q = \{s0, s1, s2, s3\}$, representing the SOPN markings $M(p1,p2)^T = \{00, 11, 01, 10\}$ respectively. The state change from one to another takes place based on the occurrence of the respective events as explained in the previous mappings.

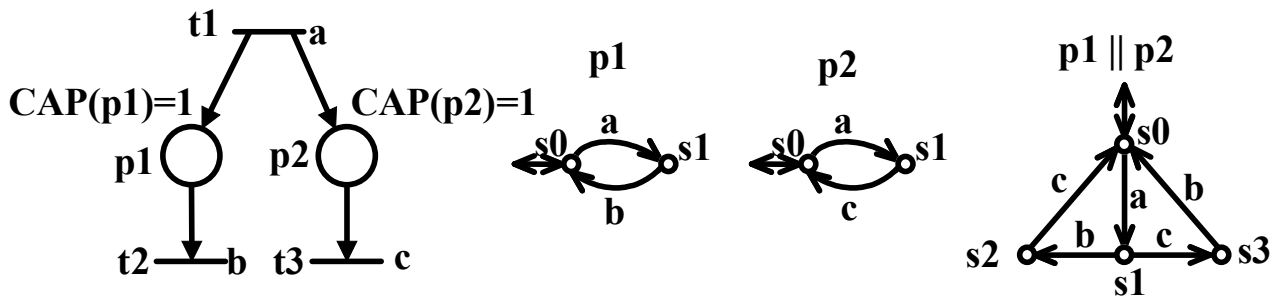


Figure 7. A fork (concurrency) example for p1 and p2 and its automaton representation

4.1.5. Mapping for join (synchronisation)

Two or more activities (processes) can be joined (synchronised). To model this property in PN, a transition with more than one input place is used. A join example is shown in Figure 8, together with its automaton representation. To represent the respective events, t1, t2 and t3 are labelled with 'a', 'b' and 'c' respectively. Initially both t1 and t2 are enabled. When there is no token in p1 (or p2) and t1 (respectively t2) is fired, then a token is put in p1 (respectively p2). When there is a token each in p1 and p2, this means that both activities represented by p1 and p2 are ready to be joined (synchronised). When there is a token each in p1 and p2 and t3 is fired, the tokens are removed from p1 and p2 and thus the two processes are joined (synchronised). Each place is represented by a two-state automaton as shown in Figure 8. Then the automaton representation of the SOPN, shown in Figure 8, is obtained by the synchronous product p1||p2 of automata models p1 and p2. In the automaton model p1||p2, there are four states, Q = {s0, s1, s2, s3}, representing the SOPN markings $M(p1,p2)^T$ = {00, 11, 10, 01} respectively. The state change from one to another takes place based on the occurrence of the respective events as explained in the previous mappings.

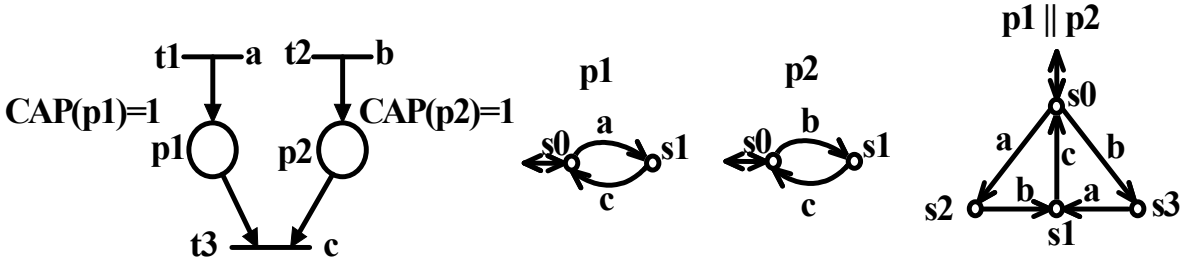


Figure 8. A join (synchronisation) example for p1and p2 and its automaton representation

4.1.6. Mapping for an inhibitor arc

A single place with one input transition, one output transition and an inhibitor arc In(p1,t3) is shown in Figure 9, together with its automaton representation. In Figures 9(a) and 9(b), the initial markings of p1 are $M_0(p1)$ = 0 and 1 respectively. In this SOPN, t1, t2 and t3 are labelled with 'a', 'b' and 'c' respectively to represent the respective events. In Figure 9(a), initially both t1 and t3 are enabled and can fire. If t1 is fired, a token is deposited in p1. If p1 contains a token, then t2 is enabled to fire but t3 is not enabled and therefore cannot fire. When p1 contains a token and t2 is fired, the token is removed from p1. In brief, when there is no token in p1, t3 is enabled to fire but when there is a token in p1, t3 is not enabled to fire. In the automaton representation of the SOPN shown in Figure 9 (a), there are two states: s0 is the initial and marker state and represents the absence of a token in p1, while s1 represents the presence of a token in p1. At the initial state, the self-looped transition [s0, c, s0] allows event 'c' to occur. If the automaton is in s0 and if event 'a' occurs, then it changes its state from s0 to s1. When the automaton is in s1, event 'c' is not allowed to occur. When the automaton is in s1 and event 'b' occurs, then it changes its state from s1 to s0. In Figure 9(b), initially only t2 is enabled and since there is a token in p1, t3 is not enabled. If t2 is fired, the token is removed from p1. When p1 is void of token, both t1 and t3 are enabled to fire. When t1 is fired, a token is deposited in p1. In the automaton representation of the SOPN shown in Figure 9(b), there are two states: s0 is the initial and marker state and represents the presence of a token in p1, while s1 represents the absence of a token in p1. When the automaton is in s0, event 'c' is not allowed to occur. At the initial state, if event 'b' occurs, then the automaton changes its state from s0 to s1. When the automaton is in s1, event 'c' is allowed to occur by means of the self-looped transition [s1, c, s1]. When the automaton is in s1 and event 'a' occurs, then the automaton changes its state from s1 to s0.

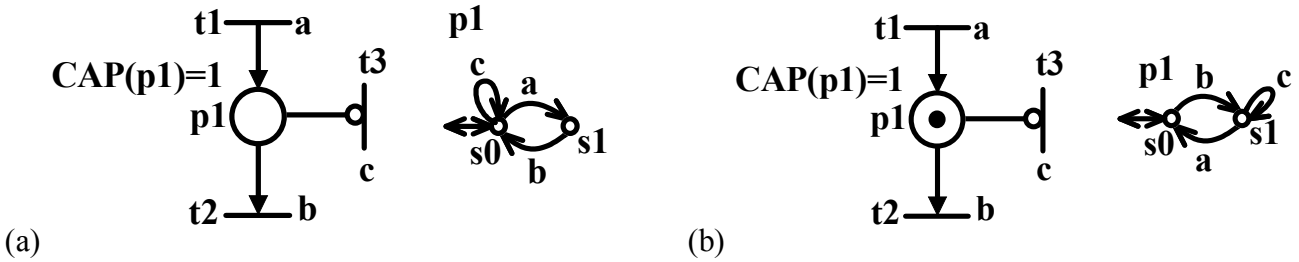


Figure 9. A place p1 with one input transition, one output transition and an inhibitor arc In(p1,t3), and its automaton representation: a) with the initial marking $M_0(p1) = 0$; b) with the initial marking $M_0(p1) = 1$

4.1.7. Mapping for an enabling arc

A single place with one input transition, one output transition and an enabling arc En(p1,t3) is shown in Figure 10, together with its automaton representation. In Figures 10(a) and 10(b), the initial markings of p1 are $M_0(p1) = 0$ and 1 respectively. In this SOPN, t1, t2 and t3 are labelled with 'a', 'b' and 'c' respectively to represent the respective events. In Figure 10(a), initially only t1 is enabled and since there is no token in p1, t3 is not enabled. If t1 is fired, a token is put in p1. When there is a token in p1, both t2 and t3 are enabled to fire. When there is a token in p1 and t3 is fired, the token remains in p1. When there is a token in p1 and t1 is fired, the token is removed from p1. In the automaton representation of the SOPN shown in Figure 10(a), there are two states: s0 is the initial and marker state and represents the absence of a token in p1, while s1 represents the presence of a token in p1. When the automaton is in s0, event 'c' is not allowed to occur. At the initial state, if event 'a' occurs, then the automaton changes its state from s0 to s1. When the automaton is in s1, event 'c' is allowed to occur by means of the self-looped transition [s1,c,s1]. When the automaton is in s1 and event 'a' occurs, then the automaton changes its state from s1 to s0. In Figure 10(b), initially both t1 and t3 are enabled and can fire. When there is a token in p1 and t3 is fired, the token remains in p1. When there is a token in p1 and t1 is fired, the token is removed from p1. If p1 contains no token, then t1 is enabled to fire but t3 is not enabled and therefore cannot fire. When p1 contains no token and t1 is fired, a token is put in p1. In brief, when there is a token in p1, t3 is enabled to fire, but when there is no token in p1, t3 is not enabled to fire. In the automaton representation of the SOPN shown in Figure 10(b), there are two states: s0 is the initial and marker state and represents the presence of a token in p1, while s1 represents the absence of a token in p1. At the initial state, the self-looped transition [s0,c,s0] allows event 'c' to occur. When the automaton is in s0 and event 'b' occurs, then the automaton changes its state from s0 to s1. When the automaton is in s1, event 'c' is not allowed to occur. When the automaton is in s1 and event 'a' occurs, then the automaton changes its state from s1 to s0.

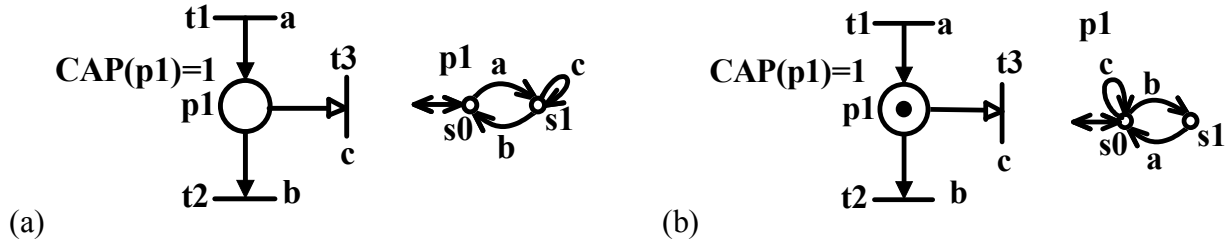


Figure 10. A place p1 with one input transition, one output transition and an enabling arc En(p1,t3), and its automaton representation: (a) with the initial marking $M_0(p1) = 0$; (b) with the initial marking $M_0(p1) = 1$

4.1.8. Mapping for mutual exclusion

Mutual exclusion is used to avoid the simultaneous use of a common resource, examples of which include machines, robots and fixtures in the manufacturing context. There may be two or more processes trying to access the common resource at the same time. Consider a mutual exclusion example shown in Figure 11, together with its automaton representation. Initially, it is assumed that there are no tokens in p1 and p2 while there is a token in p3. In this SOPN, t1, t2, t3 and t4 are labelled with 'a', 'b', 'c' and 'd' respectively to represent the respective events. In this SOPN, p1 and p2 represent two processes using a common resource represented by a token in p3. Here, p3 ensures the following: M(*p1*) + M(*p2*) + M(*p3*) = 1. This means that at any time the sum of tokens in places p1, p2 and p3 is just 1. Initially both t1 and t3 are enabled. When there is no token in p1 (or p2) and there is a token in p3, if t1 (respectively t3) is fired, then the token is removed from p3 and a token is put in p1 (respectively p2). This means that only one activity represented by t1 or t3 can be initiated. When there is a token in p1 (or p2), t2 (respectively t4) is enabled. When p1 (or p2) contains a token and t2 (respectively t4) is fired, the token is removed from p1 (respectively p2) and a token is deposited in p3. Each place is represented by a two-state automaton as shown in Figure 11. Then, the automaton representation of the SOPN shown in Figure 11 is obtained by the synchronous product p1||p2||p3 of automata models p1, p2 and p3. In the automaton model p1||p2||p3, there are three states, Q = {s0, s1, s2}, representing the following SOPN markings: $M(p1,p2,p3)^T$ = {001, 100, 010} respectively. The state change from one state to another takes place based on the occurrence of the respective events as explained in the previous mappings.

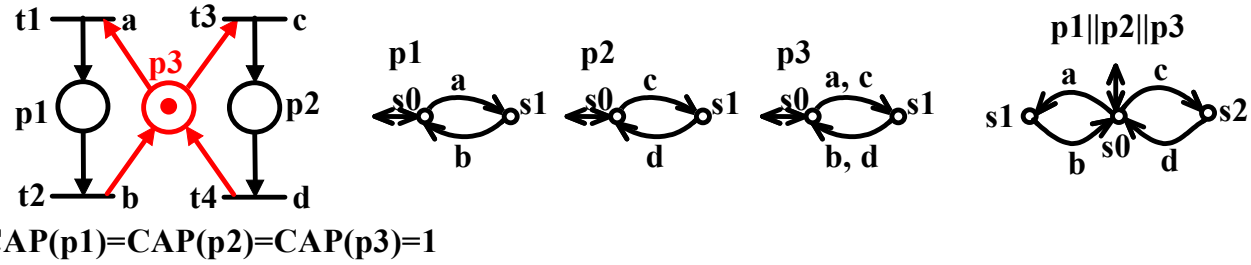


Figure 11. A mutual exclusion example in SOPN and its automaton representation

The place p3 shown in Figure 11 is called a control place or a monitor and is widely used in the literature to enforce control specifications. In this particular example the specification to be enforced is M(p1) + M(p2) ≤ 1. This means that the sum of tokens in places p1 and p2 must be less than or equal to 1. At this point an alternative way to enforce the same specification is considered.

Consider a mutual exclusion example shown in Figure 12, together with its automaton representation. Initially, it is assumed that there are no tokens in p1 and p2. Initially both t1 and t3 are enabled, as both p1 and p2 are void of tokens. When there is no token in p1 (or p2) and t1 (respectively t3) is fired, then a token is put in p1 (respectively p2). When there is a token in p1 (respectively p2), the inhibitor arc In(p1,t3) (respectively In(p2,t1)) inhibits t3 (respectively t1) from firing. This means that only one activity represented by t1 or t3 can be initiated. When there is a token in p1 (or p2), t2 (respectively t4) is enabled. When p1 (or p2) contains a token and t2 (respectively t4) is fired, the token is removed from p1 (respectively p2). Each place is represented by a two-state automaton as shown in Figure 12. Then the automaton representation of the SOPN shown in Figure 12 is obtained by the synchronous product p1||p2 of the automata models p1 and p2. In the automaton model p1||p2, there are three states, Q = {s0, s1, s2}, representing the following SOPN markings: $M(p1,p2)^T$ = {00, 10, 01} respectively. The state change from one to another takes place based on the occurrence of the respective events as explained in the previous mappings. As can be seen, the synchronous products p1||p2||p3 and p1||p2 in Figures 11 and 12 respectively are identical. In other words, these automata generate the same language and therefore the SOPN shown in Figure 11 and the one shown in Figure 12 enforce the same specification. In this respect, these two SOPN are said to be "control equivalent."

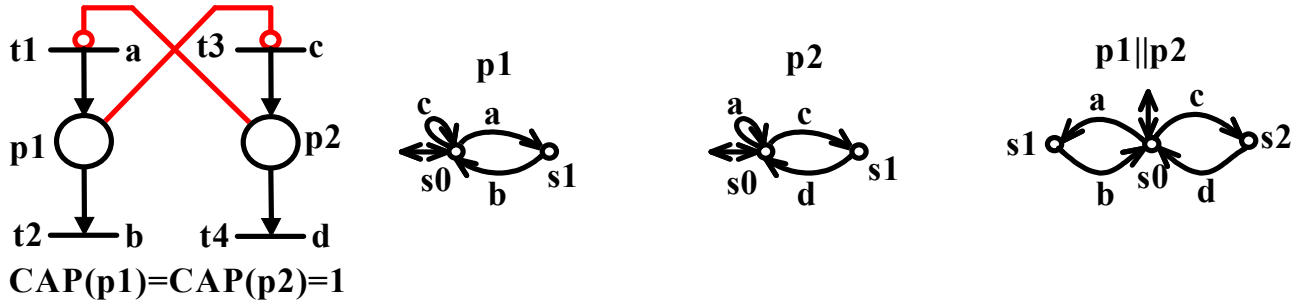


Figure 12. An alternative way to enforce the mutual exclusion specification M(*p1*) + M(*p2*) ≤ 1 in SOPN, and its automaton representation

4.2. Mappings from PN with Weighted Arcs to FSA

In this section, a set of mappings from PN with weighted input, output, inhibitor and enabling arcs to FSA is proposed. The first requirement for mapping a PN with weighted input, output, inhibitor and enabling arcs to FSA is that the PN considered must be bounded; otherwise the mapping is not possible. The token capacity of a given place *p*, represented as CAP(*p*), must be known to start the mapping. Then each place *p* of a given PN model is considered separately, together with its input, output, inhibitor and/or enabling arcs. If *p* is bounded in accordance with CAP(*p*) ≤ b, for instance, then *p* is modelled by a buffer with capacity b (i.e. b+1 states). For example, if a PN place *p* has a token capacity of 3, i.e. CAP(*p*) ≤ 3, then it can be represented by 4 states. The transitions (including the self-loops) between states of the automaton characterise the behaviour of the PN modelled. Every transition of *p* is represented by a transition within the automaton model of *p*. For the examples considered here, a label is assigned for each transition to represent the respective event. Note that the transitions considered in this section are all shown as uncontrollable, but it is also possible to consider them as controllable. In this respect, there is no difference. Also note that the concurrent firing of enabled transitions is not considered and therefore at any time only one transition can fire. In this section, some important mapping examples are considered. In these examples, only one place is considered with different arcs to show the proposed method.

4.2.1. Mapping for a single place with a token capacity CAP(*p*) ≤ b

Here, two mapping examples are considered. As can be seen from Figure 13, the first example involves mapping for a single place p1 with a token capacity CAP(*p1*) ≤ 4, together with an ordinary input arc Pre(t1,p1) = 1 and an ordinary output arc Post(p1,t2) = 1. Initially it is assumed that p1 is void of tokens. As long as the number of tokens in p1 is fewer than 4, every firing of t1 deposits a token in p1. Provided that there is a token(s) in p1, every firing of t2 removes a token from p1. To represent the respective events, t1 and t2 are labelled with 'a' and 'b' respectively. Since p1 is bounded in accordance with CAP(*p*) ≤ 4, it is modelled by a buffer with capacity 5 as shown in Figure 13. State s0 is the initial and marker state and represents the absence of tokens in p1. States s1, s2, s3 and s4 represent the presence of 1, 2, 3 and 4 tokens in p1 respectively. In the automaton model, the occurrence of event 'a' represents both the firing of t1 and the depositing of a token in p1, while the occurrence of event 'b' represents both the firing of t2 and the removing of a token from p1.

As can be seen from Figure 14, the second example involves mapping for a single place *p1* with a token capacity CAP(*p1*) ≤ 6, together with an ordinary input arc Pre(t1,p1) = 1, an ordinary output arc Post(p1,t2) = 1, a weighted input arc Pre(t3,p1) = 2, and a weighted output arc Post(p1,t4) = 3. Initially it is assumed that p1 is void of tokens. As long as the number of tokens in p1 is fewer than 6, every firing of t1 deposits a token in p1. Similarly, if the number of tokens in p1 is fewer than 5, every firing of t3 deposits two tokens in p1. Provided that there is a token(s) in p1, every firing of t2 removes a token from p1. Similarly, if the number of tokens in p1 is greater than or equal to 3, every

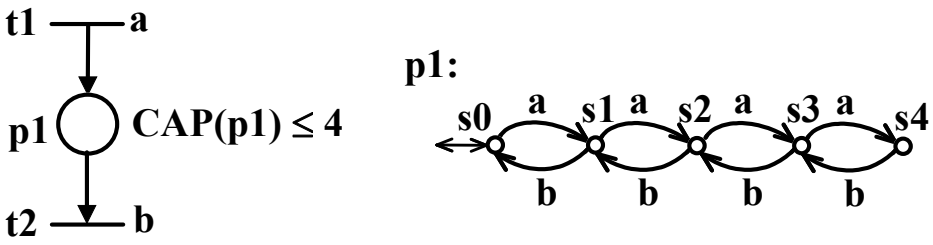


Figure 13. A single place with a token capacity CAP(*p1*) ≤ 4, together with an ordinary input arc Pre(t1,p1) = 1 and an ordinary output arc Post(p1,t2) = 1, and its automaton representation

firing of t4 removes three tokens from p1. To represent the respective events, t1, t2, t3 and t4 are labelled with 'a', 'b', 'c' and 'd' respectively. Since p1 is bounded in accordance with CAP(*p*) ≤ 6, it is modelled by a buffer with capacity 7 as shown in Figure 14. State s0 is the initial and marker state and represents the absence of tokens in p1. States s1, s2, s3, s4, s5 and s6 represent the presence of 1, 2, 3, 4, 5 and 6 tokens in p1 respectively. In the automaton model, the occurrence of event 'a' represents both the firing of t1 and the depositing of a token in p1, while the occurrence of event 'b' represents both the firing of t2 and the removing of a token from p1. Likewise, the occurrence of event 'c' represents both the firing of t3 and the depositing of two tokens in p1, while the occurrence of event 'd' represents both the firing of t4 and the removing of three tokens from p1.

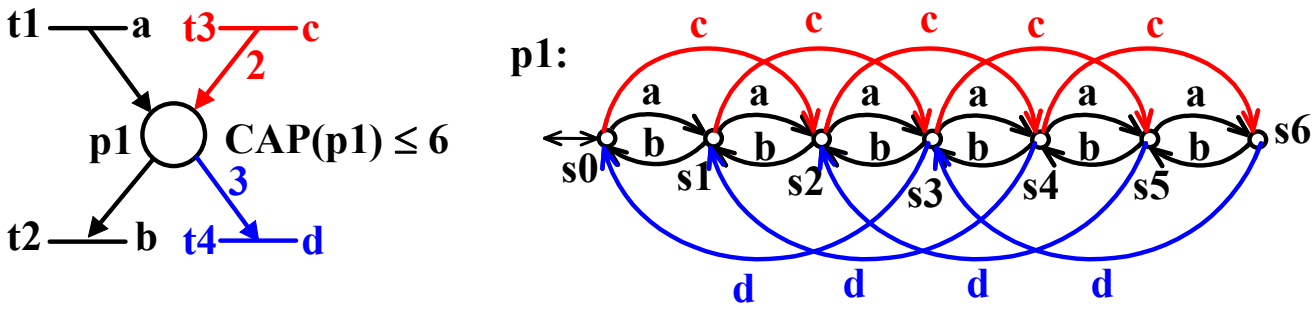


Figure 14. A single place with a token capacity CAP(*p1*) ≤ 6, together with a weighted input arc Pre(t3,p1) = 2 and a weighted output arc Post(p1,t4) = 3, and its automaton representation

4.2.2. Mapping for weighted inhibitor arc

Mapping for weighted inhibitor arc is explained by means of an example in this section. As can be seen from Figure 15, this example involves mapping for a single place p1 with a token capacity CAP(*p1*) ≤ 5, together with an ordinary input arc Pre(t1,p1) = 1, an ordinary output arc Post(p1,t2) = 1 and a weighted inhibitor arc In(p1,t3) = 4. Initially it is assumed that p1 is void of tokens. As long as the number of tokens in p1 is fewer than 5, every firing of t1 deposits a token in p1. Provided that there is a token(s) in p1, every firing of t2 removes a token from p1. As long as the number of tokens in p1 is fewer than 4, t3 is allowed to fire; otherwise it is not allowed to fire. The firing of t3 does not change the number of tokens present in p1. To represent the respective events, t1, t2, and t3 are labelled with 'a', 'b' and 'c' respectively. Since p1 is bounded in accordance with CAP(*p*) ≤ 5, it is modelled by a buffer with capacity 6 as shown in Figure 15. State s0 is the initial and marker state and represents the absence of tokens in p1. States s1, s2, s3, s4 and s5 represent the presence of 1, 2, 3, 4 and 5 tokens in p1 respectively. In the automaton model, the occurrence of event 'a' represents both the firing of t1 and the depositing of a token in p1, while the occurrence of event 'b' represents both the firing of t2 and the removing of a token from p1. Self-loops [s0,c,s0], [s1,c,s1], [s2,c,s2] and [s3,c,s3] are added to states s0, s1, s2 and s3 respectively to represent that the occurrence of event 'c' is allowed in these

states. On the other hand, as there are no such self-loops assigned to states s4 and s5, the occurrence of event ‘c’ is not allowed in these states.

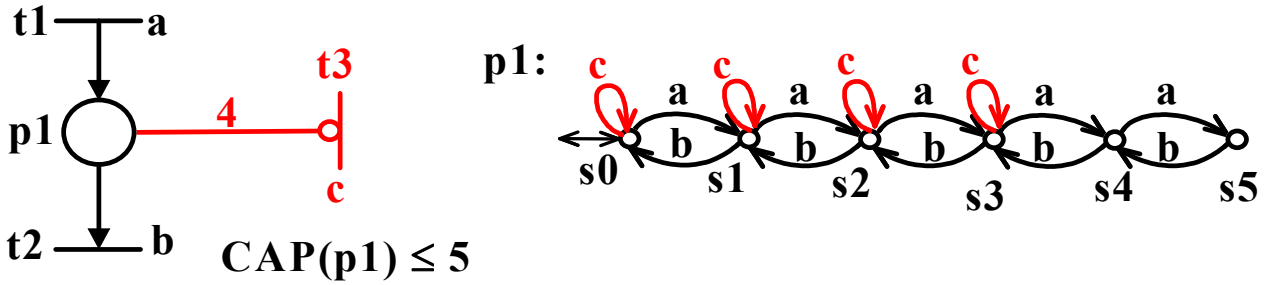


Figure 15. A single place with a token capacity CAP(*p1*) ≤ 5, together with a weighted inhibitor arc In(p1,t3) = 4 and its automaton representation

4.2.3. Mapping for weighted enabling arc

Mapping for weighted enabling arc is explained by means of an example in this section. As can be seen from Figure 16, this example involves mapping for a single place p1 with a token capacity CAP(*p1*) ≤ 5, together with an ordinary input arc Pre(t1,p1) = 1, an ordinary output arc Post(p1,t2) = 1 and a weighted enabling arc En(p1,t3) = 4. Initially it is assumed that p1 is void of tokens. As long as the number of tokens in p1 is fewer than 5, every firing of t1 deposits a token in p1. Provided that there is a token(s) in p1, every firing of t2 removes a token from p1. As long as the number of tokens in p1 is larger than or equal to 4, t3 is allowed to fire; otherwise it is not allowed to fire. The firing of t3 does not change the number of tokens present in p1. To represent the respective events, t1, t2 and t3 are labelled with ‘a’, ‘b’ and ‘c’ respectively. Since p1 is bounded in accordance with CAP(*p*) ≤ 5, it is modelled by a buffer with capacity 6 as shown in Figure 16. State s0 is the initial and marker state and represents the absence of tokens in p1. States s1, s2, s3, s4 and s5 represent the presence of 1, 2, 3, 4 and 5 tokens in p1 respectively. In the automaton model, the occurrence of event ‘a’ represents both the firing of t1 and the depositing of a token in p1, while the occurrence of event ‘b’ represents both the firing of t2 and the removing of a token from p1. Self-loops [s4,c,s4] and [s5,c,s5] are added to states s4 and s5 respectively to represent that the occurrence of event ‘c’ is allowed in these states. On the other hand, as there are no such self-loops assigned to states s0, s1, s2 and s3, the occurrence of event ‘c’ is not allowed in these states.

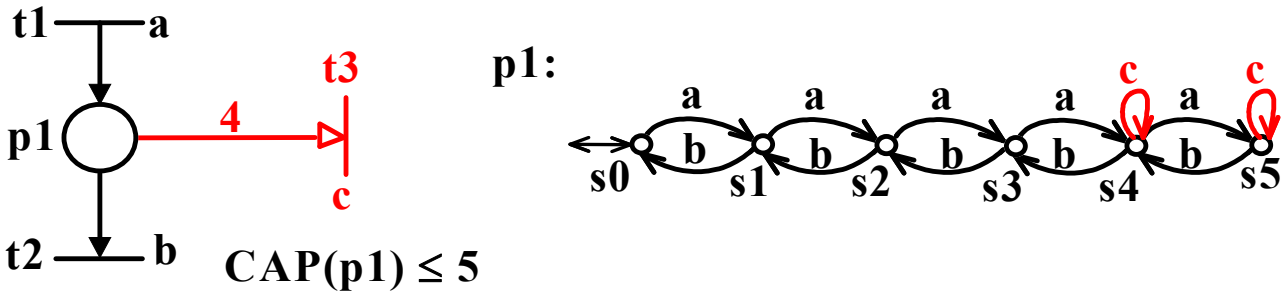


Figure 16. A single place with a token capacity CAP(*p1*) ≤ 5, together with a weighted enabling arc En(p1,t3) = 4 and its automaton representation

5. Analysis of PN-Based Controllers by Using TCT

The proposed analysis approach based on TCT software can be summarised as follows:

1. Assume that an uncontrolled PN model (UPNM) of a DES with initial marking, feedback control elements (a set of control places) and also the related set of forbidden-state specifications are given.
2. Check that UPNM is bounded and determine explicit bounds on the place markings.
3. Convert the UPNM and the specifications into equivalent buffer models. Buffer models of UPNM and specifications are called PLANT and SPEC respectively.
4. Apply SCT to obtain a Ramadge-Wonham (RW) supervisor (RWSUPER) for a given problem by using PLANT and SPEC.
5. Convert the given feedback control elements (a set of control places) into equivalent buffer models called C1, C2, C3, ….
6. Obtain a controlled model called PNSUPER by using PLANT and buffer models of the given feedback control elements C1, C2, C3, ….
7. Compare RWSUPER and PNSUPER to see if they are identical.

In this paper, PN-based controllers enforcing forbidden-state specifications are analysed to see whether they are correct. In addition, liveness enforcing feedback control elements can also be analysed. In such problems, the specification is non-blockingness with respect to the initial state. In the literature, control places (or monitors) are very common feedback control elements with ordinary (or weighted) input and output arcs. However, in some problems it is also possible to carry out the same control action with some other type of control elements containing inhibitor arcs. The computations for the SCT can be performed by standard SCT software; in this paper the package TCT [11] is used. A quick review on some of the TCT commands used in this paper is given in the Appendix.

The boundedness computation in step 2 can be carried out with available PN tools [30] or, in simple cases, by inspection or application by hand of integer programming [29]. For step 3, to carry out the conversions, the mappings proposed in Section 4 can be used. In the conversion of UPNM into PLANT the following steps are followed. First, each place of UPNM, together with its input and output transitions (with input, output and possibly inhibitor or enabling arcs), is converted into an automaton (a buffer) model. Next, all automata models for the places are defined by using the Create() command within the TCT. Then the synchronous product command, namely Sync(), of TCT is used to obtain the PLANT automata model of UPNM. It can be shown that the PLANT model is actually isomorphic to the reachability graph of UPNM. If the problem considered is that of a forbidden-state problem, then in the conversion of specifications into SPEC, the following 3 steps are followed. First, if the specification requires more than one automaton model, namely SPEC1, SPEC2, … $SPEC_n$, then they are all constructed and defined by using the Create() command within the TCT. Second, self-loops labelled with events that are possible in PLANT but not constrained by automaton model of each specification must be adjoined to one another. To accomplish this task, Selfloop() command is used. Finally, Meet() command of TCT is used to obtain the SPEC automaton model of the specifications. If the problem considered is that of a liveness enforcing problem, then the specification is non-blockingness with respect to the initial state. In this case, the non-blockingness specification is obtained by using Allevents() command: ALL = Allevents(PLANT), which results in a 1-state DES called ALL with all events of PLANT as self-loops.

If the problem considered is that of a forbidden-state problem, then step 4 is performed by supcon() command of TCT as follows: RWSUPER = Supcon(PLANT,SPEC). If the problem considered is that of a liveness enforcing problem, then step 4 is performed as follows: RWSUPER = Supcon(PLANT,ALL). Since RWSUPER is the result of a Supcon computation, it is automatically proper, i.e. trim, controllable and non-blocking with respect to PLANT. That it is maximally permissive is also a consequence of the semantics of Supcon, but maximal permissiveness is not included in the definition of "proper". This is a consequence of the semantics of Supcon, as stated explicitly by Wonham [29]. The proper RWSUPER obtained will be used as a proper supervisor reference to test the correctness of the PN-based feedback control elements.

In step 5, to carry out the conversions, the mappings proposed in Section 4 can be used. In the conversion of given feedback control elements (a set of control places) into equivalent buffer models, the following steps are followed. First, each feedback control element (a control place), together with its input and output transitions (with input, output and possibly inhibitor or enabling arcs), is converted into an automaton (buffer) model. Next, all automata models, called C1, C2, C3, …, for the given feedback control elements are defined by using the Create() command within the TCT.

In step 6, a controlled model, namely PNSUPER, is obtained by using PLANT and C1, C2, C3, …. To carry out this task, the synchronous product command, namely Sync(), is used. As with RWSUPER, PNSUPER obtained also represents the controlled behaviour of the PLANT, i.e. the subset of uncontrolled behaviour that this time survives under PN-based supervision.

In the final step, RWSUPER and PNSUPER are compared to see if they are identical (up to possible recoding of state sets). To carry out this task, the test command Isomorph() is used as follows: Isomorph(PNSUPER,RWSUPER). If RWSUPER and PNSUPER are identical (or not identical), the outcome of this test will be 'true' (respectively 'false'). If the test result is 'true', then this proves the correctness of the PN-based feedback control elements. In that case it can be concluded that PNSUPER obtained by using PN-based feedback control elements is proper, i.e. maximally permissive and non-blocking. Note that when considering liveness enforcing problem, i.e. non-blockingness with respect to the initial state, if all transitions are controllable, then the trimmed plant, obtained by Trim() command as follows: TPLANT = Trim(PLANT), will also be isomorphic to PNSUPER (or RWSUPER). To check this property, the following test is carried out by using Isomorph() command: Isomorph(TPLANT,PNSUPER). However, when considering the liveness enforcing problem, if there are both controllable and uncontrollable transitions within the considered UPNM, then trimmed plant TPLANT = Trim(PLANT) may not be isomorphic to the PNSUPER (or RWSUPER). In that case, the RWSUPER is the supremal controllable sublanguage defining the maximally permissive and non-blocking behaviour. Isomorph can certainly be used to check the equality of closed and respectively marked behaviours in each of two DES, e.g. G1 and G2. If Isomorph(G1,G2) = true, then indeed one has that L(G1) = L(G2) and Lm(G1) = Lm(G2). If, however, Isomorph(G1,G2) = false, it may still be the case that the language equalities are true. In the case of a 'false' outcome one must check, in addition, Isomorph(MG1,MG2) = true/false, where MGi = Minstate(Gi). If this second Isomorph returns 'true', then (even though the first Isomorph returns 'false') the language equalities do actually hold; if 'false', then one or both of the language equalities fails. This point has been made explicitly [29].

Up to now the analysis of optimal PN-based supervisors is explained. However, in the PN literature it is possible to come across suboptimal solutions to a given control problem. In order to analyse the correctness of a given suboptimal PN-based solution, RW-type supervisor RWSUPER can still be used as a reference for optimal controlled behaviour. Here, how this analysis is carried out is explained. The analysis steps explained above are still followed up to the last step (step 7). It is obvious that in this case the PN-based controlled model will be a sublanguage of the one generated by RWSUPER. Firstly, the suboptimal behaviour generated by PN-based controlled model PNSUPER is checked to see that it is not conflicting with the RWSUPER as follows: Nonconflict(PNSUPER,RWSUPER). This test must result in "true". Secondly, the reachable Cartesian product of PNSUPER and RWSUPER is computed: MRW_PN = Meet(PNSUPER,RWSUPER). The resulting DES MRW_PN must be equal to that of PNSUPER because in this case any suboptimal solution such as PNSUPER is expected to be sublanguage of the optimal solution RWSUPER. In short, the following result must be obtained: true = Isomorph(MRW_PN,PNSUPER;identity).

6. Application Examples

In this section three examples are considered to show the applicability of the proposed analysis method. The detailed computation steps are provided in the TCT-generated log files for each example.

6.1. Example 1

The first example consists of a simple flexible manufacturing system (FMS) with deadlock (liveness enforcing) problem. Two different controllers solving the deadlock problem for this FMS are verified to be correct and "control equivalent." Note that the PN considered in this example is in the class of SOPN unless otherwise stated. Consider the FMS shown in Figure 17 with two machines M1 and M2, each of which can process only one part at a time, and one robot, which can hold one part at a time. The parts enter the FMS through input/output buffers I/O1 and I/O2. Two parts are considered: P1 and P2. Initially it is assumed that there are no parts in the system. The production sequences are:

P1: M1 → Robot → M2
P2: M2 → Robot → M1

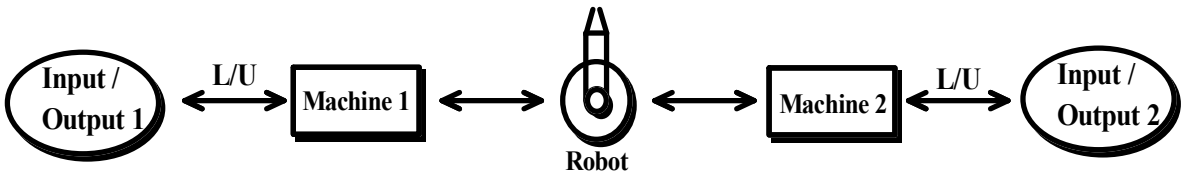


Figure 17. An example of FMS

Figure 18(a) shows the PN model (PNM) of the FMS taken from Uzam [20] for these production sequences. In this model there are eleven places, P = {p1, p2, …, p11}, and eight transitions, T = {t1, t2, …, t8}. All transitions are assumed to be controllable. Places p2, p5 and p8 represent the operation of M1, Robot and M2 respectively for the first production sequence. The number of tokens in p1, i.e. M(p1) = 3, represents the number of concurrent activities that can take place for P1. Similarly, places p10, p7 and p4 represent the operation of M2, Robot and M1 respectively for the second production sequence. The number of tokens in p11, i.e. M(p11) = 3, represents the number of concurrent activities that can take place for P2. Places p3, p6 and p9 denote the shared resources M1, Robot and M2 respectively. In order to have the correct system behaviour, it is desirable that each production sequence can finish. It can be shown that this system suffers from deadlock problem and for this system the specification is to obtain the live system behaviour (non-blockingness in SCT terminology). It can be shown that places p1 and p11 are redundant and therefore they can be removed from the model without any problem. When they are removed from the model, a SOPN model is obtained for the FMS as shown in Figure 18(b).

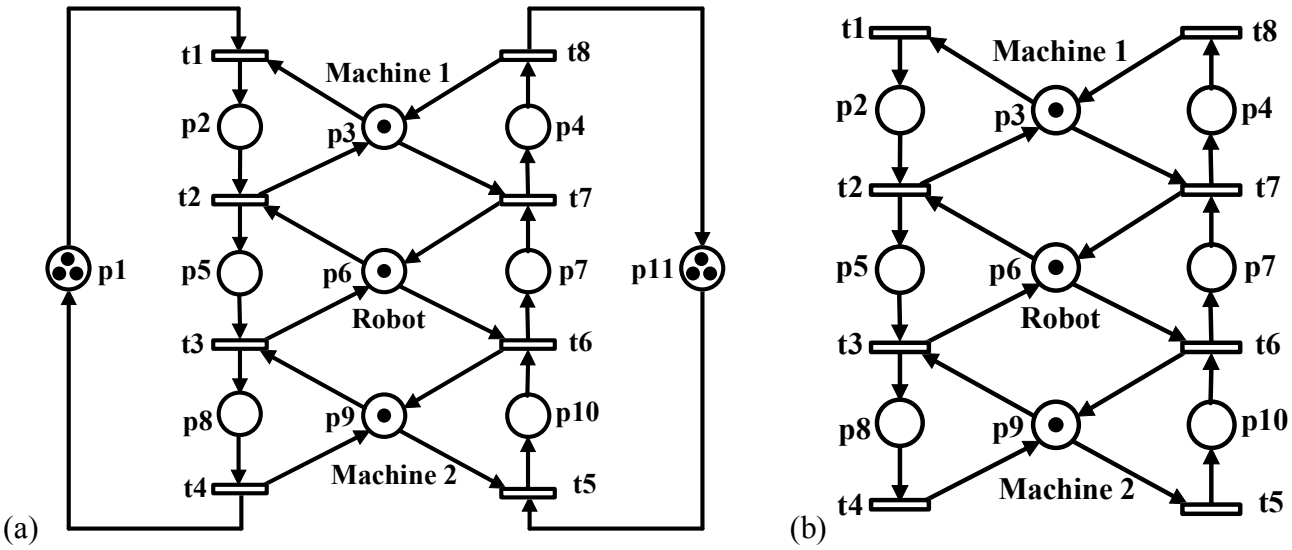


Figure 18. (a) PN model of FMS for two production sequences; (b) SOPN model for the FMS (UPNM)

It can be verified that the PN model of the FMS suffers from deadlock (the same applies to the SOPN). In the reachability graph of this model, there are 20 states, 15 of which are good states representing the optimal live behaviour and 5 of which are bad ones including 2 deadlock states. The maximally permissive controller obtained previously [20] is depicted in Figure 19(a). When the controller is augmented with the uncontrolled model (SOPN), the controlled model shown in Figure 19(b) is obtained.

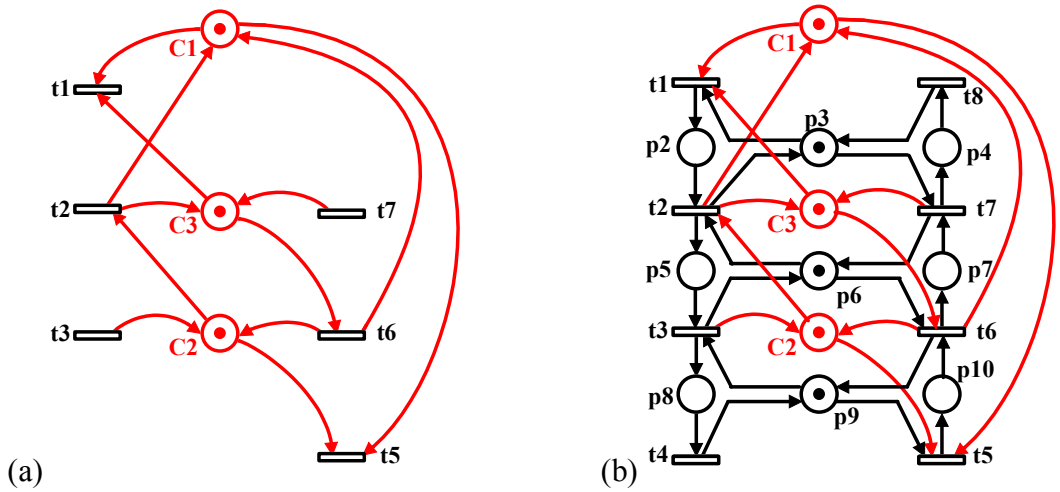


Figure 19. (a) Controller previously obtained [20] for Example 1; (b) Controlled model for the FMS

For this example the proposed analysis approach is followed as shown below.

Step 1. The UPNM is the one given in Figure 18(b); feedback control elements are three control places as depicted in Figure 19(a), and the specification for the UPNM is non-blockingness with respect to the initial state, i.e. liveness enforcing.

Step 2. It can be verified that UPNM is bounded with the place marking bounds p2:1, p3:1, p4:1, p5:1, p6:1, p7:1, p8:1, p9:1 and p10:1. It can be seen that UPNM is actually a SOPN.

Step 3. Each place of UPNM is converted into an equivalent buffer model as shown in Figure 20. The following event coding is used in the conversion process according to the TCT requirement that odd and even integers be used for controllable and uncontrollable events respectively.

Event coding for Example 1:

UPNM:	t1	t2	t3	t4	t5	t6	t7	t8
TCT:	1	3	5	7	15	13	11	9

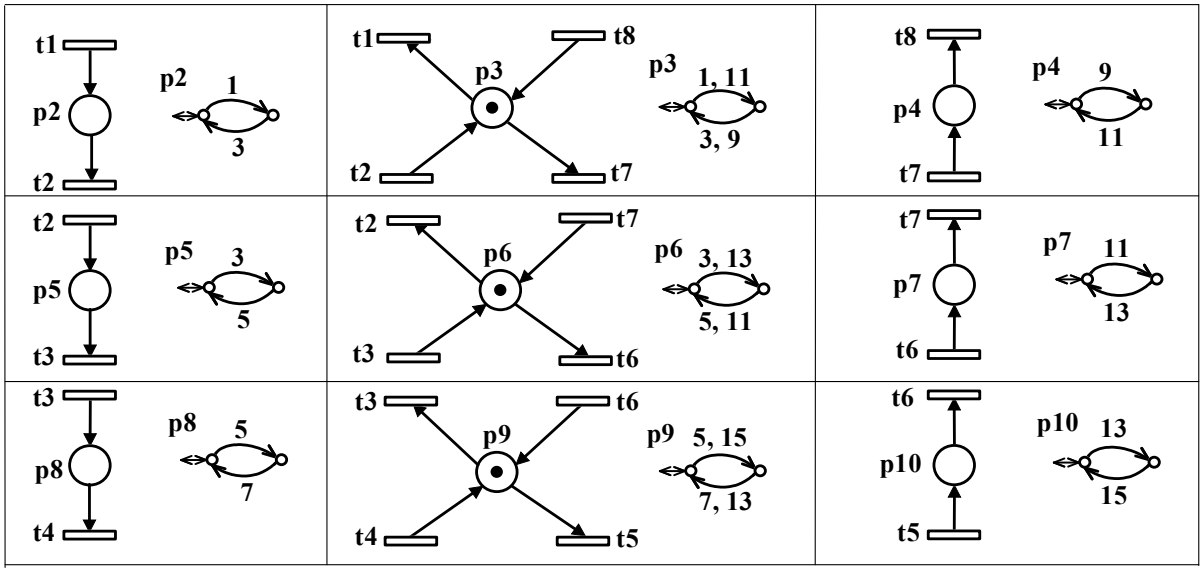


Figure 20. Mappings from UPNM to automata in Example 1

The following steps are carried out using TCT. In the following, PLANT defines automaton representation of UPNM.

First each place is created as an automaton as follows:
p2 = Create(p2,[mark 0],[tran [0,1,1],[1,3,0]]) (2,2)
p3 = Create(p3,[mark 0],[tran [0,1,1],[0,11,1],[1,3,0],[1,9,0]]) (2,4)
p4 = Create(p4,[mark 0],[tran [0,11,1],[1,9,0]]) (2,2)
p5 = Create(p5,[mark 0],[tran [0,3,1],[1,5,0]]) (2,2)
p6 = Create(p6,[mark 0],[tran [0,3,1],[0,13,1],[1,5,0],[1,11,0]]) (2,4)
p7 = Create(p7,[mark 0],[tran [0,13,1],[1,11,0]]) (2,2)
p8 = Create(p8,[mark 0],[tran [0,5,1],[1,7,0]]) (2,2)
p9 = Create(p9,[mark 0],[tran [0,5,1],[0,15,1],[1,7,0],[1,13,0]]) (2,4)
p10 = Create(p10,[mark 0],[tran [0,15,1],[1,13,0]]) (2,2)

Then the PLANT is the synchronous product of all places, i.e. PLANT = p2||p3||p4||p5||p6||p7||p8||p9||p10 and is obtained as follows (and depicted in Figure 21):
PLANT = Sync(P2,P3) (4,6) Blocked_events = None
PLANT = Sync(PLANT,P4) (3,4) Blocked_events = None
PLANT = Sync(PLANT,P5) (6,10) Blocked_events = None
PLANT = Sync(PLANT,P6) (12,20) Blocked_events = None
PLANT = Sync(PLANT,P7) (8,12) Blocked_events = None
PLANT = Sync(PLANT,P8) (16,30) Blocked_events = None
PLANT = Sync(PLANT,P9) (32,60) Blocked_events = None
PLANT = Sync(PLANT,P10) (20,34) Blocked_events = None

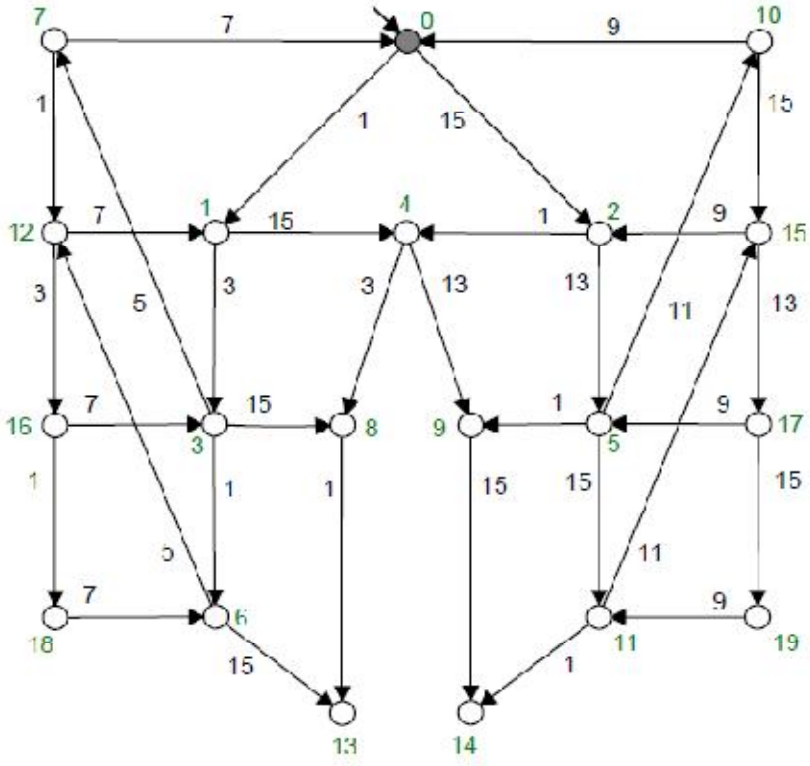


Figure 21. The automaton PLANT = p2||p3||p4||p5||p6||p7||p8||p9||p10 representing the uncontrolled system behaviour in Example 1

The problem considered is that of liveness enforcing. Therefore, the specification is non-blockingness with respect to the initial state and it is obtained by using Allevents() command as follows: ALL = Allevents(PLANT) (1,8). In addition, the following test shows that PLANT will block in the absence of control: false = Nonconflict(PLANT,ALL).

Step 4. A RW supervisor RWSUPER is obtained as follows: RWSUPER = Supcon(PLANT,ALL) (15,24). The RW-type supervisor RWSUPER obtained is depicted in Figure 22.

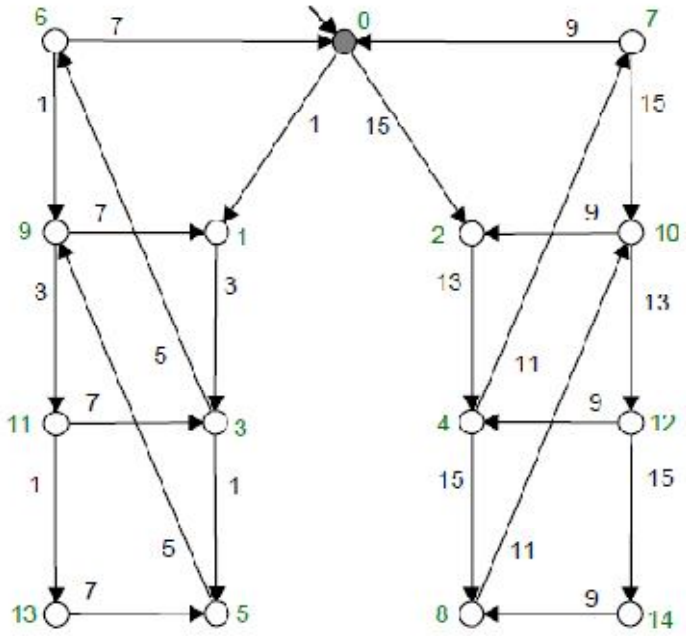


Figure 22. The supervisor RWSUPER computed for Example 1

Step 5. The given feedback control elements, i.e. a set of control places, are converted into equivalent buffer models as shown in Figure 23. Control places are created within TCT as follows:
C1 = Create(C1,[mark 0],[tran [0,1,1],[0,15,1],[1,3,0],[1,13,0]]) (2,4)
C2 = Create(C2,[mark 0],[tran [0,3,1],[0,15,1],[1,5,0],[1,13,0]]) (2,4)
C3 = Create(C3,[mark 0],[tran [0,1,1],[0,13,1],[1,3,0],[1,11,0]]) (2,4)

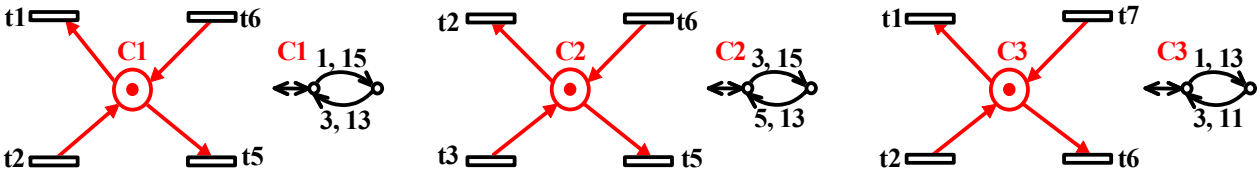


Figure 23. Mappings from given feedback control elements to automata in Example 1

Step 6. The controlled model PNSUPER (PNSUPER = PLANT||C1||C2||C3) is obtained by using PLANT and buffer models of the given feedback control elements C1, C2, and C3 as follows:
PNSUPER = Sync(PLANT,C1) (17,26) Blocked_events = None
PNSUPER = Sync(PNSUPER,C2) (16,25) Blocked_events = None
PNSUPER = Sync(PNSUPER,C3) (15,24) Blocked_events = None

Step 7. The following test verifies that the controlled model PNSUPER obtained is isomorphic (identical) to RWSUPER: true = Isomorph(PNSUPER,RWSUPER; identity). This means that the controller obtained in the PN domain is also a proper supervisor, i.e. it is maximally permissive and non-blocking.

The following shows that the optimal controlled (maximally permissive and non-blocking) behaviour PNSUPER is simply the language generated by the trimmed plant as expected. This is because all transitions are controllable.
TPLANT = Trim(PLANT) (15,24)
true = Isomorph(TPLANT,PNSUPER; identity)

Here, alternative feedback control elements are established and tested for Example 1. When the control places C1, C2 and C3 are examined in detail, it can be seen that these three control places implement a control mechanism in which, when there is a token in p2 or p5, t5 is not allowed to fire. This means that t5 is stopped from firing when M(p2) = 1 or M(p5) = 1. Likewise, when there is a token in p7 or p10, t1 is not allowed to fire, i.e. t1 is stopped from firing. This means that t1 is stopped from firing when M(p7) = 1 or M(p10) = 1. Actually, this control mechanism can be implemented by using the feedback control elements shown in Figure 24(a). When these feedback control elements are augmented with the uncontrolled model (SOPN), the controlled model shown in Figure 24(b) is obtained. Now the correctness of this new controller consisting of the alternative feedback control elements shown in Figure 24(a) is verified.

From this point onward the proposed analysis approach is followed for the new controller from step 5 as shown below.

Step 5. The feedback control elements shown in Figure 24(a) are converted into equivalent buffer models as shown in Figure 25. Control places are created within TCT as follows:
SP1 = Create(SP1,[mark 0],[tran [0,1,1],[0,15,0],[1,3,0]]) (2,3)
SP2 = Create(SP2,[mark 0],[tran [0,3,1],[0,15,0],[1,5,0]]) (2,3)
SP3 = Create(SP3,[mark 0],[tran [0,1,0],[0,13,1],[1,11,0]]) (2,3)
SP4 = Create(SP4,[mark 0],[tran [0,1,0],[0,15,1],[1,13,0]]) (2,3)

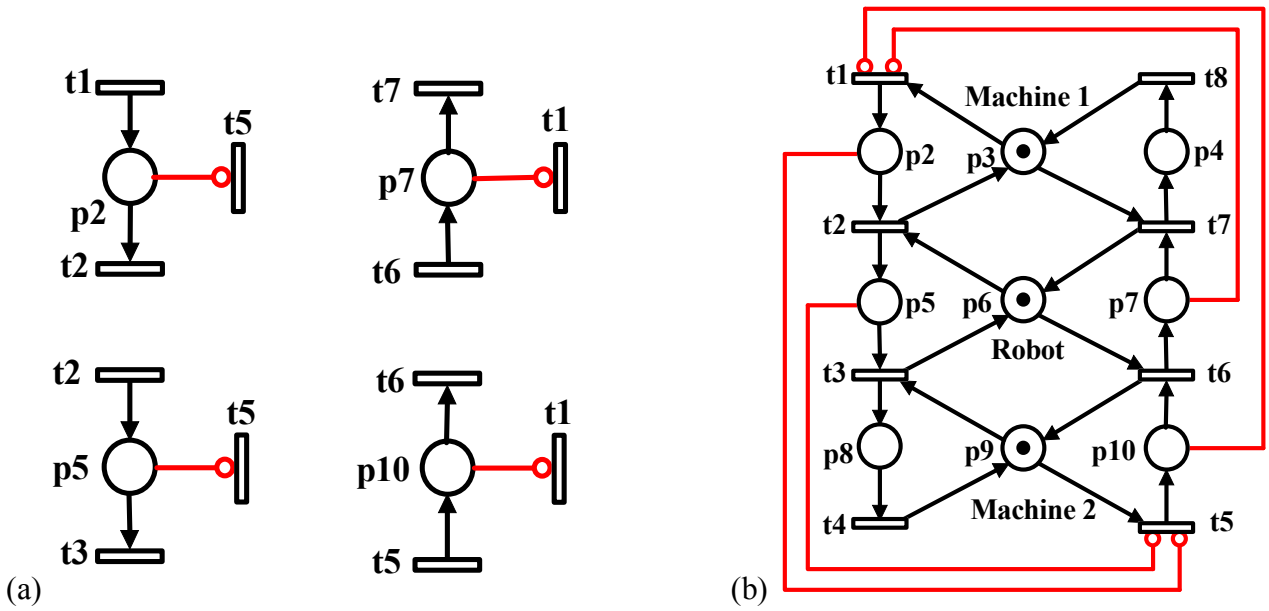


Figure 24. (a) Alternative feedback control elements constructed for Example 1; (b) Controlled model obtained by means of alternative feedback control elements

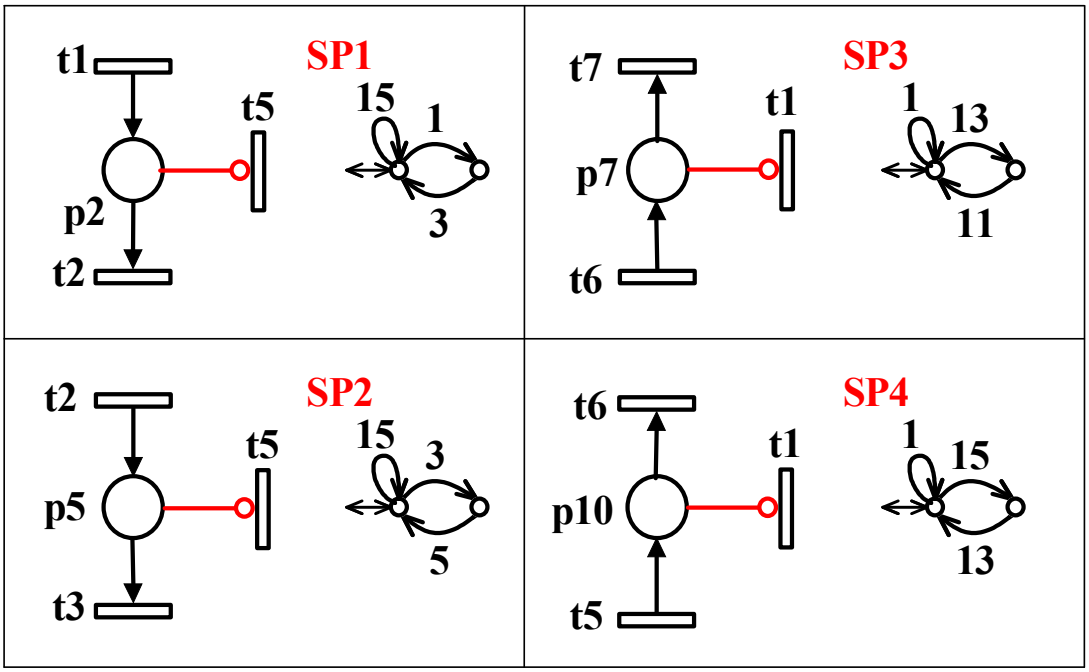


Figure 25. Mappings from PN to automata for the new controller

Step 6. The controlled model CM1 (CM1 = PLANT||SP1||SP2||SP3||SP4) is obtained by using PLANT and buffer models of the new feedback control elements SP1, SP2, SP3 and SP3 as follows:
CM1 = Sync(PLANT,SP1) (20,31) Blocked_events = None
CM1 = Sync(CM1,SP2) (20,30) Blocked_events = None
CM1 = Sync(CM1,SP3) (19,28) Blocked_events = None
CM1 = Sync(CM1,SP4) (15,24) Blocked_events = None

Step 7. Finally, the test below verifies that the new controlled model CM1 is identical in terms of control action to that of RWSUPER, which also means that it is identical to PNSUPER. In other words, both controlled models shown in Figure 19(b) and Figure 24(b) generate the same language and

therefore they enforce the same specification. In this respect, the feedback control elements shown in Figure 23 and the ones shown in Figure 24(a) are said to be "control equivalent."

true = Isomorph(RWSUPER,CM1;identity)

6.2. Example 2

As a second example, a generalised mutual exclusion constraint (GMEC) to be enforced on a PNM is considered. This example is taken from Basile et al. [31] and it consists of the UPNM shown in Figure 26. In the UPNM there are seven places, P = {p1, p2, ..., p7}, and six transitions, T = {t1, t2, ... , t6}. The initial marking is $M_0 = (0, 0, 0, 0, 0, 2, 2)^T$. It can be verified that the UPNM shown in Figure 26 is bounded, live, reversible and conservative. In the UPNM, transitions t1, t5 and t6 are controllable while transitions t2, t3 and t4 are uncontrollable. The specification, namely GMEC, to be enforced on this UPNM is $M(p1) \leq 1$, i.e. the number of tokens in place p1 must be fewer than or equal to 1. There are three solutions to be analysed for this example. The first two of them are two different optimal solutions and the third one is a suboptimal solution. These three solutions are considered respectively in the next three subsections.

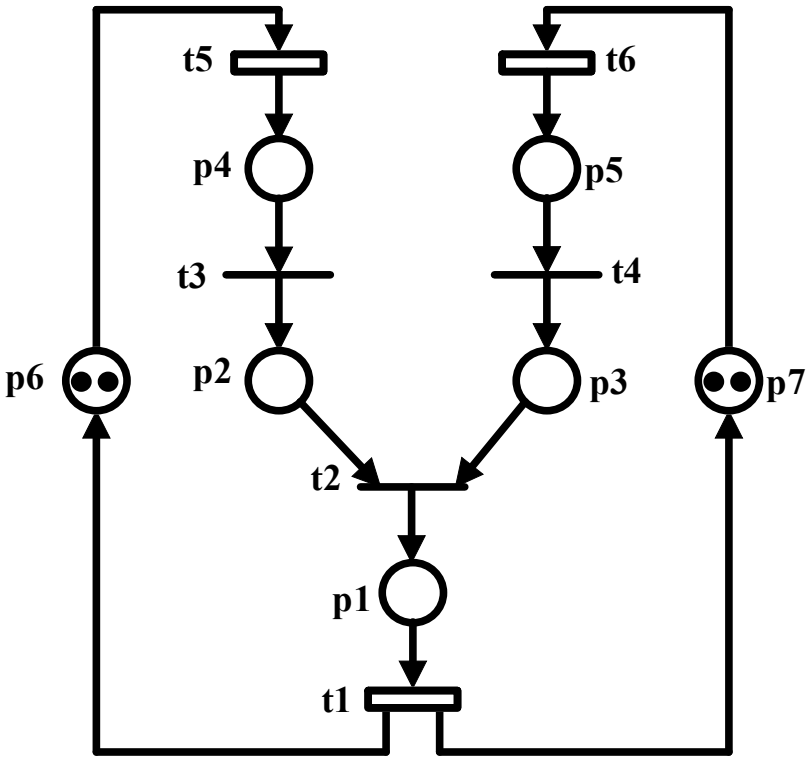


Figure 26. A PN with controllable (t1, t5, t6) and uncontrollable (t2, t3, t4) transitions [31] (without dashed arcs and places)

6.2.1. Example 2 – optimal solution 1

In order to optimally enforce GMEC $M(p1) \leq 1$ on the UPNM shown in Figure 26, the control places C1, C2, C3 and C4 in Figure 27 are provided [10]. It was shown [10] that the controlled model in Figure 28 can produce the supremal controllable sublanguage which has 32 states with 62 transitions. In short, the controlled model shown in Figure 28 is maximally permissive and non-blocking.

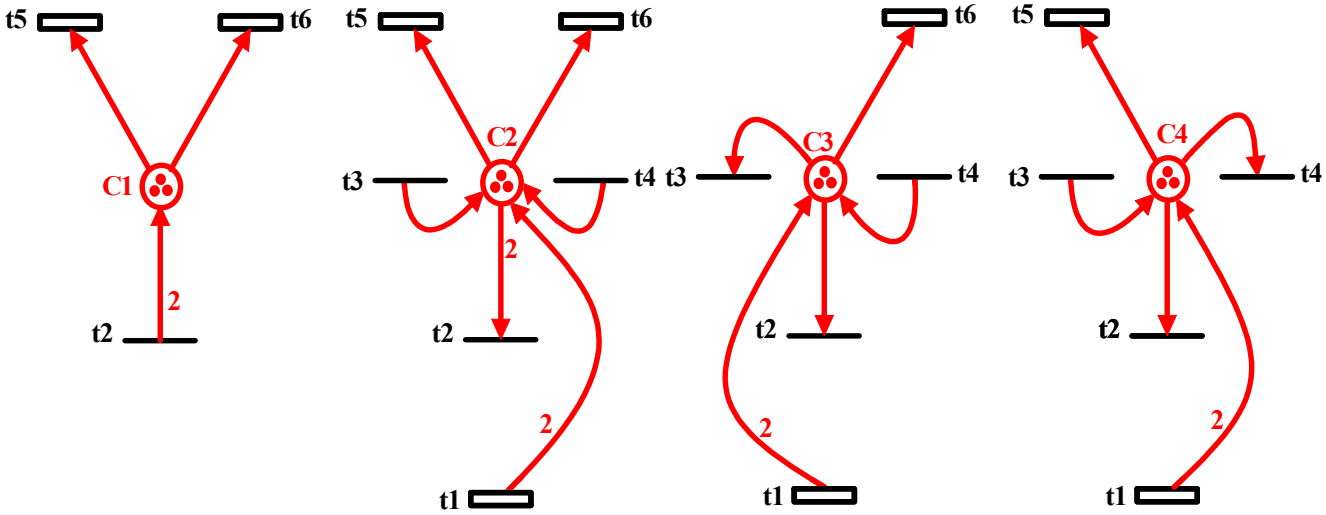


Figure 27. Four control places [10] enforcing the constraint $M(p1) \leq 1$ optimally on the PN shown in Figure 26

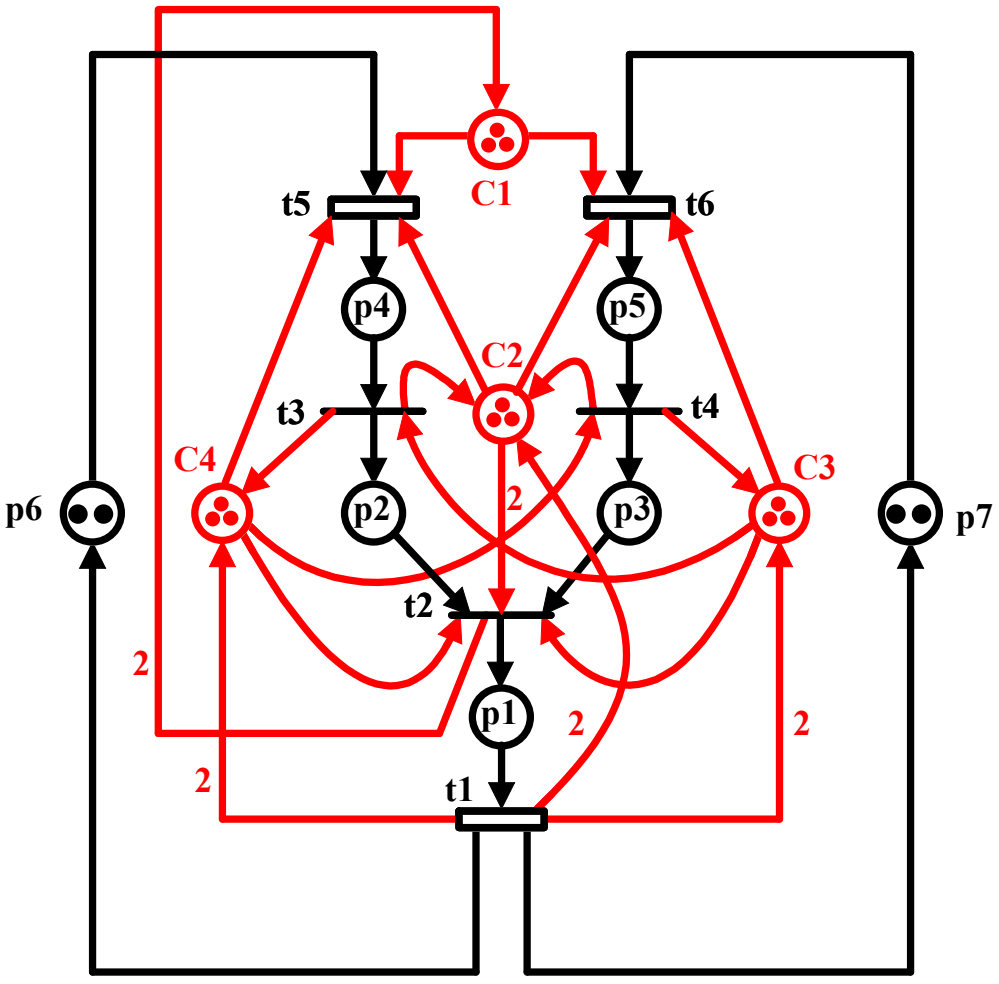


Figure 28. The controlled PN model obtained by including four control places within the PN shown in Figure 26

For this example, to verify the correctness of control places C1, C2, C3 and C4, the proposed analysis approach is followed as shown below.

Step 1. The UPNM is the one given in Figure 26; the feedback control elements are the control places as depicted in Figure 27, and the specification for the UPNM is M(p1) ≤ 1, i.e. the number of tokens in place p1 must be less than or equal to 1.

Step 2. It can be verified that UPNM is bounded with the place marking bounds p1:2, p2:2, p3:2, p4:2, p5:2, p6:2 and p7:2.

Step 3. Each place of UPNM is converted into an equivalent buffer model as shown in Figure 29. The following event coding is used in the conversion process according to the TCT requirement that odd and even integers be used for controllable and uncontrollable events respectively.

Event coding for Example 2:

UPNM:	t1	t2	t3	t4	t5	t6
TCT:	1	2	30	4	5	61

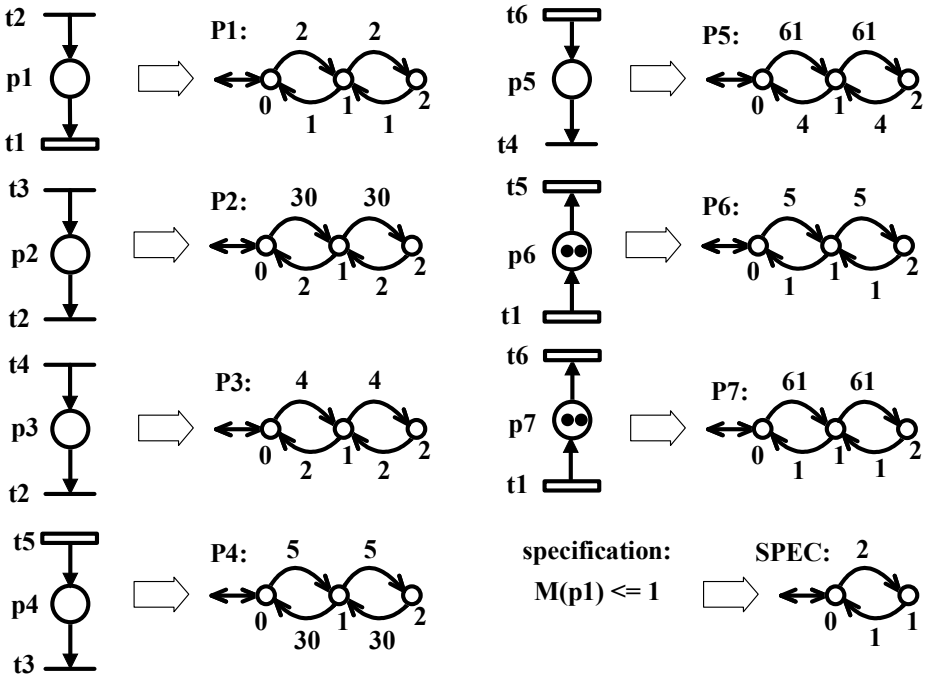


Figure 29. Mappings from UPNM to automata and specification model SPEC in Example 2

The following steps are carried out using TCT. In the following, PLANT defines automaton representation of UPNM.

First, each place is created as an automaton as follows:

P1 = Create(P1,[mark 0],[tran [0,2,1],[1,1,0],[1,2,2],[2,1,1]]) (3,4)
P2 = Create(P2,[mark 0],[tran [0,30,1],[1,2,0],[1,30,2],[2,2,1]]) (3,4)
P3 = Create(P3,[mark 0],[tran [0,4,1],[1,2,0],[1,4,2],[2,2,1]]) (3,4)
P4 = Create(P4,[mark 0],[tran [0,5,1],[1,5,2],[1,30,0],[2,30,1]]) (3,4)
P5 = Create(P5,[mark 0],[tran [0,61,1],[1,4,0],[1,61,2],[2,4,1]]) (3,4)
P6 = Create(P6,[mark 0],[tran [0,5,1],[1,1,0],[1,5,2],[2,1,1]]) (3,4)
P7 = Create(P7,[mark 0],[tran [0,61,1],[1,1,0],[1,61,2],[2,1,1]]) (3,4)

Then the PLANT is the synchronous product of all places, i.e. PLANT = P1||P2||P3||P4||P5||P6||P7, and is obtained as follows:

PLANT = Sync(P1,P2) (9,16) Blocked_events = None
PLANT = Sync(PLANT,P3) (27,62) Blocked_events = None
PLANT = Sync(PLANT,P4) (81,222) Blocked_events = None
PLANT = Sync(PLANT,P5) (243,774) Blocked_events = None
PLANT = Sync(PLANT,P6) (90,232) Blocked_events = None
PLANT = Sync(PLANT,P7) (46,104) Blocked_events = None

Recall that the specification is M(p1) ≤ 1, i.e. the number of tokens in place p1 must be less than or equal to 1. This specification can be represented as a buffer automaton model as depicted in Figure 29. This specification model simply declares that once event "2" occurs, it cannot reoccur until event "1" takes place. This defines specification M(p1) ≤ 1 as a buffer automaton model, which is defined within TCT as follows:
SPEC = Create(SPEC,[mark 0],[tran [0,2,1],[1,1,0]]) (2,2)

After this operation, self-loops labelled with events that are possible in PLANT but not constrained by the specification must be adjoined to each state. To do this, self-loops with event labels '4', '5', '30' and '61' are adjoined to each state of SPEC as follows:
SPEC = Selfloop(SPEC,[4,5,30,61]) (2,10)
Note that as there is only one specification model, it is not necessary to use Meet() command.

Step 4. A RW supervisor RWSUPER is obtained as follows:
RWSUPER = Supcon(PLANT,SPEC) (32,62)
The RW-type supervisor RWSUPER obtained is depicted in Figure 30 and is given below.

RWSUPER # states: 32 state set: 0 ... 31 initial state: 0
marker states: 0 vocal states: none # transitions: 62
transitions:
[0,5,1][0,61,2][1,5,3][1,30,4][1,61,5][2,4,6][2,5,5][2,61,7][3,30,8][3,61,9][4,5,8][4,61,10]
[5,4,11][5,5,9][5,30,10][5,61,12][6,5, 11][6,61,13][7,4,13][7,5,12][8,30,14][8,61,15]
[9,4,16][9,30,15][10,4,17] [10,5,15][10,61,18][11,5,16][11,30,17][11,61,19][12,4,19]
[12,30,18][13,4,20][13,5,19][14,61,21][15,4,22][15,30,21][16,30,22][17,2,23][17,5,22]
[17,61,24][18,4,24][19,4,25][19,30,24][20,5,25][21,4,26][22,2,27][22,30,26][23,1,0]
[23,5,27][23,61,28][24,2,28][24,4,29][5,30,29][26,2,30][27,1,1][27,30,30][28,1,2][28,4,31]
[29,2,31][30,1,4][31,1,6]

Step 5. The given feedback control elements, i.e. control places C1, C2, C3 and C4, shown in Figure 27, are converted into equivalent buffer models as shown in Figure 31. The control places C1, C2, C3 and C4 are created within TCT as follows:

C1 = Create(C1,[mark 0], [tran [0,5,1], [0,61,1], [1,5,2], [1,61,2], [2,2,0], [2,5,3], [2,61,3], [3,2,1]]) (4,8)
C2 = Create(C2,[mark 0],[tran [0,2,2], [0,5,1], [0,61,1], [1,2,3], [1,4,0], [1,5,2], [1,30,0], [1,61,2], [2,1,0], [2,4,1], [2,5,3], [2,30,1], [2,61,3], [3,1,1], [3,4,2], [3,30,2]]) (4,16)
C3 = Create(C3,[mark 0], [tran [0,2,1], [0,30,1], [0,61,1], [1,2,2], [1,4,0], [1,30,2], [1,61,2], [2,1,0], [2,2,3], [2,4,1], [2,30,3], [2,61,3],[3,1,1],[3, 4,2]]) (4,14)
C4 = Create(C4,[mark 0], [tran [0,2,1], [0,4,1], [0,5,1], [1,2,2], [1,4,2], [1,5,2], [1,30,0], [2,1,0], [2,2,3], [2,4,3], [2,5,3], [2,30,1], [3,1,1], [3,30,2]]) (4,14)

Step 6. The controlled model PNSUPER1 (PNSUPER1 = PLANT||C1||C2||C3||C4) is obtained by using PLANT and buffer models C1, C2, C3 and C4 of given control places as follows:
PNSUPER1 = Sync(PLANT,C1) (37,76) Blocked_events = None
PNSUPER1 = Sync(PNSUPER1,C2) (35,69) Blocked_events = None
PNSUPER1 = Sync(PNSUPER1,C3) (34,66) Blocked_events = None
PNSUPER1 = Sync(PNSUPER1,C4) (32,62) Blocked_events = None

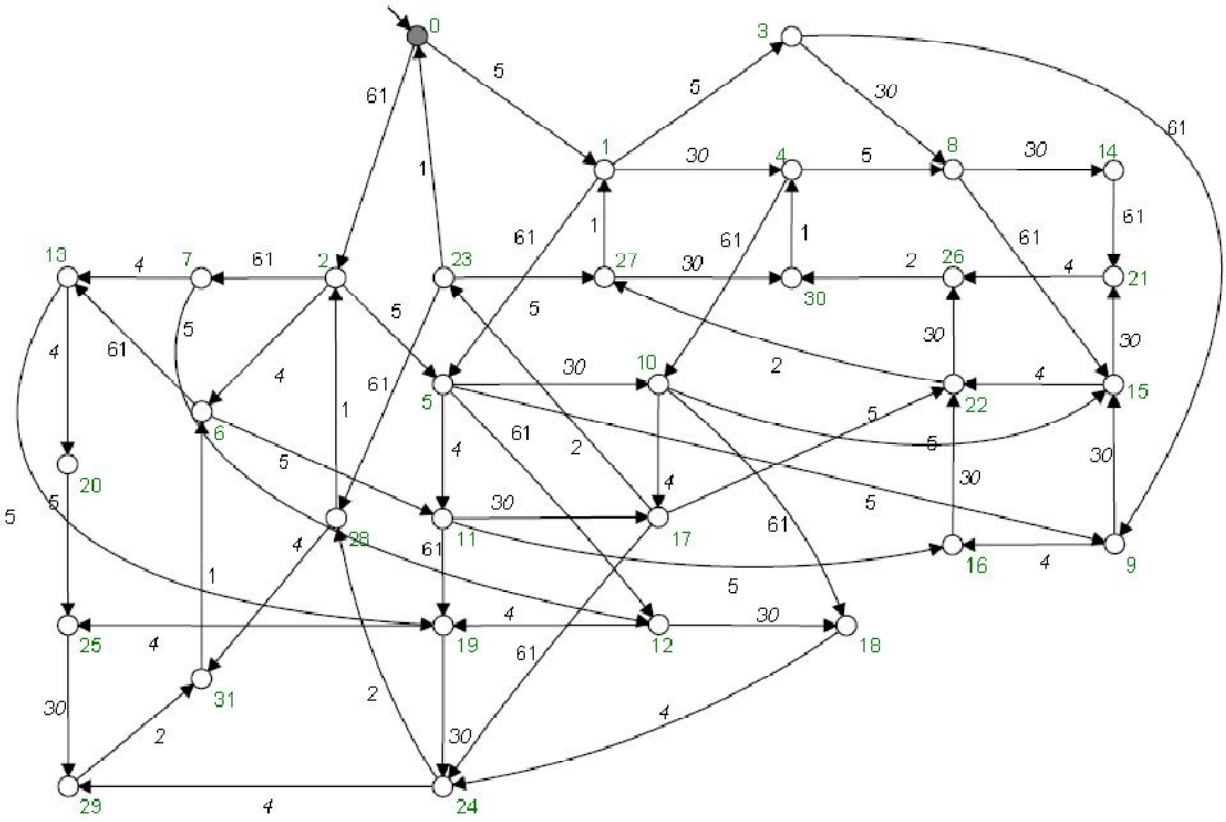


Figure 30. The supervisor RWSUPER computed for Example 2

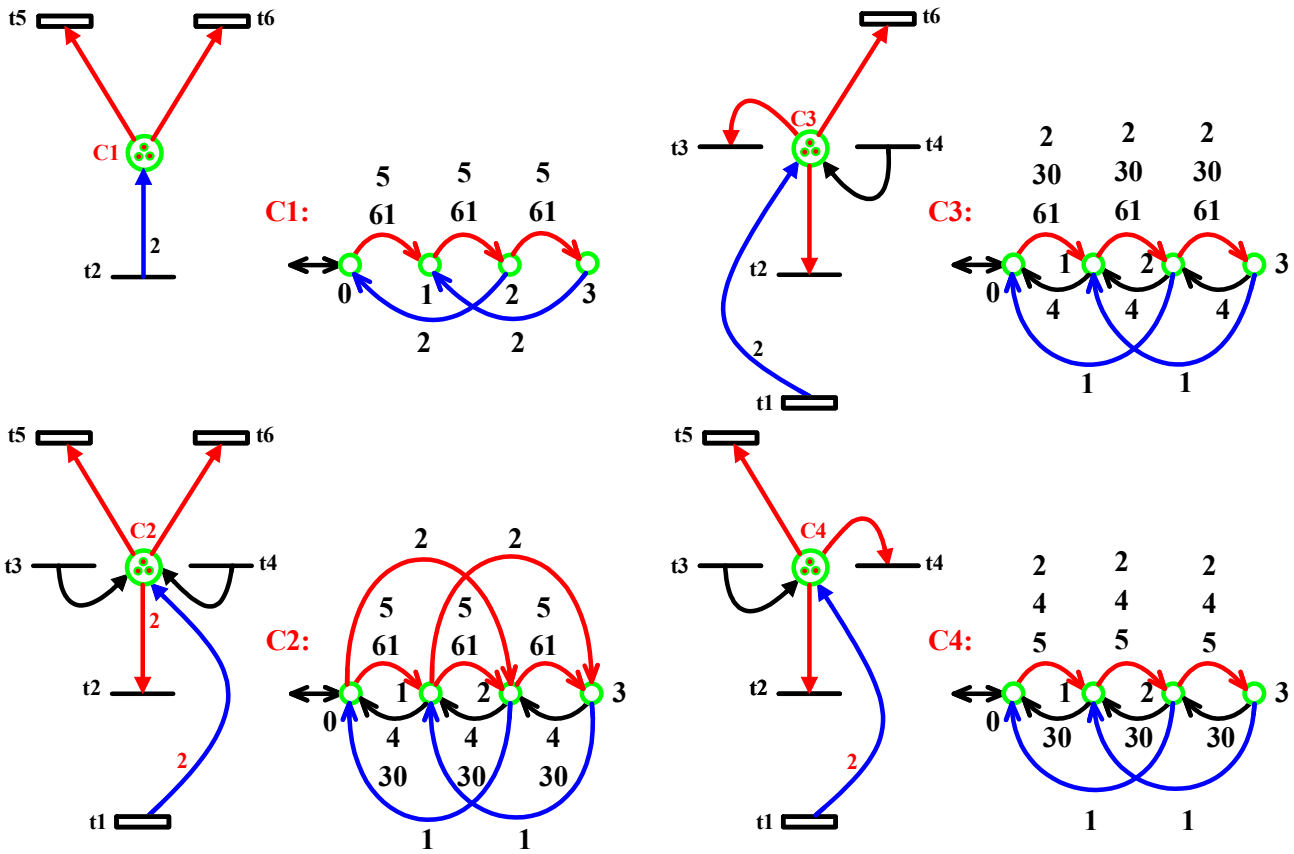


Figure 31. Mappings from control places C1, C2, C3 and C4 to automata in Example 2

Step 7. The test below verifies that the controlled model PNSUPER1 obtained is isomorphic (identical) to RWSUPER. This means that the controller obtained in the PN domain is also a proper supervisor, i.e. it is maximally permissive and non-blocking.

true = Isomorph(PNSUPER1,RWSUPER;identity)

6.2.2. Example 2 – optimal solution 2

As an alternative control mechanism to optimally enforce GMEC M(p1) ≤ 1 on the UPNM of Figure 26, the control place C shown in Figure 32(a) is also computed [10]. It was shown [10] that the controlled model in Figure 32(b) can produce the supremal controllable sublanguage which has 32 states with 62 transitions. In short, the controlled model shown in Figure 32(b) is also maximally permissive and non-blocking.

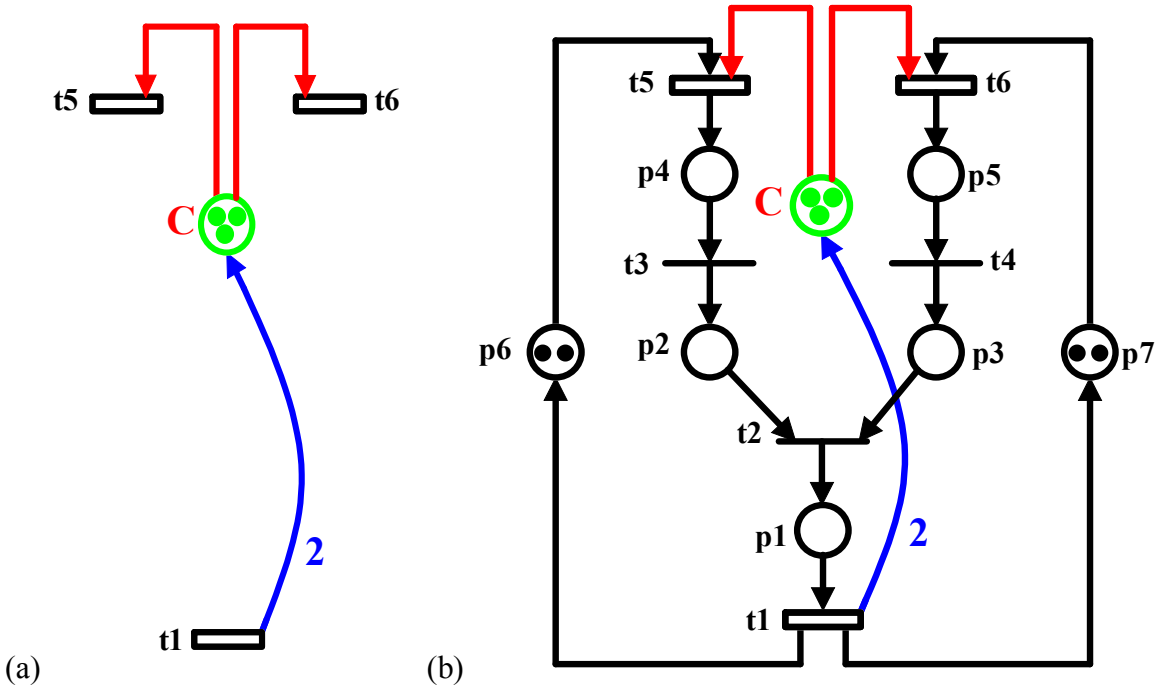


Figure 32. (a) A computed control place C [10]; (b) Controlled model

As the first four analysis steps are the same as the ones provided in the previous section, for this example, to verify the correctness of the control place C, a proposed analysis approach is followed as shown below.

Step 5. The given feedback control element, i.e. control place C shown in Figure 32(a), is converted into an equivalent buffer model as shown in Figure 33. The control place C is created within TCT as follows:

C = Create(C,[mark 0],[tran[0,5,1],[0,61,1],[1,5,2],[1,61,2],[2,1,0],[2,5,3],[2,61,3], [3,1,1]]) (4,8)

Step 6. The controlled model PNSUPER2 (PNSUPER2 = PLANT||C) is obtained by using PLANT and buffer model C of the given control place as follows:

PNSUPER2 = Sync(PLANT,C) (32,62) Blocked_events = None

Step 7. The test below verifies that the controlled model PNSUPER2 obtained is isomorphic (identical) to RWSUPER. This means that the controller obtained in the PN domain is also a proper supervisor, i.e. it is maximally permissive and non-blocking.

true = Isomorph(PNSUPER2,RWSUPER;identity)

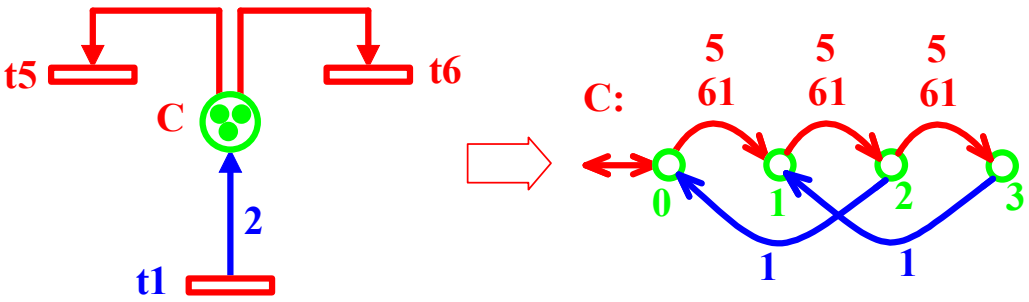


Figure 33. Mapping from control place C to automata in Example 2

6.2.3. Example 2 – suboptimal solution

In this section a suboptimal solution for enforcing GMEC M(p1) ≤ 1 on the UPNM shown in Figure 26 is analysed. In Basile et al. [31], two suboptimal control places, namely pc1 and pc2, as shown in Figure 34(a), are computed. Only one of them is necessary to enforce the GMEC M(p1) ≤ 1 on the UPNM shown in Figure 26. In this section the controlled model obtained by pc1 is analysed as shown in Figure 34(b).

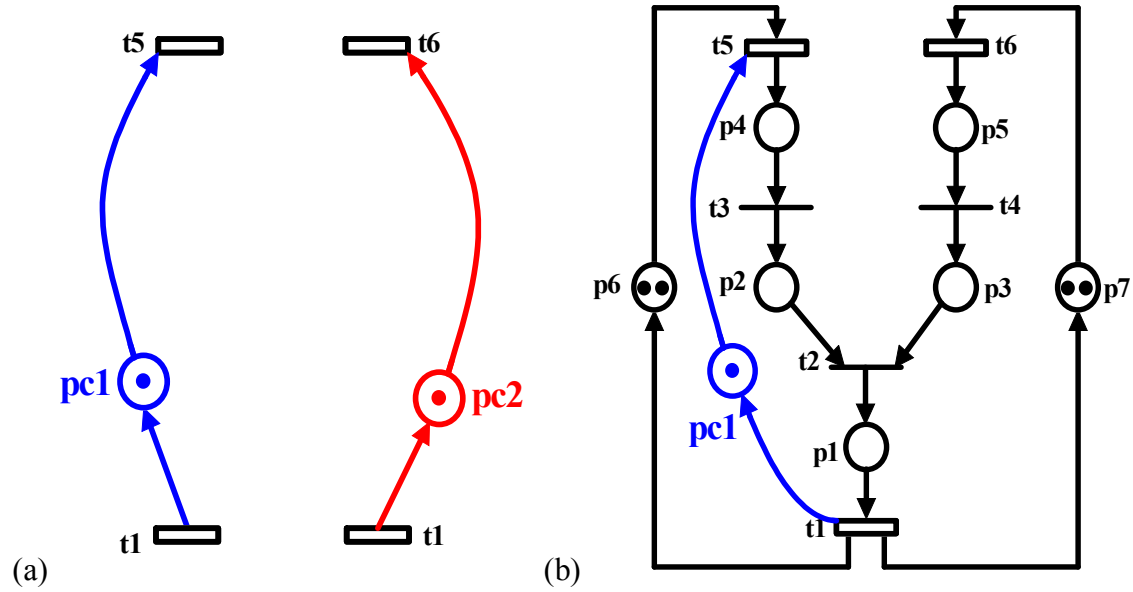


Figure 34. (a) Two computed possible control places [31] for enforcing GMEC M(p1) ≤ 1 on the UPNM shown in Figure 26; (b) Controlled model obtained by pc1

For the suboptimal case in Example 2, the first four analysis steps are the same as the ones provided in the previous section.

Step 5. The given feedback control element pc1 is converted into an equivalent buffer model PC1 as shown in Figure 35. PC1 is created within TCT as follows:
PC1 = Create(PC1,[mark 0],[tran [0,5,1],[1,1,0]]) (2,2)

Step 6. The controlled model SPNSUPER (SPNSUPER = PLANT||PC1) is obtained by using PLANT and buffer model PC1 of the given control place as follows:
SPNSUPER = Sync(PLANT,PC1) (21,38) Blocked_events = None

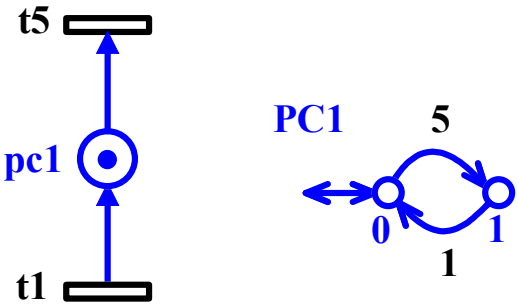


Figure 35. Mapping from pc1 to automaton model PC1 in Example 2

Step 7. The following tests verify that the controlled model SPNSUPER obtained is the sublanguage of the optimal solution RWSUPER. This means that the controller obtained in the PN domain is a suboptimal supervisor for enforcing GMEC $M(p1) \leq 1$ on the UPNM shown in Figure 26.
true = Nonconflict(SPNSUPER,RWSUPER)
MRW_PN = Meet(SPNSUPER,RWSUPER) (21,38)
true = Isomorph(MRW_PN,SPNSUPER;identity)

6.3. Example 3

In this section, an example of FMS prone to deadlock, as shown in Figure 36(a), is considered. The system is composed of two robots, namely R1 and R2, each of which can hold one part at a time, and three machines, namely M1, M2 and M3. M3 can process only one part at a time while M1 and M2 can process two parts at a time. For loading of the system, there are two loading buffers I1 and I2, and for unloading, there are three unloading buffers O1, O2 and O3. The action areas of robot R1 are I1, M1 and M3, and for robot R2, they are M1 and M2. For the sake of simplicity, it is assumed that inputting parts from I2 to M2 and outputting parts from M3 to O1, M2 to O2, and M1 to O3 are the part of machining operation. As shown in Figure 36(b), two part types are considered: P1 and P2. P1 is taken from I1 by R1 and it is either put into M3 or M1. After being processed by M3, P1 is moved to O1 by M3. When put into M1, P1 is processed by M1 and after that it is moved from M1 to M2 by R2. After being processed by M2, P1 is finally moved to O2 by M2. In the production process of P1, R1&M3 or R1&M1 and R2&M2 are used. Similarly, P2 is taken from I2 by M2, and after being processed by M2 it is moved from M2 to M1 by R2. Finally, after being processed by M1, P2 is moved to O3 by M1. In the production process of P2, M2&R2&M1 are used.

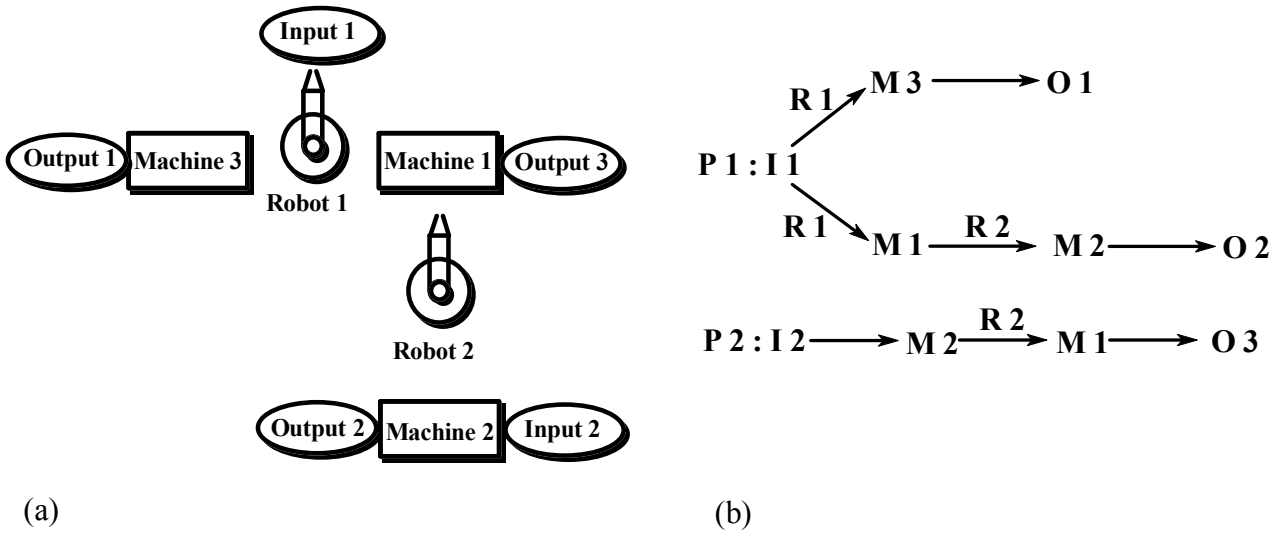


Figure 36. (a) An example of FMS; (b) The production sequences

Figure 37(a) shows the PNM of the system [18]. Initially it is assumed that there are no parts in the system. In the PNM there are fifteen places, P = {p1, p2, …, p10, R1, R2, M1, M2, M3}, and eleven transitions, T = {t0, t1, t2, …, t10}. It is assumed that once the manufacturing of the products starts, no event in the production processes can be prevented from occurring. Therefore, only transitions t1, t2 and t10 are controllable. Places p2 and p3 represent the operation of R1 and M3 respectively for the production sequence of the part type P1. Similarly, places p2, p4, p5 and p6 represent the operation of R1, M1, R2 and M2 respectively for production sequence of the part type P1. For production sequence of the part type P2, places p9, p8 and p7 represent the operation of M2, R2 and M1 respectively. The number of tokens in p1, i.e. M(p1) = 3, represents the number of concurrent activities that can take place for P1. The number of tokens in p10, i.e. M(p10) = 3, represents the number of concurrent activities that can take place for P2. Places R1 and M3 denote the resources robot 1 and machine 3 respectively. Places M1, R2 and M2 denote the shared resources machine 1, robot 2 and machine 2 respectively. Initial markings of places R1, R2 and M3 are all one as robots can hold one part and machine 3 can process one part at a time. Initial markings of places M1 and M2 are all two as machine 1 and machine 2 can process two parts at a time. In order to have correct system behaviour, it is desirable that each production sequence can finish. It can be shown that this system suffers from deadlock problem and for this system the specification is to obtain the live system behaviour (non-blockingness in SCT terminology). The optimal (maximally permissive and non-blocking) controller consisting of three control places, namely pc1, pc2 and pc3, obtained by Ghaffari et al.[18] is depicted in Figure 37(b). When the controller is augmented with the uncontrolled PNM, the controlled model shown in Figure 37(c) is obtained. It was reported [18] that the controlled model shown in Figure 37(c) can produce the optimal live behaviour with 215 good states.

For this example, to verify the correctness of the control places pc1, pc2 and pc3, the proposed analysis approach is followed as shown below.

Step 1. The UPNM is the one given in Figure 37(a). Feedback control elements are three control places pc1, pc2 and pc3 as depicted in Figure 37(b), and the specification for the UPNM is non-blockingness with respect to the initial state, i.e. liveness enforcing.

Step 2. It can be verified that UPNM is bounded with the place marking bounds p1:3, p2:1, p3:1, p4:2, p5:1, p6:2, p7:2, p8:1, p9:2, p10:3, R1:1, R2:1, M1:2, M2:2 and M3:1.

Step 3. Each place of UPNM is converted into an equivalent buffer model as shown in Figure 38. The following event coding is used in the conversion process according to the TCT requirement that odd and even integers be used for controllable and uncontrollable events respectively.

Event coding for Example 3:

UPNM:	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
TCT:	0	1	21	30	4	50	6	70	8	90	101

The following steps are carried out using TCT. In the following, FMS defines automaton representation of UPNM

.

First, each place is created as an automaton as follows:

P1 = Create(P1,[mark 0],[tran [0,0,1],[1,0,2],[1,6,0],[1,30,0],[2,0,3],[2,6,1],[2,30,1],[3,6,2], [3,30,2]]) (4,9)

P2 = Create(P2,[mark 0],[tran [0,0,1],[1,1,0],[1,21,0]]) (2,3)

P3 = Create(P3,[mark 0],[tran [0,1,1],[1,30,0]]) (2,2)

P4 = Create(P4,[mark 0],[tran [0,21,1],[1,4,0],[1,21,2],[2,4,1]]) (3,4)

P5 = Create(P5,[mark 0],[tran [0,4,1],[1,50,0]]) (2,2)

P6 = Create(P6,[mark 0],[tran [0,50,1],[1,6,0],[1,50,2],[2,6,1]]) (3,4)

P7 = Create(P7,[mark 0],[tran [0,8,1],[1,8,2],[1,70,0],[2,70,1]]) (3,4)

P8 = Create(P8,[mark 0],[tran [0,90,1],[1,8,0]]) (2,2)

P9 = Create(P9,[mark 0],[tran [0,101,1],[1,90,0],[1,101,2],[2,90,1]]) (3,4)

P10 = Create(P10,[mark 0],[tran [0,101,1],[1,70,0],[1,101,2],[2,70,1],[2,101,3],[3,70,2]]) (4,6)

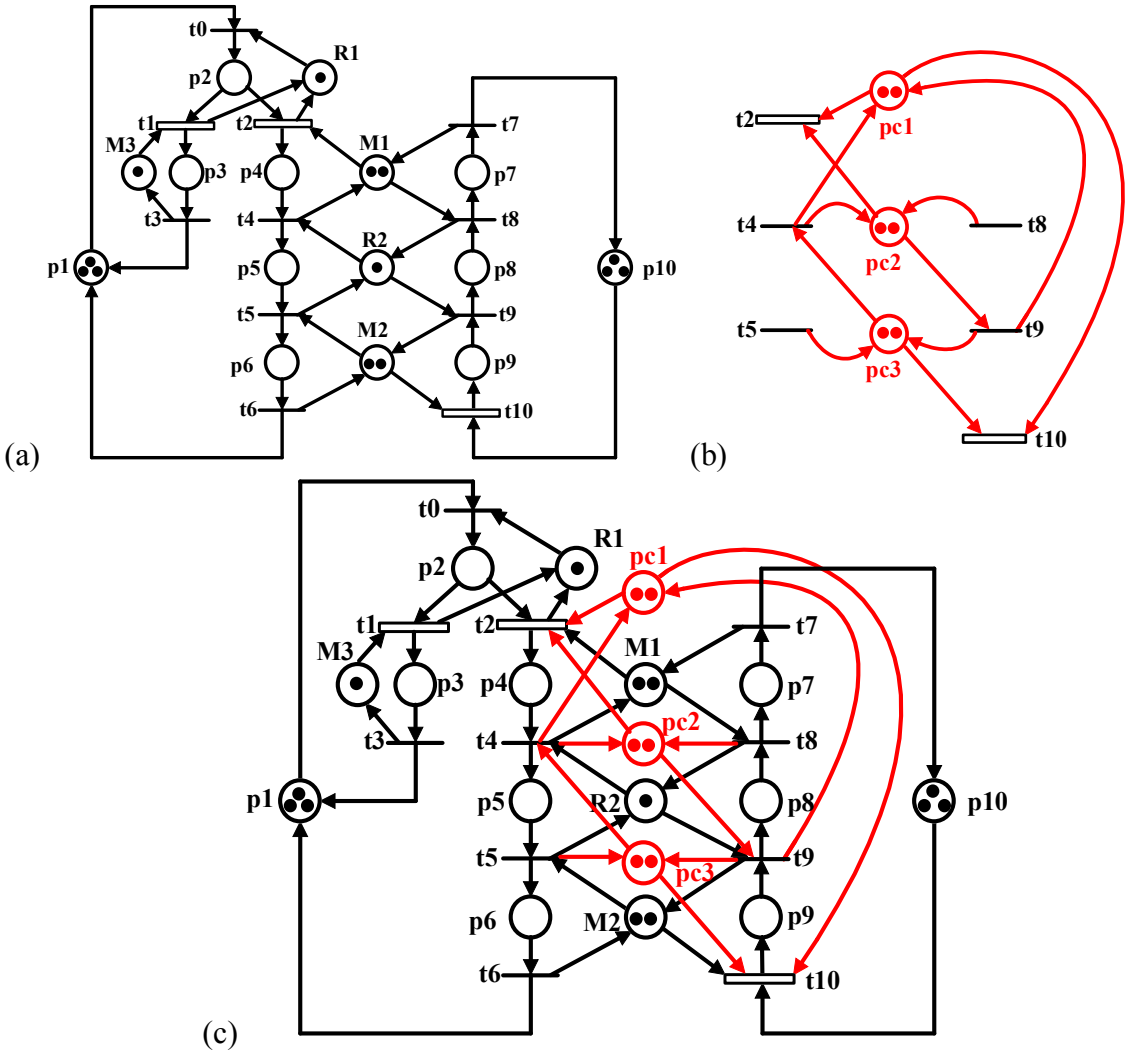


Figure 37. (a) PNM of the FMS [18]; (b) Three computed control places [18]; (c) Controlled model

R1 = Create(R1,[mark 0],[tran [0,0,1],[1,1,0],[1,21,0]]) (2,3)
R2 = Create(R2,[mark 0],[tran [0,4,1],[0,90,1],[1,8,0],[1,50,0]]) (2,4)
M1= Create(M1,[mark 0],[tran [0,8,1],[0,21,1],[1,4,0],[1,8,2],[1,21,2],[1,70,0],[2,4,1], [2,70,1]]) (3,8)
M2= Create(M2,[mark 0],[tran [0,50,1],[0,101,1],[1,6,0],[1,50,2],[1,90,0],[1,101,2],[2,6,1], [2,90,1]]) (3,8)
M3 = Create(M3,[mark 0],[tran [0,1,1],[1,30,0]]) (2,2)

Then the FMS is the synchronous product of all places, i.e. FMS = p1||p2||p3||p4||p5||p6||p7||p8||p9||p10||R1||R2||M1||M2||M3 and is obtained as follows:
FMS = Sync(P1,P2) (8,23) Blocked_events = None
FMS = Sync(FMS,P3) (16,36) Blocked_events = None
FMS = Sync(FMS,P4) (48,132) Blocked_events = None
FMS = Sync(FMS,P5) (96,280) Blocked_events = None
FMS = Sync(FMS,P6) (36,88) Blocked_events = None
FMS = Sync(FMS,P7) (108,408) Blocked_events = None
FMS = Sync(FMS,P8) (216,852) Blocked_events = None
FMS = Sync(FMS,P9) (648,2880) Blocked_events = None
FMS = Sync(FMS,P10) (504,2168) Blocked_events = None
FMS = Sync(FMS,R1) (504,2168) Blocked_events = None
FMS = Sync(FMS,R2) (426,1696) Blocked_events = None
FMS = Sync(FMS,M1) (341,1303) Blocked_events = None

FMS = Sync(FMS,M2) (261,933) Blocked_events = None
FMS = Sync(FMS,M3) (261,933) Blocked_events = None

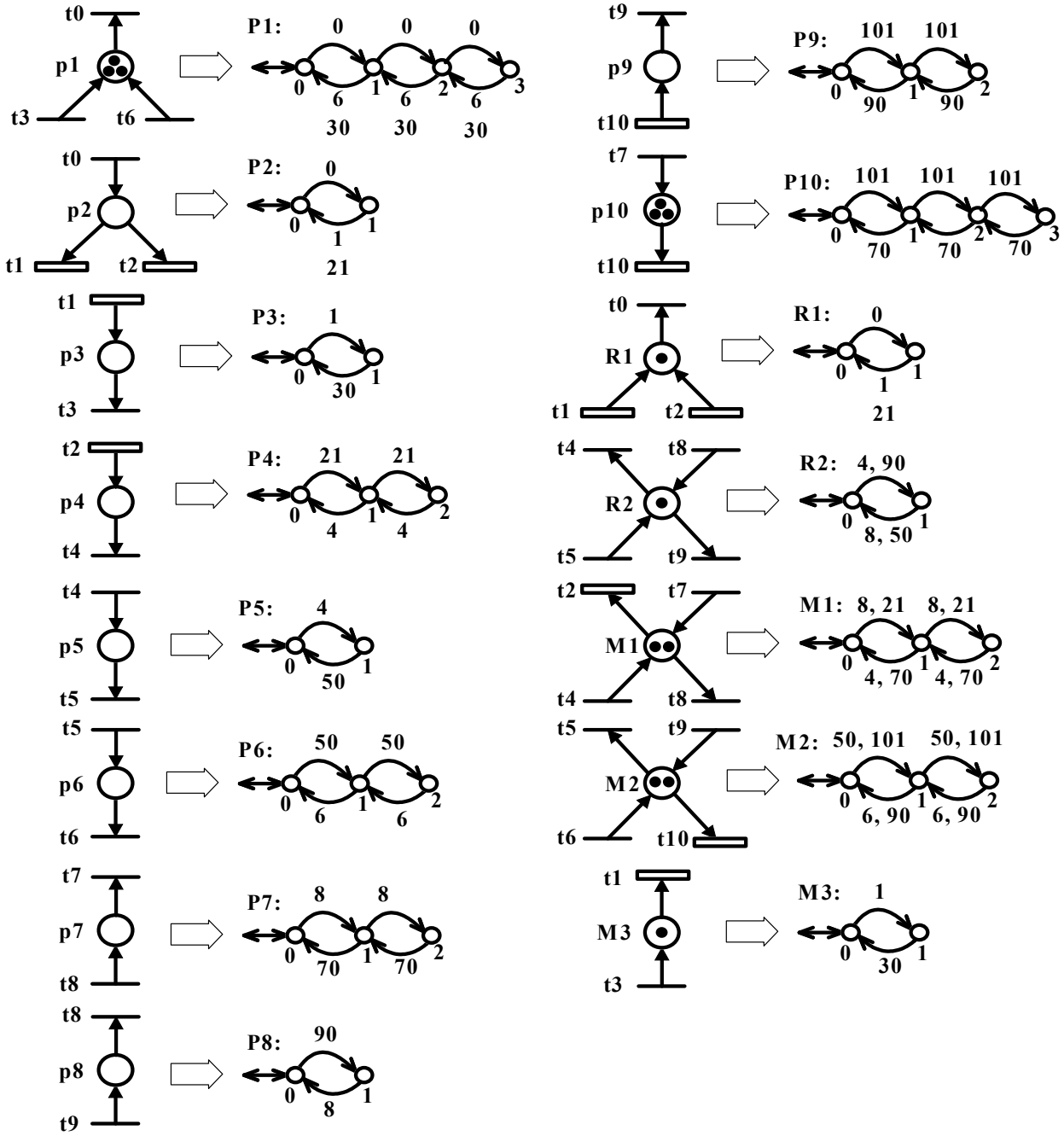


Figure 38. Mappings from UPNM to automata in Example 3

The problem considered is that of liveness enforcing problem. Therefore, the specification is non-blockingness with respect to the initial state and it is obtained by using Allevents() command as follows: ALL = Allevents(FMS) (1,11). In addition, the following test shows that FMS will block in the absence of control: false = Nonconflict(FMS,ALL).

Step 4. A RW supervisor RWSUPER is obtained as follows:
RWSUPER = Supcon(FMS,ALL) (215,771)

Step 5. The given feedback control elements, i.e. control places pc1, pc2 and pc3, are converted into equivalent buffer models as shown in Figure 39. Control places are created within TCT as follows:

PC1 = Create(PC1,[mark 0],[tran [0,21,1], [0,101,1], [1,4,0], [1,21,2], [1,90,0], [1,101,2], [2,4,1], [2,90,1]]) (3,8)
PC2 = Create(PC2,[mark 0],[tran [0,21,1],[0,90,1],[1,4,0],[1,8,0],[1,21,2],[1,90,2],[2,4,1], [2,8,1]]) (3,8)
PC3 = Create(PC3,[mark 0],[tran [0,4,1], [0,101,1], [1,4,2], [1,50,0], [1,90,0], [1,101,2], [2,50,1], [2,90,1]]) (3,8)

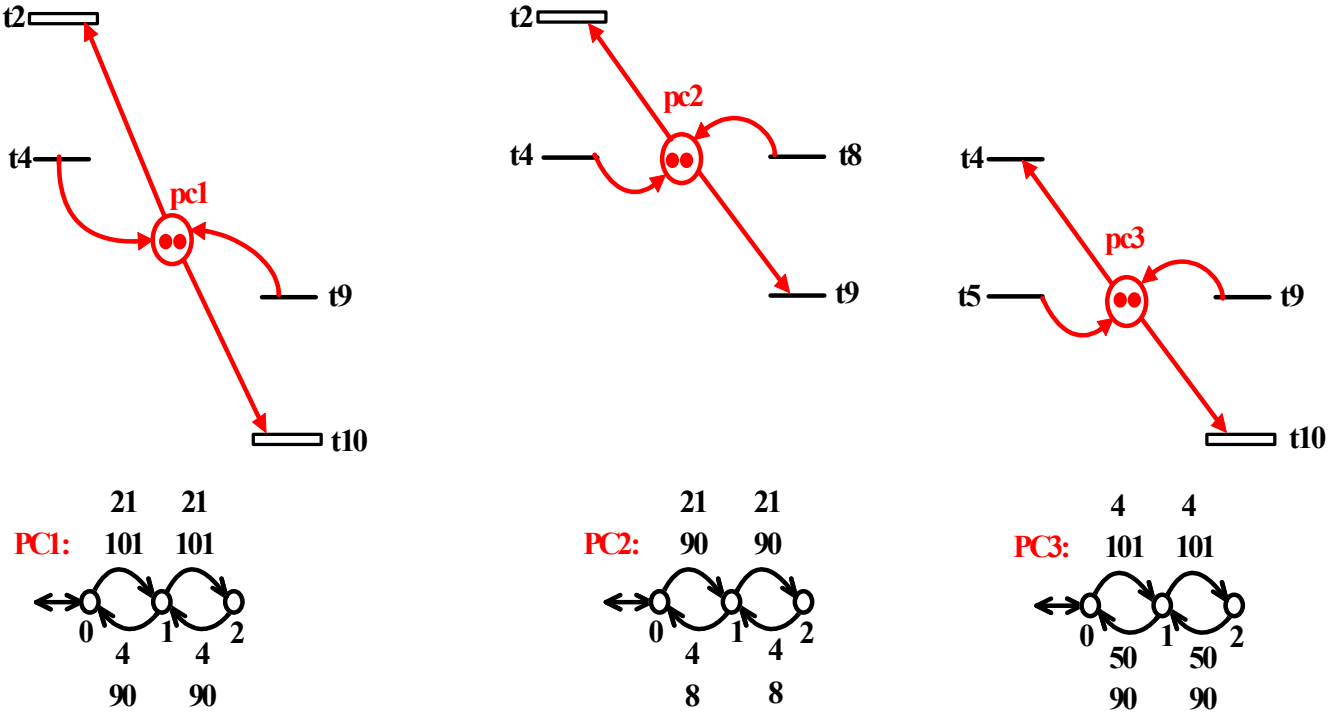


Figure 39. Mappings from control places pc1, pc2 and pc3 to automata in Example 3

Step 6. The controlled model PNSUPER (PNSUPER=FMS||PC1||PC2||PC3) is obtained by using FMS and buffer models of the given feedback control elements PC1, PC2 and PC3 as follows:
PNSUPER = Sync(FMS,PC1) (227,800) Blocked_events = None
PNSUPER = Sync(PNSUPER,PC2) (223,793) Blocked_events = None
PNSUPER = Sync(PNSUPER,PC3) (215,771) Blocked_events = None

Step 7. The following test verifies that the controlled model PNSUPER obtained is isomorphic (identical) to RWSUPER: true = Isomorph(RWSUPER,PNSUPER;identity). This means that the controller obtained in the PN domain is also a proper supervisor, i.e. it is maximally permissive and non-blocking.

The following shows that the optimal controlled behaviour PNSUPER (or RWSUPER) is not simply the language generated by the trimmed plant. This is because not all transitions are controllable. In this case the optimal controlled behaviour PNSUPER (or RWSUPER) is the supremal controllable sublanguage defining the maximally permissive and non-blocking behaviour.
TRIMFMS = Trim(FMS) (232,838)
false = Isomorph(PNSUPER,TRIMFMS)

7. Conclusions

This paper has proposed a novel and general methodology for carrying out the correctness analysis for the computed PN-based controllers by using the TCT implementation tool of SCT. Assuming that an uncontrolled PN model of a system, a set of specifications, and a PN-based controller are given, the proposed analysis method provides the result as to whether the PN-based

controller enforces the specification on the uncontrolled PN model optimally. Three examples have been considered for illustration.

8. Acknowledgements

This work was supported by the research grant of the Scientific and Technological Research Council of Turkey (Türkiye Bilimsel ve Teknolojik Araştırma Kurumu - TÜBİTAK) under the project number TÜBİTAK-107E125. The authors would like to thank the editor and three anonymous referees whose comments and suggestions greatly helped us to improve the presentation and the quality of the paper.

9. References

1. P. J. Ramadge and W. M. Wonham, "Modular supervisory control of discrete event systems", *Lect. Notes Contr. Inform. Sci.*, **1986**, *83*, 202-214.
2. P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes", *SIAM J. Contr. Optim.*, **1987**, *25*, 206-230.
3. P. J. Ramadge and W. M. Wonham, "The control of discrete event systems", *Proc. IEEE*, **1989**, *77*, 81-98.
4. W. M. Wonham and P. J. Ramadge "On the supremal controllable sublanguage of a given language", *SIAM J. Contr. Optim.*, **1987**, *25*, 637-659.
5. A. Giua, "Petri net techniques for supervisory control of discrete event systems", Proceedings of 1st International Workshop on Manufacturing and Petri Nets, **1996**, Osaka, Japan, pp. 1-21.
6. A. Giua, F. DiCesare and M. Silva, "Generalized mutual exclusion constraints on nets with uncontrollable transitions", Proceedings of IEEE International Conference on Systems, Man and Cybernetics, **1992**, Chicago, Illiois, USA, pp. 974-979.
7. K. Yamalidou, J. O. Moody, M. Lemmon and P. J. Antsaklis, "Feedback control of Petri nets based on place invariants", *Automatica*, **1996**, *32*, 15-28.
8. J. O. Moody and P. J. Antsaklis, "Petri net supervisors for discrete event systems with uncontrollable and unobservable transitions", *IEEE Trans. Automat. Contr.*, **2000**, *45*, 462-476.
9. R. Zurawski and M. C. Zhou, "Petri nets and industrial applications: A tutorial", *IEEE Trans. Ind. Electron.*, **1994**, *41*, 567-583.
10. M. Uzam, "On suboptimal supervisory control of Petri nets in the presence of uncontrollable transitions via monitor places", *Int. J. Adv. Manufac. Technol.*, **2010**, *47*, 567-579.
11. Systems Control Group, "TCT: A design software supporting supervisory control theory", www.control.utoronto.ca/DES (Accessed: **2009)**.
12. M. Uzam and G. Gelen, "The real-time supervisory control of an experimental manufacturing system based on a hybrid method", *Contr. Eng. Pract.*, **2009**, *17*, 1174-1189.
13. M. Uzam and W. M. Wonham, "A hybrid approach to supervisory control of discrete event systems coupling RW supervisors to Petri nets", *Int. J. Adv. Manufac. Technol.*, **2006**, *28*, 747-760.
14. M. V. Iordache and P. J. Antsaklis, "Supervisory Control of Concurrent Systems, A Petri Net Structural Approach", Birkhauser, Boston, **2006**, p. 21.
15. P. Darondeau, "Equality of languages coincides with isomorphism of reachable state graphs for bounded and persistent Petri nets", *Inform. Process. Lett.*, **2005**, *94*, 241-245.
16. M. Droste and R. M. Short, "From Petri nets to automata with concurrency", *Appl. Categor. Struct.*, **2002**, *10*, 173-191.
17. A. Egri-Nagy and C. L. Nehaniv, "Algebraic properties of automata associated to Petri nets and applications to computation in biological systems", *BioSystems*, **2008**, *94*, 135-144.
18. A. Ghaffari, N. Rezg and X. Xie, "Design of a live and maximally permissive Petri net controller using the theory of regions", *IEEE Trans. Robot. Automat.*, **2003**, *19*, 137-142.
19. E. Badouel and P. Darondeau, "Theory of regions", in *Lectures on Petri Nets I: Basic Models*, Vol. 1491, Springer-Verlag, Berlin, **1998**, pp.529-586.

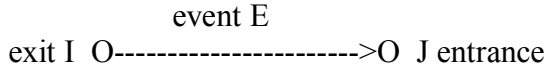
20. M. Uzam, "An optimal deadlock prevention policy for flexible manufacturing systems using Petri net models with resources and the theory of regions", *Int. J. Adv. Manufac. Technol.*, **2002**, *19*, 192-208.
21. M. Uzam, "The use of Petri net reduction approach for an optimal deadlock prevention policy for flexible manufacturing systems", *Int. J. Adv. Manufac. Technol.*, **2004**, *23*, 204-219.
22. M. Uzam, "Synthesis of feedback control elements for discrete event systems using Petri net models and theory of regions", *Int. J. Adv. Manufac. Technol.*, **2004**, *24*, 48-69.
23. C. A. Petri, "Kommunikation mit automaten schriften des rheinisch", Westfalischen Inst. fur Intrumentelle Mathematik and der Universitat Bonn (translated by C. F. Green), Suppl 1 to Tech Report RADC-TR-65-337, Applied Data Research Inc., New York, **1962**.
24. J. L. Peterson, "Petri Net Theory and the Modeling of Systems", Prentice-Hall Inc., Englewood Cliffs, N.J., **1981**.
25. M. A. Marsan, G. Balbo, G. Conte, S. Donatelli and G. Francesshinis, "Modelling with Generalized Stochastic Petri Nets", John Wiley and Sons Ltd., Turin, **1995**.
26. M. Uzam, "Petri-net-based supervisory control of discrete event systems and their ladder logic diagram implementations", *PhD. Thesis*, **1998**, University of Salford, UK. (Posted at URL: http://host.nigde.edu.tr/muzam/)
27. M. Uzam and A. H. Jones, "Discrete event control system design using automation Petri nets and their ladder diagram implementation", *Int. J. Adv. Manufac. Technol.*, **1998**, *14*, 716-728.
28. C. G. Cassandras and S. Lafortune, "Introduction to Discrete Event Systems", Kluwer Academic Publishers, New York, **1999**, pp. 30, 56-57, 63, 65.
29. W. M. Wonham, "Supervisory control of discrete event systems", Section 8.2, http://www.control.utoronto.ca/DES/ (Updated **2010**).
30. "A Petri net analysis tool" Software Version 1.0, **1987-1996**, Pedagogical University of Rzesów, Poland.
31. F. Basile, P. Chiacchio and A. Giua, "Suboptimal supervisory control of Petri nets in presence of uncontrollable transitions via monitor places", *Automatica*, **2006**, *42*, 995-1004.

10. Appendix

TCT: General Information
TCT is a program for the synthesis of supervisory controls for DES. Generators and recognisers are represented as standard DES in the form of a 5-tuple:

[**Size**, **Init**, **Mark**, **Voc**, **Tran**]

Size is the number of states (the standard state set is {0, ..., Size-1}), **Init** is the initial state (always taken to be 0), **Mark** lists the marker states, **Voc** the vocal states, and **Tran** the transitions. A vocal state is a pair [I,V] representing positive integer output V at state I. A transition is a triple [I,E,J] representing a transition from the exit (source) state I to the entrance (target) state J and having event label E. E is an odd or even non-negative integer, depending on whether the corresponding event is controllable or uncontrollable.



All DES transition structures must be deterministic: distinct transitions from the same exit state must carry distinct labels.

TCT: Synthesis Procedures. A quick review of some of the TCT commands used in this paper:

DES = Create(DES)
is a new discrete-event system (DES). Option 1 allows fast user input via a sequence of prompts, resulting in direct creation of a .DES file. Option 2 allows user to create a text (.ADS) file with any ASCII text editor; this file can be converted to a .DES file using the TCT procedure FD.

DES2 = Selfloop(DES1,[SELF-LOOPED EVENTS])
is DES1 augmented by self-loops at each state, with common event labels listed by the user. To prevent non-determinism, this list must not contain event labels appearing in DES1.

DES3 = Sync(DES1,DES2)
is the (reachable) synchronous product of DES1 and DES2.

DES3 = Supcon(DES1,DES2)
is a trim recogniser for the supremal controllable sublanguage of the marked ("legal") language generated by DES2 with respect to the marked ("plant") generated by DES1. DES3 provides a proper supervisor for DES1.

DES2 = Trim(DES1)
is the trim (reachable and co-reachable) substructure of DES1.

DES3 = Meet(DES1,DES2)
is the meet (reachable Cartesian product) of DES1 and DES2. DES3 need not be co-reachable.

DES2 =Allevents(DES1)
is a one-state DES self-looped with all the events of DES1.

True/False = Nonconflict(DES1,DES2)
tests whether DES1 and DES2 are non-conflicting, namely whether all reachable states of the product DES are co-reachable.

True/False = Isomorph(DES1,DES2)
tests whether DES1 and DES2 are identical up to renumbering of states; if so, their state correspondence is displayed.