

Chapter 2

Related Work

Let us now formally specify a concurrent control system by an *Interpreted Petri net*, which is a powerful mathematical tool and naturally reflects concurrency in prototyped systems. Note, that the prototyping, analysis and decomposition methods presented in the book apply to the *integrated concurrent control systems* (cf. Chap. 1). Therefore, unless otherwise stated, we use the simple notations *concurrent control system*, *concurrent controller* in reference to this particular group.

2.1 Petri Nets and Interpreted Petri Nets

This section introduces basic definitions and notations regarding the Petri nets [5, 7, 8, 10, 13, 15, 17, 19, 23–25, 27–31, 33–35, 39].

Definition 2.1 A *Petri net* is a 4-tuple:

$$PN = (P, T, F, M_0), \quad (2.1)$$

where

- P is a finite set of places,
- T is a finite set of transitions,
- $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs,
- M_0 is an initial marking.

Set $P \cup T$ is a set of *nodes* of a Petri net.

Definition 2.2 (*Marking*) State of a Petri net is called a *marking*, which can be seen as a distribution of tokens in the net places. If a place contains one or more tokens, it is called a *marked place*. A marking can be changed by means of *firing* (execution) of a transition.

Definition 2.3 (*Transition firing*) A transition t can be *fired* if every of its input places contains a token. Transition firing removes a token from every input place of t and adds a token to every output place of t .

An exemplary Petri net PN_1 (taken from [14, 37]) is shown in Fig. 2.1. The net contains six places $P = \{p_1, \dots, p_6\}$ and three transitions $T = \{t_1, t_2, t_3\}$.

Definition 2.4 (*Input and output places*) Place p is an *input* place of transition t , if $(p, t) \in F$. Place p' is an *output* place of transition t , if $(t, p') \in F$. The set of input places of transition t is denoted by $\bullet t$, while $t\bullet$ denotes the set of output places of t .

Definition 2.5 (*Input and output transitions*) Transition t is an *input* transition of place p , if $(t, p) \in F$. Transition t' is an *output* transition of place p , if $(p, t') \in F$. The set of input transitions of place p is denoted by $\bullet p$, while $p\bullet$ denotes the set of output transitions of p .

Definition 2.6 (*Reachable marking*) A marking M' is *reachable* from marking M , if M' can be obtained from M by a finite sequence of transition firings. When marking M is not specified explicitly, a reachable marking of a net is understood as a marking reachable from its initial marking M_0 .

A state of a Petri net is obtained by a distribution markers (tokens) on the places. Figure 2.2 illustrates all the possible markings of PN_1 . There are three reachable markings in the presented net, that can be reached by consecutive firing of transitions t_1 , t_2 and t_3 . Changes between particular states for this net can be simply denoted as: $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_3} M_0$.

Fig. 2.1 An exemplary Petri net PN_1

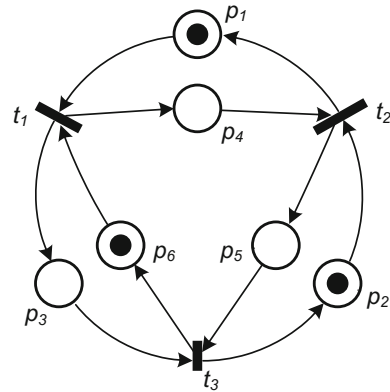
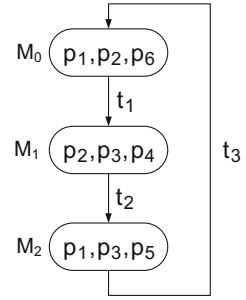


Fig. 2.2 All reachable markings of PN_1



The state M_0 consists of three places p_1 , p_2 and p_6 , which means that those places are initially marked. After firing of the transition t_1 , tokens from its input places (p_1 and p_6) are moved to the output places (p_3 and p_4) of t_1 and the net reaches marking M_1 . Please note that token at place p_2 remains unchanged. State M_1 enables transition t_2 , which execution leads to the third marking M_2 . At this state three places are marked: p_1 , p_3 and p_5 . Finally, firing of t_3 returns tokens into the initial marking M_0 .

Definition 2.7 (*Enabled transitions in a given marking*) For a Petri net, transition t is *enabled in marking* M if $\forall p \in \bullet t : p \in M$, that is, all its input places are simultaneously marked in M . Any transition enabled in M can fire, changing the marking M to M' . $M[t >$ denotes that t is enabled in M , while $M[>$ indicates the set of all enabled transitions in M .

For PN_1 , transition t_1 is enabled in M_0 , which is denoted by $M_0[t_1 >$. Furthermore, firing of t_1 changes an initial marking M_0 to M_1 . Such a situation is represented as $M_0[t_1 > M_1$. Similarly, $M_1[t_2 > M_2$. Finally, $M_2[t_3 > M_0$ moves tokens to the initial state.

Definition 2.8 (*Liveness*) A transition t is *live* if for every reachable marking M , t can be fired in M or in a marking reachable from M . A Petri net is *live* if all transitions in the net are live.

Definition 2.9 (*Safeness*) A place of a net is *safe* if there is no reachable marking such that the place contains more than one token. A Petri net is *safe* if each place in the net is safe.

The characterization of liveness and safeness has been studied by the researches all over the world. Fundamental theory regarding liveness and safeness of Petri nets was introduced in [6, 16]. Such analysis was extended in [15]. Finally, advanced algorithms allowing checking liveness and safeness of particular subclasses of Petri nets were proposed in [2–4, 21, 40].

Let us point out that analysis of liveness and safeness is out of the scope of this monograph. In our opinion, existing theorems and algorithms (especially those

presented in [2–4, 15, 16, 21, 40]) exhausted this subject enough. Therefore, we decided to focus on the concurrency and sequentiality aspects of analysis of Petri nets.

Definition 2.10 (*Reversibility*) A Petri net is *reversible* if for each marking M , the initial marking M_0 is reachable from M . In other words, Petri net is reversible if it can always reach its the initial state.

Definition 2.11 (*Well-formed net*) A Petri net is *well-formed* if it is *live*, *safe* and *reversible*.

Definition 2.12 (*Conservativeness*) A Petri net is *conservative* if all the reachable markings contain the same number of tokens.

Definition 2.13 (*Pure net*) A Petri net is *pure* if it has no self-loops.

Definition 2.14 (*Path, strong connectedness*) A *path* in a Petri net PN is a sequence of nodes, connected by arcs. A net is *strongly connected* if for any pair (n_i, n_j) of its nodes there is a path leading from n_i to n_j .

Theorem 2.1 [3] *Let PN be a live and safe Petri net. Then PN is strongly connected.*

Petri net PN_1 is live, safe and conservative, therefore it is well-formed. Moreover, the net is pure and conservative, since all the markings contain exactly three tokens. PN_1 is strongly connected, because for any pair of nodes there is a path that connects them.

Definition 2.15 (*Interpreted Petri net*) An *interpreted Petri net* is a well-formed Petri net, defined as a 6-tuple:

$$PN = (P, T, F, M_0, X, Y), \quad (2.2)$$

where:

- P is a finite set of places,
- T is a finite set of transitions,
- $F \subseteq (P \times T) \cup (T \times P)$, is a finite set of arcs,
- M_0 is an initial marking,
- $X = \{x_1, x_2, \dots, x_m\}$ is a binary vector of logic inputs,
- $Y = \{y_1, y_2, \dots, y_n\}$ is a binary vector of logic outputs.

Interpreted Petri nets are very often used to specify the real-life controllers. The system communicates with the environment via input and output signals. The inputs are associated with transitions while its outputs are bounded to places. The transition is enabled if all the transition inputs are active (or condition tied to the transition is fulfilled). Therefore, input signals may preserve the net conflict-free. We shall show such a situation later in this section. Notice, that from the definition an interpreted

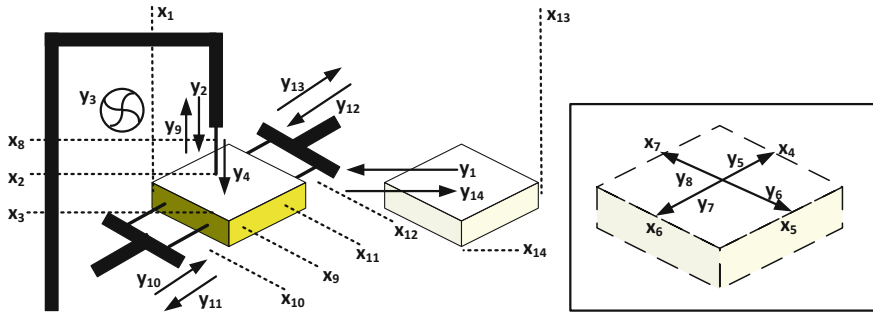


Fig. 2.3 A model of a milling machine

Petri net is well-formed, thus it is live, safe and reversible. It implies important additional properties. For example, based on the Theorem 2.1, each interpreted net is strongly connected.

Let us illustrate interpreted Petri nets by a real-life example. Figure 2.3 shows a modified system of the milling process, initially proposed in [38]. The main aim of the presented machine is to cut the square shapes from the wooden plank. The process is driven by a logic controller, specified by an interpreted Petri net PN_2 , shown in Fig. 2.4.

There are 14 input signals denoted by $x_1, \dots, x_{14}$, and 14 output signals marked as $y_1, \dots, y_{14}$. Placement of a wooden plank on the tray (indicated by the sensor x_{14}) starts the whole process. First, the plank is moved (output signal y_1), until reached the right position (signalized by x_1). Simultaneously, a drill (y_2) is being set into the starting position (x_2). Next, the machine starts cutting the required shape from the wood (in the presented example—a square). The move of the drill is denoted by y_4 (immersion into the wood), y_5 (the drill moves to the right), y_6 (the drill moves to the down), y_7 (the drill moves to the left), y_8 (the drill moves to the top), y_9 (the drill goes up, to the initial position). Reaching the remaining positions is signalized by sensors x_3, x_4, x_5, x_6, x_7 and x_8 , respectively. At the same time, while the shape is drilled, three concurrent actions are performed: a vacuum cleaner is turned on (y_3), and two assembly holes are drilled (signals y_{10}, y_{11} and y_{12}, y_{13}). Finally, the drilled shape is moved to the platform (y_{14}) to the tray. When the plank is taken away ($\bar{x}_{14}$), the system is ready for the further actions.

The net PN_2 consists of $|P| = 21$ places and $|T| = 17$ transitions. It is live, safe, reversible, and strongly connected. However, it is not conservative, since firing of t_1 moves and splits a single token from p_1 into p_2 and p_4 .

Figure 2.5 shows another real-life example of a concurrent control system. The picture illustrates an advanced traffic lights system, driven by the FPGA device. There are three independent traffic lights that control particular lanes for cars (turning left, going straight, turning right). Additionally, two traffic lights for pedestrians are considered. In our consideration a simplified version of the controller (initially shown in [36]) will be presented. Assume, that all the car lanes operate in the same manner

Fig. 2.4 An interpreted Petri net PN_2 that controls the milling machine

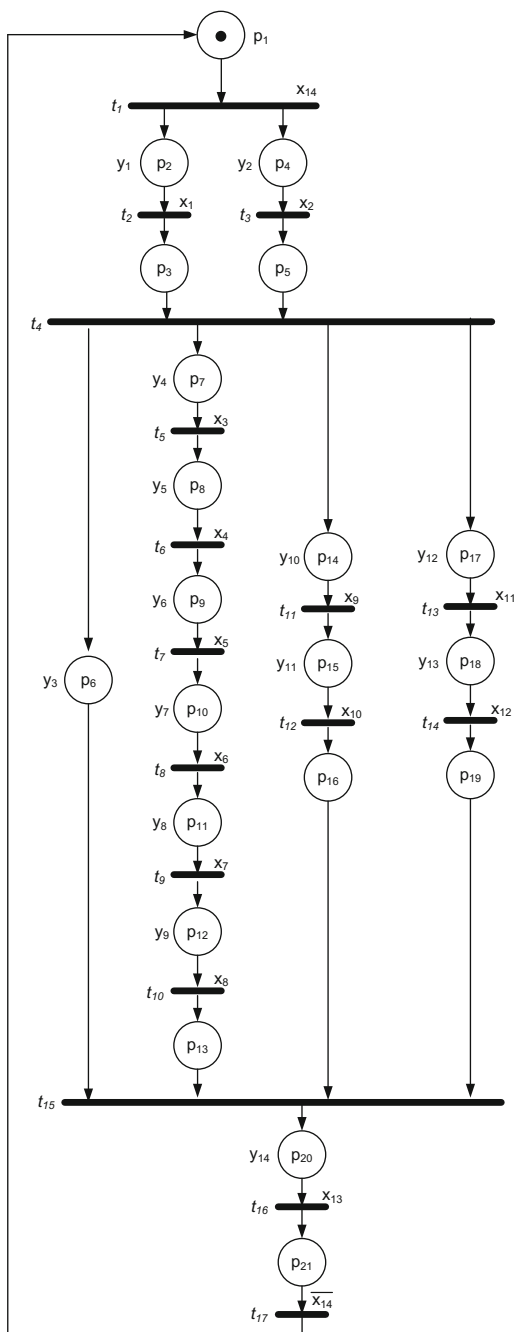


Fig. 2.5 Traffic lights system



and they are treated as a single traffic light. Similarly, both traffic lights for pedestrians are coupled. Such a system can be easily described by an interpreted Petri net (PN_3), as it is presented in Fig. 2.6. Initially, the red lights for cars (place p_4 , output R_C) and for pedestrians (place p_6 , output R_P) are active. If there is no request from pedestrians to cross the street (latched input signal req is inactive), the system enters the state, when the green light for cars (place p_2 , output G_C) is shown (note, that the red light for pedestrians prevents collision). Such a situation takes place until a pedestrian wishing to cross the street pushes the button (signal req), which results in firing the transition t_2 . Next, the yellow light for cars (place p_3 , output Y_C) is flashed and the system goes back to the initial state. Since signal req is active, the transition t_4 is enabled and executed. The green light for pedestrians is shown to cross the street (it is assumed, that signal req is zeroed). The system returns to the initial state and the whole procedure is repeated.

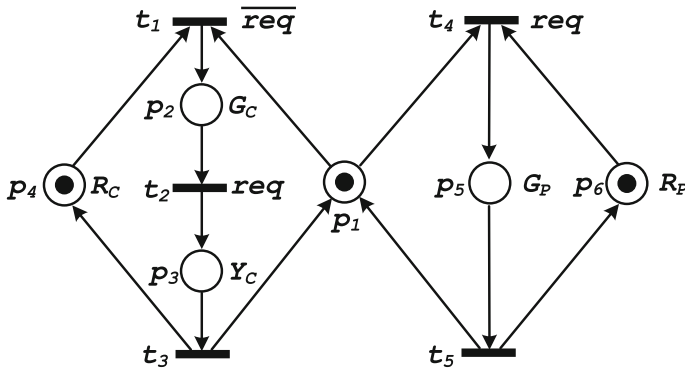


Fig. 2.6 An interpreted net PN_3 that controls the simplified version of traffic lights

Petri net PN_3 consists of $|P| = 6$ places and $|T| = 5$ transitions. It is live, safe, and reversible. The net is not conservative, because firing of t_1 consumes two tokens (from places p_1 and p_4), while only one token is moved to p_2 .

Note that place p_1 has two output transitions: $p_1 \bullet = \{t_1, t_4\}$. Such a situation is called *conflict* in a Petri net. In case of interpreted Petri nets, it can be easily resolved by the input signals. In the particular example, signal *req* indicates, which transition is enabled. If there are no conflicts in the net, it is called *conflict-free* Petri net. For example, PN_1 is conflict-free, because each place has exactly one output transition.

It is assumed that interpreted Petri nets shown in this book are free of conflicts. It means that either the net is conflict-free, or such a conflict is resolved by the input signals. Furthermore, any Petri net that is live, safe and conflict-free can be classified as an interpreted Petri net. Thus, $PN_1 = (P, T, F, M_0, \emptyset, \emptyset)$ is an interpreted Petri net, but the sets of its input and output signals are empty: $X = Y = \emptyset$. Of course one may say that a controller specified by such a net does not have any sense, since there is no communication with the environment. Obviously it is true. However, such nets can be successfully applied for theoretical purposes or at those prototyping stages (mainly analysis, cf. Chap. 5), where input and output signals are not considered.

Definition 2.16 (*Concurrency in interpreted Petri nets*) Two places are *concurrent* if they are marked simultaneously at some reachable marking.

Definition 2.17 (*Sequentiality in interpreted Petri nets*) Two places are *sequential* if they cannot be marked simultaneously, i.e., there is no marking that contains both places.

It is assumed, that (in case of interpreted Petri nets) concurrency and sequentiality are complementary, i.e., particular place is either concurrent or sequential to the other place.

Definition 2.18 (*Reachability set, concurrency set*) *Reachability set* or *concurrency set* of a concurrent control system is the set of its reachable states (markings).

Reachability (concurrency) set, beside obvious concurrency analysis, can be also used to verify the correctness of the system behavior [11]. Popular representation of such a set is *reachability graph*.

Definition 2.19 (*Reachability graph*) *Reachability graph* of a concurrent control system is the directed graph of its reachable states, and the direct transitions between them.

Figure 2.2 (shown at the beginning of current chapter) presents an exemplary reachability graph for Petri net PN_1 (from Fig. 2.1).

Petri nets are classified according to their structure. In our consideration we use the following classification of a Petri net (taken directly from [25]):

1. State Machine (SM),
2. Marked Graph (MG),

3. Free-Choice net (FC-net),
4. Extended Free-Choice net (EFC-net),
5. Simple Net (SN), also known as Asymmetric Choice net (AC-net).

Let us define each of the presented subclasses.

Definition 2.20 (*State machine, SM-net*) A *state machine* is a Petri net for which every transition has exactly one input place and exactly one output place, i.e., $\forall t \in T : |\bullet t| = |t \bullet| = 1$.

Definition 2.21 (*Marked graph, MG-net*) A *marked graph* is a Petri net for which every place has exactly one input transition and exactly one output transition, i.e., $\forall p \in P : |\bullet p| = |p \bullet| = 1$.

Definition 2.22 (*Free-choice, FC-net*) A *free-choice net* is a Petri net for which every outgoing arc from a place is unique or is a unique incoming arc to a transition, i.e., $\forall p \in P : |p \bullet| \leq 1$ or $\bullet(p \bullet) = \{p\}$.

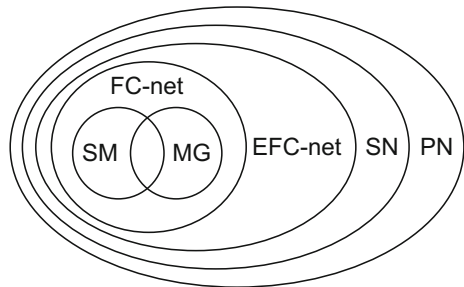
Definition 2.23 (*Extended free-choice net, EFC-net*) An *Extended Free-Choice net* is a Petri net for which every two places having a common output transition, have all their output transitions in common, i.e., $\forall p_i, p_j \in P : p_i \bullet \cap p_j \bullet \neq \emptyset \Rightarrow p_i \bullet = p_j \bullet$.

Definition 2.24 (*Simple net, SN*) A *Simple net* is a Petri net for which every two places having a common output transition, one of them has all the output transitions of the other (and possibly more), i.e., $\forall p_i, p_j \in P : p_i \bullet \cap p_j \bullet \neq \emptyset \Rightarrow (p_i \bullet \subseteq p_j \bullet)$ or $(p_i \bullet \supseteq p_j \bullet)$.

All the nets, that do not belong to any of above subclasses are just classified as *Petri nets* (PNs). For the interpreted Petri nets, we shall use the same abbreviations for their subclasses. For example an interpreted Petri net that is classified as an EFC-net, will be shortly classified as an *interpreted EFC-net*.

Particular subclasses are structured as follow: SM and MG are simplest possible structures of a Petri net. FC is a generalization of SMs and MGs. It means that each net that belongs to SM is also classified as an FC-net. Similarly, each MG is an FC-net, as well. Furthermore, EFC is a generalization of FC, while SN is a generalization of EFC. Finally, PN is a generalization of SN. Figure 2.7 illustrates the hierarchy of subclasses of Petri nets.

Fig. 2.7 Subclasses of Petri nets



Recall net PN_1 , shown in Fig. 2.1. Every place of PN_1 has exactly one input transition and exactly one output transition (there are no multiple arcs outgoing from places). It means that such a net belongs to MG. Moreover, it is automatically classified as FC-net, EFC-net, SN, and PN, as well. Similarly, net PN_2 is also classified as MG-net, but PN_3 belongs to SN.

Classification of a Petri net may be useful during analysis of the net [25]. For example, liveness, safeness of some subclasses (SM, MG, FC, EFC) can be checked polynomially [2–4, 15, 16, 21].

Definition 2.25 (*State machine component*) A state machine component (SMC, SM-component, S-component) of a Petri net $PN=(P, T, F, M_0)$ is a Petri net $S=(P', T', F', M_0)$ such that:

1. S is an SM-net,
2. S is strongly connected,
3. $P' \subseteq P$,
4. $T' = \bullet P' \cup P' \bullet$,
5. $F' = F \cap (P' \times T') \cup (T' \times P')$,
6. S contains exactly one token in M_0 .

Figure 2.8 shows all the state machine components that can be obtained in the net PN_1 . There are four SMCs, consisting of the following places: $S_1 = \{p_1, p_4\}$, $S_2 = \{p_3, p_6\}$, $S_3 = \{p_2, p_5\}$, $S_4 = \{p_4, p_5, p_6\}$.

Definition 2.26 (*SM-decomposition*) State machine decomposition (SM-decomposition, S-decomposition) of a Petri net $PN = (P, T, F, M_0)$ is a set $\mathcal{S}$ of elements (often called components or modules) $\mathcal{S} = \{S_1, \dots, S_n\}$ such that each place $p_i \in P$ is a place of exactly one component $S_j \in \mathcal{S}$. Each component S_j is an SM-net. If the particular place exists in more than one component, it is replaced by a place not belonging to P , called a non-operational place (NOP).

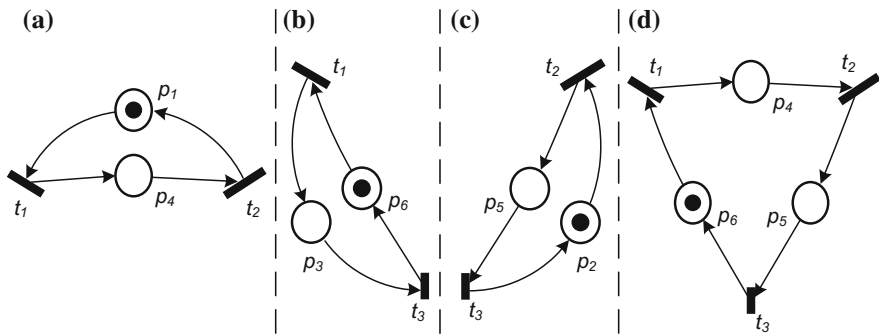


Fig. 2.8 All the SMCs of PN_1

Definition 2.27 (*Non-operational place, NOP*) Let PN be a Petri net and let S be a set of SMCs achieved during SM-decomposition of PN . If place $p \in P$ exists in more than one $S \in \mathcal{S}$, it is replaced by a *non-operational place* (NOP) in all $S \in \mathcal{S}$, but one. The set $P' = \{p_i, \dots, p_j\}$ of places of the same component $S \in \mathcal{S}$ can be replaced by a single NOP if there exists a path in P' leading from p_i to p_j . Then, all transitions and arcs between p_i and p_j are removed, as well. A NOP is initially marked if any place substituted by it is initially marked.

Let us explain decomposition and NOPs by examples. Petri net PN_1 can be decomposed into three components: $\mathcal{S} = \{S_1, S_2, S_3\}$, where $S_1 = \{p_1, p_4\}$, $S_2 = \{p_3, p_6\}$, and $S_3 = \{p_2, p_5\}$. This is the only one possibility to decompose PN_1 . There are no places that exist in more than one component, thus there is no need to apply non-operational places.

Consider Petri net PN_4 shown in Fig. 2.9a. There are six places in the net $P = \{p_1, \dots, p_6\}$ and four transitions $T = \{t_1, \dots, t_4\}$. The net is safe, live, and reversible. There are three SMCs in the PN_4 : $S_1 = \{p_1, p_3, p_4\}$, $S_2 = \{p_2, p_3, p_4, p_6\}$, $S_3 = \{p_5, p_6\}$.

Figure 2.9b illustrates one of the possible decompositions of PN_4 . In the presented example, two NOPs are applied, therefore the final set of decomposed components consists of the following places: $S_1 = \{p_1, p_3, p_4\}$, $S_2 = \{p_2, NOP_1, p_6\}$, $S_3 = \{p_5, NOP_2\}$.

The first non-operational place (NOP_1) is used in module S_2 . It substitutes places p_3 and p_4 that already exist in the first component S_1 . All the transitions and arcs

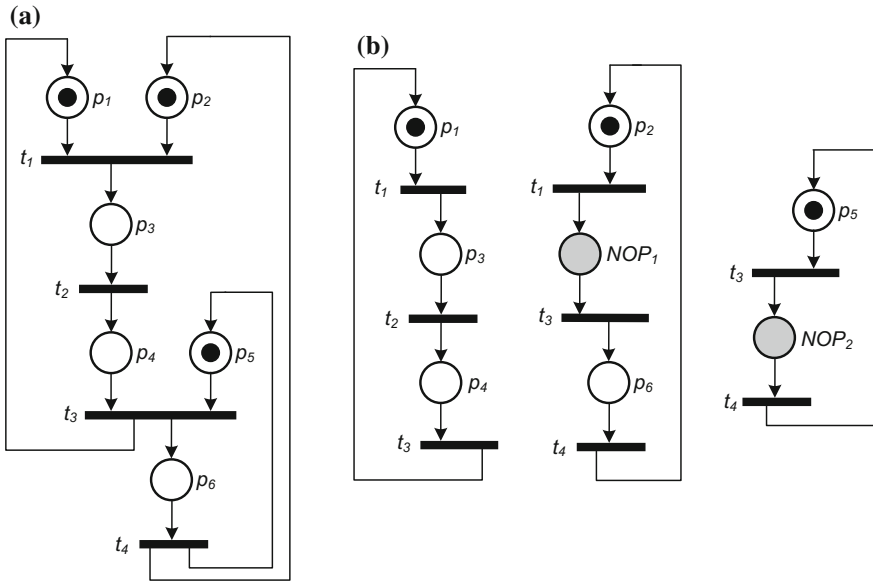


Fig. 2.9 Petri net PN_4 (a) and decomposed PN_4 (b)

belonging to the path connecting such places are removed as well. Furthermore, NOP_2 replaces p_6 in the third component S_3 , since p_6 already exists in S_2 .

Let us point out that there are more ways to decompose PN_4 , depending on the application of non-operational places. For example, the alternative decomposition of the net may consist of the following components: $S_1 = \{p_1, NOP_1\}$, $S_2 = \{p_2, p_3, p_4, NOP_2\}$, $S_3 = \{p_5, p_6\}$, while NOP_1 substitutes places p_3, p_4 , and NOP_2 replaces p_6 .

More information about decomposition of Petri nets as well as decomposition algorithms can be found in Chap. 6.

Definition 2.28 (*SM-cover*) *State machine cover (SM-cover, S-cover)* of a Petri net $PN = (P, T, F, M_0)$ is a set $\mathcal{C}$ of state machine components $\mathcal{C} = \{S_1, \dots, S_n\}$ such that each place $p_i \in P$ is a place of at least one component $S_j \in \mathcal{C}$.

SM-cover is very often confused with SM-decomposition. The main difference is that the particular place may exist in more than one component that belongs to the SM-cover. In opposite, the set of decomposed modules contains each place of the net exactly once. There are known conversion methods between SM-decomposition and SM-cover. Such a transformation can be done relatively easy, unless the conditions of the existence of SM-decomposition (or SM-cover) are not satisfied. More details can be found in [20].

Let us now introduce theorems regarding SM-decomposition and SM-cover. A very important relation between SM-cover and well-formed EFC-nets was shown in [9] (initially proposed for FC-nets in [15]):

Theorem 2.2 [9] *Well-formed EFC-nets are covered by SM-components.*

Since every interpreted EFC-net is well-formed, we immediately have

Theorem 2.3 *Interpreted EFC-nets are covered by SM-components.*

Proof Follows directly from Definition 2.15 and Theorem 2.2. □

Furthermore, the existence of SM-decomposition of a net depends on its safeness, which was proved in [20]:

Theorem 2.4 [20] *For a Petri net exists an SM-decomposition, if and only if PN is safe.*

Since every interpreted Petri net is safe by definition, we obtain a very important statement:

Theorem 2.5 *For an interpreted Petri net there always exists an SM-decomposition.*

Proof Follows directly from Definition 2.15 and Theorem 2.4. □

2.2 Computational Complexity of Algorithms

This section briefly introduces some preliminaries regarding computational complexity of algorithms. The presented notation we shall use during analysis of the computational complexity of algorithms shown in Chaps. 3–6.

The computational complexity presented in this book refers to the *time complexity*. Therefore, unless otherwise stated, we shall estimate the amount of *time* that is required in order to solve the problem. Note that only basic assumptions are presented. The detailed descriptions regarding computational complexity of algorithms can be found in [1, 12, 18, 22, 26, 32].

- *Time computational complexity of an algorithm* refers to the maximum number of interactions (steps) that the algorithm uses on a given size of a given input. Formally, *time complexity of algorithm $\mathcal{A}$* is the function $f : \mathcal{N} \rightarrow \mathcal{N}$, where $f(n)$ is the maximum number of interactions (steps) that $\mathcal{A}$ uses on any input of length n [32].
- *Big- O notation* is used to estimate the upper bound for the number of interactions (steps) executed by an algorithm. Formally, for $f(n) = O(g(n))$ we say that $g(n)$ is an upper bound for $f(n)$, where $f(n)$ is the maximum number of interactions of an algorithm [32]. Note, that *Big- O* refers to the worst-case complexity of the algorithm.
- *Polynomial bound of an algorithm* means that the upper bound of the algorithm can be represented in the form n^c for c greater than 0 [32]. In other words, algorithm for which the number of interactions is estimated as $f(n) = O(n^c)$ for $c > 0$ is bounded by a polynomial in the size of inputs n .
- *Exponential bound of an algorithm* means the bounds in the form c^n for $c > 1$. In other words, algorithm for which the number of interactions is estimated as $f(n) = O(c^n)$ for $c > 1$ is bounded by an exponential in the size of inputs n .
- *Total polynomial time, polynomial complexity* or just *polynomial time* means that the total run-time of an algorithm to generate all solutions (outputs) is bounded by a polynomial in the size of the input (cf. [18]).
- *Exponential complexity* or just *exponential time* means that the total run-time of an algorithm to generate all solutions (outputs) is bounded by an exponential in the size of the input (cf. [18]).
- *Polynomial delay* means that an algorithm generates results in such a way, that the time between subsequent outputs is bounded by a polynomial in the size of the input (cf. [12, 18]). We will say, that *subsequent outputs* are generated (computed, calculated) in *polynomial time*.

References

1. Aho AV, Hopcroft JE (1974) The design and analysis of computer algorithms. Addison-Wesley Longman Publishing Co. Inc., Boston
2. Barkaoui K, Minoux M (1992) A polynomial-time graph algorithm to decide liveness of some basic classes of bounded Petri nets. In: Proceedings of the 13th international conference on application and theory of Petri nets, Sheffield, UK, pp 62–75
3. Best E (1987) Structural theory of Petri nets: the free choice hiatus. In: Lecture notes in computer science, vol 254, New York, 1987. Springer, pp 168–206
4. Best E, Thiagarajan P (1987) Some classes of live and safe Petri nets. In: Voss K, Genrich H, Rozenberg G (eds) Concurrency and nets. Springer, Berlin, pp 71–94
5. Blanchard M (1979) Comprendre., Matriser Et Appliquer Le GrafctetCepadues, Toulouse
6. Commoner F, Holt AW, Even S, Pnueli A (1971) Marked directed graphs. J Comput Syst Sci 5(5):511–523
7. David R, Alla H (1992) Petri nets and grafctet: tools for modelling discrete event systems. Prentice-Hall Inc., Upper Saddle River
8. David R, Alla H (2010) Discrete, continuous, and hybrid Petri nets, 2nd edn. Springer, Incorporated
9. Desel J, Esparza J (1995) Free choice Petri nets. Cambridge University Press, New York
10. Desel J, Neuendorf K-P, Radola M-D (1996) Proving nonreachability by modulo-invariants. Theoret Comput Sci 153(1&2):49–64
11. Diaz M (1987) Applying Petri net based models in the design of systems. In: Voss K, Genrich H, Rozenberg G (eds) Concurrency and nets. Springer, Heidelberg, pp 71–94
12. Eiter T (1994) Exact transversal hypergraphs and application to boolean μ -functions. J Symbolic Comput 17(3):215–225
13. Esparza J, Silva M (1990) Top-down synthesis of live and bounded free choice nets. In: Applications and Theory of Petri Nets'90, pp 118–139
14. Hippo homepage. <http://www.hippo.iee.uz.zgora.pl>. Accessed 4 March 2016
15. Hack M (1974) Analysis of production schemata by Petri nets. MIT Project MAC TR-94 (1972) Corrections: Project MAC. Computation Structures Note 17
16. Holt A, Commoner F (1970) Events and conditions: introduction. In: Dennis JB (ed) Record of the project MAC conference on concurrent systems and parallel computation. ACM, New York, pp 3–5
17. Hu H, Zhou M, Li Z (2012) Liveness and ratio-enforcing supervision of automated manufacturing systems using Petri nets. IEEE Trans Syst Man Cybern Part A Syst Humans 42(2):392–403
18. Johnson DS, Papadimitriou CH, Yannakakis M (1988) On generating all maximal independent sets. Inf Process Lett 27(3):119–123
19. Karatkevich A (2007) Dynamic analysis of Petri net-based discrete systems, vol 356, Lecture notes in control and information sciences. Springer, Berlin
20. Karatkevich A, Wiśniewski R (2015) Relation between SM-covers and SM-decompositions of Petri nets. In: 11th International conference of computational methods in sciences and engineering (ICCMSE), volume 1702 of AIP conference proceedings, Greece, Athens, pp 1–4
21. Kemper P (2004) $O(|P||T|)$ -algorithm to compute a cover of S-components in EFC-nets
22. Knuth DE (1997) The art of computer programming: fundamental algorithms, vol 1, 3rd edn. Addison Wesley Longman Publishing Co., Inc, Redwood City, CA, USA
23. Kozłowski T, Dagless E, Saul J, Adamski M, Szajna J (1995) Parallel controller synthesis using Petri nets. IEE Proc Comput Dig Tech 142(4):263–271
24. Lasota A (2012) Modeling of production processes with UML activity diagrams and Petri nets. PhD thesis, University of Zielona Góra
25. Murata T (1989) Petri nets: properties, analysis and applications. Proc IEEE 77:548–580
26. Papadimitriou HC (1994) Computational complexity. Addison Wesley
27. Peterson JL (1981) Petri net theory and the modeling of systems. Prentice Hall PTR, Upper Saddle River

28. Petri CA (1962) Kommunikation mit automaten. Bonn: Institut fr Instrumentelle Mathematik, Schriften des IIM Nr. 2
29. Roguska A (2001) Evaluation of the practical interpreted Petri nets, grafcet networks and networks based on the SEC binary control system design, 2001. Bachelors Thesis, Technical University of Zielona Góra
30. Rozenberg G, Engelfriet J (1998) Elementary net systems. In: Lectures on Petri nets I: basic models, advances in Petri nets, the volumes are based on the Advanced course on Petri nets, London, UK, 1998. Springer, pp 12–121
31. Silva M (2013) Half a century after Carl Adam Petri's Ph.D. thesis: a perspective on the field. *Ann Rev Control*, 37(2):191–219
32. Sipser M (1996) Introduction to the theory of computation, 1st edn. International Thomson Publishing
33. Valette R (1978) Comparative study of switching representation tool with GRAFCET and Petri nets. *Nouv Autom* 23(12):377–382
34. Van Der Aalst W, Hee V (2004) Workflow management: models, methods, and systems. The MIT Press
35. Wiśniewska M (2012) Application of hypergraphs in decomposition of discrete systems, vol 23, Lecture notes in control and computer science. University of Zielona Góra Press, Zielona Góra
36. Wiśniewski R, Grobelna I, Grobelny M, Wiśniewska M (2015) Design and verification of distributed logic controllers with application of Petri nets. In: 11th international conference of computational methods in sciences and engineering (ICCMSE), volume 1702 of AIP conference proceedings, Greece, Athens, pp 1–4
37. Wiśniewski R, Stefanowicz Ł, Bukowiec A, Lipiński J (2014) Theoretical aspects of Petri nets decomposition based on invariants and hypergraphs. *Lecture notes in electrical engineering*, vol 308. Zhangjiajie, China, pp 371–376
38. Wiśniewski R, Grobelna I, Stefanowicz L (2016) Partial reconfiguration of concurrent logic controllers implemented in FPGA devices. In: 12th international conference of computational methods in sciences and engineering—ICCMSE'16, page TBD, Athens, Greece. Accepted for publication
39. Yakovlev A, Gomes L, Lavagno L (2000) Hardware design and Petri nets. Springer
40. Zakrevskij A, Pottosin Y, Cheremisinova L (2009) Design of logical control devices. TUT Press, Moskov

Prototyping of Concurrent Control Systems
Implemented in FPGA Devices

Wiśniewski, R.

2017, XI, 173 p. 90 illus., 26 illus. in color., Hardcover

ISBN: 978-3-319-45810-6