Grzegorz BOCEWICZ, Krzysztof BZDYRA*, Zbigniew BANASZAK**

TOWARDS AUTOMATED SYNTHESIS OF CONCURRENT CONTROL PROGRAMS

Abstract

One of the key problems in the area of automatic synthesis of real-time concurrent control programs is a problem of automated modeling of systems comprising multiple activities or processes that proceed concurrently. In that context, the discrete-event systems framework enabling to predict the modeled objects performance and then to design of assumed quality control program on the basis of the given specification of the processes controlled, plays a crucial role. From that point of view, the Petri nets can be seen as a most widely recognized tool aimed at modeling system's concurrency. So, the main objective of our contribution is to illustrate the Petri nets based approach from both: IF…THEN… rules system's behavior specification and its Turing machine like representation points of view.

1. INTRODUCTION

Predicting the performance of a computer, manufacturing, telecommunication, workflow, or transportation system is of primary importance in many day-life situations. Such systems usually comprise multiple activities or processes that proceed concurrently. In that context, as a modeling framework a class of discrete-event systems (DESs) [4,6,8] that can be viewed as making state transitions when events associated with the occupied state occur is usually applied. Among the possible DEDs representations the Petri nets [1,2,5,7] can be seen as a most widely recognized tool aimed at modeling system's concurrency.

A key problem arising in concurrent processes control regards of the resources conflict resolution. Such problems often arise in the systems where concurrently flowing processes compete with an access to common shared resources. Any two sequential processes where it is not possible to tell in advance which operations of one of them are preceded by the operations of the anther processes are treated as concurrent ones.

Among the examples of systems where concurrent processes dominate are:

- Operating systems supervising processes competing with an access to the resources of the computer system, e.g. CPU time, memories, peripheral devices, etc.

* Dept. of Management and Computer Science, Koszalin University of Technology, Śniadeckich 2, 75-453 Koszalin, bocewicz@ie.tu.koszalin.pl, krzysztof.bzdyra@tu.koszalin.pl

** Dept. of Business Informatics, Warsaw University of Technology, Narbutta 85, 02-524 Warsaw, z.banaszak@wz.pw.edu.pl

- Transmission communication networks, where broadcastings processes along routes containing specific network compete with an access to its resources, e.g. buffers, nodes, etc.
- Production systems, where concurrently flowing energy, money, material and information processes involved in simultaneous manufacturing of different products compete with an access to its resources, e.g. machine tools, robots, conveyors, storages, IT equipment, etc.

The general goal of these systems is executing certain tasks (concurrent processes), which must gain to a specific number of units of appropriate resources.

Executing process operations requires allocating to them specific and necessary for their completion, system resources. In most cases, however, the amount of available system resources is smaller than the amount of resources utilized by the processes executed. This situation results in processes competition, i.e. a resource conflicts occurrence. Consequently, it is necessary to synchronize processes activity as to supervise processes execution, aimed at:

- resources conflicts resolution guaranteeing deadlock-free and starvation-free processes execution,
- optimization of the rate of system resources utilization, ensuring efficient operation of the system while maximal processes concurrency

In order to illustrate synchronization issues and techniques for resolving they let us consider the dining philosophers problem.

The dining philosophers. At a round table sit five philosophers who alternate between thinking and eating from a large bowl of spaghetti at random intervals. Each philosopher must alternately think and eat. Eating is not limited by the amount of spaghetti left: assume an infinite supply. Unfortunately for the philosophers, there are only five forks at the table, one left and one right of each seat. (see Fig. 1). For eating each philosopher (the process) needs two forks (the resources). When a philosopher cannot grab both forks it sits and waits. Since the chopsticks are shared between the philosophers, access to the chopsticks must be protected, but if care isn't taken, concurrency problems arise. Most notably, starvation (pun intended) and deadlock (seen as a situation in which no progress is possible) can occur, i.e. if each philosopher picks up a chopstick as soon as it is available and waits indefinitely for the other, eventually they will all end up holding only one chopstick with no chance of acquiring another (see Fig. 1). There exist two deadlock states when all five philosophers are sitting at the table holding one for each. One deadlock state is when each philosopher has grabbed the fork left of him, and another is when each has the fork on his right.

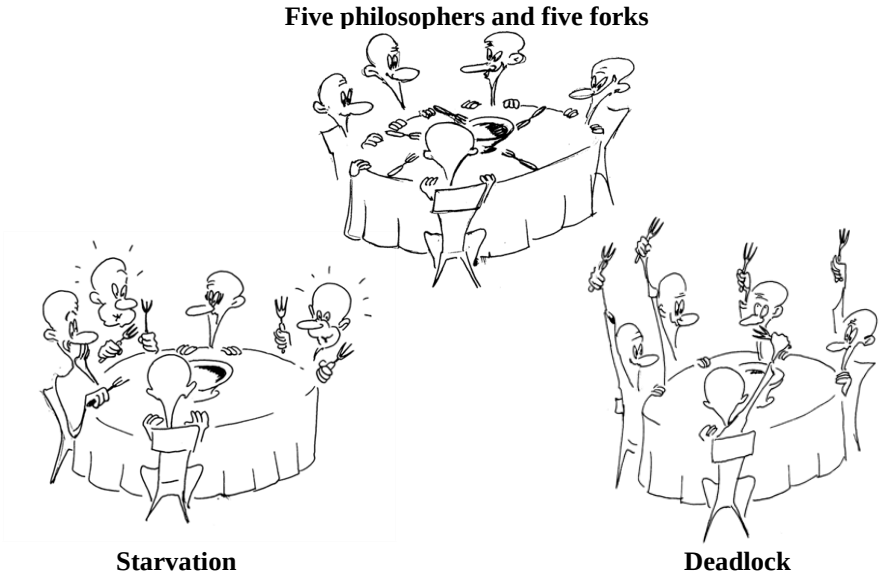


Fig. 1. The dining philosophers

So, assuming that a philosopher can only eat while holding both the fork to the left and the fork to the right, and can pick up an adjacent fork, when available, and put it down, when holding it (forks must be picked up and put down one by one) the main problem is how to design a discipline of behavior (e.g., seen as a concurrent algorithm implementing a proper synchronization mechanism) such that each philosopher won't starve, i.e. can forever continue to alternate between eating and thinking, while avoiding deadlock namely, the state in which each philosopher has picked up the fork to the left, waiting for the fork to the right to be put down.

Solution to this problem is a synchronization protocol guaranteeing required system behavior. An example of possible solution is shown in Fig. 2, where an order of the five snapshot-like cases specifies the protocol guaranteeing pleasant, i.e., starvation and deadlock free, consumption. Of course, in general case one may design may many others, for instance less fair, protocols.

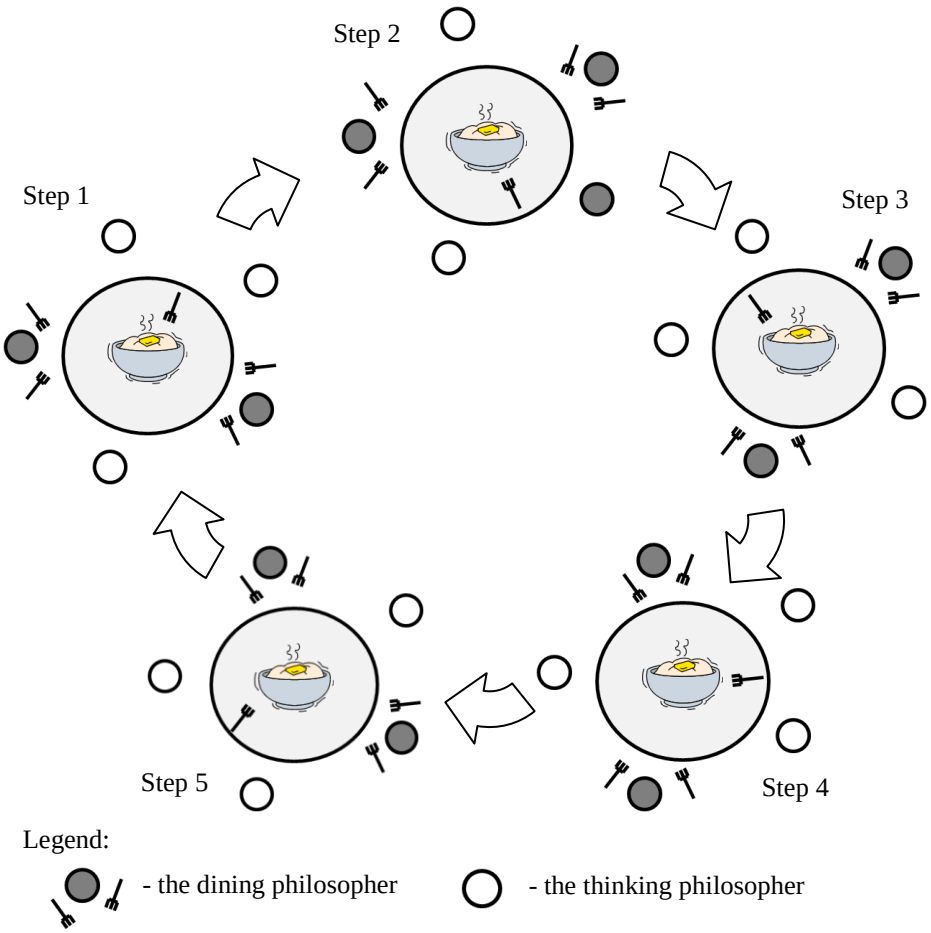


Fig. 2. Illustration of a synchronization protocol

To summarize, let us note that the failures these philosophers may experience are analogous to the difficulties that arise in real life, e.g. computer programming when multiple programs need exclusive access to shared resources. In this case the difficulties studied in the Dining Philosophers problem arise when multiple processes access sets of data that are being updated.

Deadlock handing. Let us consider a class of systems consisting of some sets of non-preemptive (i.e., for which an operation once started cannot be suspended until its completion) and reusable resources (a synchronization protocol (i.e., which are returned to the system after having been used by a process). I a synchronization protocol is assumed that:

- the number of units of each resource kind is finite and constant,
- a resource unit can be released (made again available to other processes) only by the process to which it was allocated.

The system resource utilization by correctly constructed processes determines the following order:

- **Request**; recording of the resource allocation request
- **Use**; allocating the resource
- **Release**: release of the resource.

As previously mentioned, non-preemptivness of the resources causes resource conflicts related to the fact that any given resource unit can be used by only one process at a time. Such a resource (in general a set of resources) is a critical section and all processes requesting that resource compete for access to it. The problem of mutually excluding processes related to the presence of critical sections results in the non-continuous operation of processes because they must at times wait for access to the resource which is the critical section.

Analyzing processes cooperation leads to the conclusion that mutual process blocking (deadlock) occurs when both processes gain access to the critical section consisting of the resources T and S. In such a state process p_1 using resource T is waiting for the release of the resource S which is used by process p_2. In turn, process p_2¬ using resource S is waiting for the resource each resource each resource T each resource which is used by process p_1 (see Fig. 3).

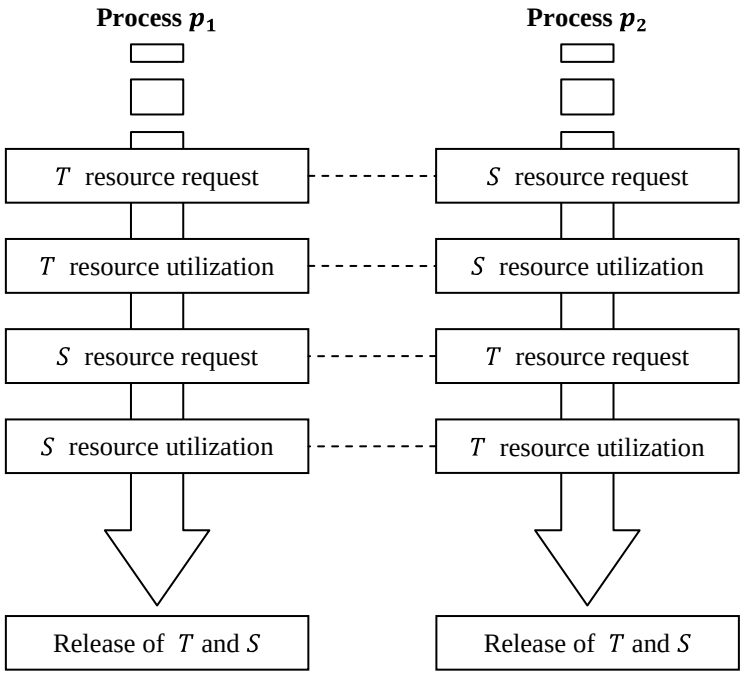


Fig. 3. An example of two processes cooperation while accessing the critical section containing two resources T and S.

One should note, however that processes description implies only a potential for blocking which actual occurrence depends on the relative speed of processes.

That was shown [1] a deadlock can arise only when the following four conditions are simultaneously satisfied:

- Mutual exclusion; at any given moment only one process can use a given resource,
- Non-preemption, a resource can be release only by the process which first took possession of it,
- Hold-while-waiting; while waiting for resources to be released by other processes a process does not release the resources allocated to him,
- Circular wait; there exists a closed cyclic chain of processes waiting for release of resources possessed by other processes.

That means, that in order to prevent blocking in the system it suffices to ensure that at least one of the above mentioned conditions will never hold. From this observation, the following problem aimed at deadlock handling can be stated.

Deadlock prevention

Protocols specifying such ways of resources requesting which make it impossible to satisfy one of the conditions necessary for blocking are sought.

Deadlock avoidance

A way of using the system resources by each process and allowing such selection of resource requests which ensure the system transition from one safe state to another is sought.

Deadlock detection and recovery

A ways allowing determining which processes and resources cause deadlock, and removing it by successive reallocating of resource units from the blocked processes and checking whether that helps to recover the blocking.

2. MODELLING OF COMPETING PROCESSES INTERACTION

The Petri net view of a system concentrates on two concepts, namely, transitions and conditions. Transitions are actions which take place in the system, and the occurrence of these transitions is controlled by the state of the system. The state of the system may be described as a set of conditions. A condition is a logical description of the state of the system.

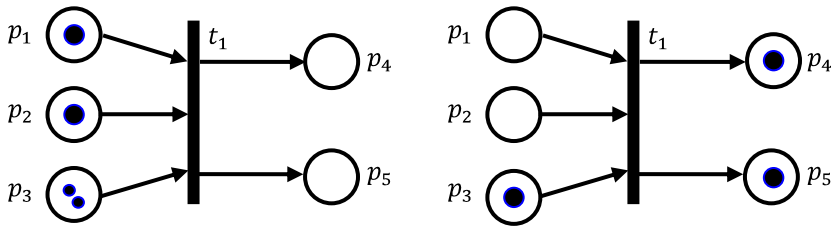
For a transition to occur, it may be necessary for certain conditions to hold and these are termed the preconditions of the transition. The occurrence of the transition may cause the precondition to case to hold and may cause other conditions, called postconditions, to be true.

A Petri net needs four entities for description: **Places**, **Transitions**, **Inputs** and **Outputs**. Places are used to represent conditions and transitions are used to represent events. The inputs are mappings from transitions to places and the outputs are mappings from places to transitions.

Markings are used in Petri nets to assign tokens to places. Tokens move through a Petri net and are used to define the execution of the Petri net. A Petri net executes by firing transitions. A transition is fired by removing tokens from its input places and creating new tokens in the output places. The removal/creation of tokens is controlled by the input/output functions of the Petri net. The firing of a transition is through enablement. A transition is enabled if there is a token in all the places specified by the output function mapping for the transition. Fig. 4 illustrates above defined entities and the firing of a transition.

The modeling of a system requires the determination of the various events and conditions which will occur and the relationships between the various events and conditions. These are, respectively, transitions, places and input/output mappings. In the system model it is apparent that events can be enabled simultaneously, hence concurrent systems can be modeled.

For illustration consider a flexible manufacturing cell shown in Fig. 5.

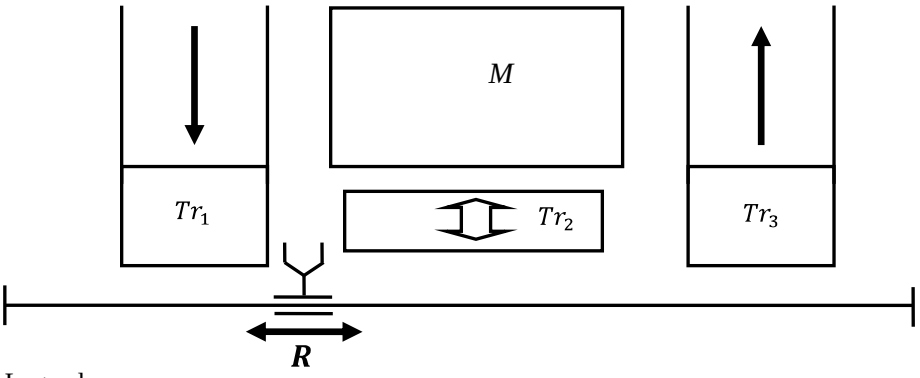


Legend:

p_i– the i-th place, t_1 – the transition,

$M = (1, 1, 2, 0, 0)$ – the marking, where $M(p_1) = 1$, $M(p_2) = 1$, $M(p_3) = 2$, $M(p_4) = 0$, and $M(p_5) = 0$,

Fig. 4. An example of the Petri net; state before a), and after b) firing.



Legend:

Tr_i – the i-th conveyor, M – the machine tool, R – the industrial robot,

Fig. 5. The flexible manufacturing cell

During the technological process in the cell considered, a workpiece waiting on conveyor Tr_1 is picked up by robot R and placed on conveyor Tr_2. Then the workpiece is carried by Tr_2 to the machine M, where its machining is performed. After completion of the operation workpiece is placed by M on Tr_2 that moves it to the position from which it is taken by robot R and placed on conveyor Tr_3.

It is assumed that there is always a workpiece waiting for machining on conveyor Tr_1 and that Tr_3 is always ready to accept the consequent completed part. However, Tr_2 can contain only one part at a time and its performance (move to/from the machine tool) is determined by the existence of a workpiece waiting for machining or to be already completed one. The machine tool starts its operation as soon as a part appears on it. Robot R starts one of the transportation operations (from Tr_1 to Tr_2 or from Tr_2 to Tr_3) immediately after it has completed a previous one and providing the existence of a part waiting for transport

is signaled. The task considered consists in the development of a procedure for the cell components control that allows the execution of the technological process given.

Let us assume that the manner of system operation can be specified by a given set of production rules:

R_1: IF $\boxed{s \text{ on } Tr_1}$ AND R exempt$^{\mathbf{a}}$ THEN s on$^{\mathbf{b}}$ R

R_2: IF s on$^{\mathbf{b}}$ R AND Tr_2 exempt$^{\mathbf{c}}$ THEN s on$^{\mathbf{d}}$ Tr_2 AND R exempt$^{\mathbf{a}}$

R_3: IF s on$^{\mathbf{d}}$ Tr_2 AND M exempt$^{\mathbf{e}}$ THEN s on$^{\mathbf{f}}$ M AND Tr_2 exempt$^{\mathbf{c}}$

R_4: IF s on$^{\mathbf{f}}$ M AND Tr_2 exempt$^{\mathbf{c}}$ THEN w on$^{\mathbf{g}}$ Tr_2 AND M exempt$^{\mathbf{e}}$

R_5: IF w on$^{\mathbf{g}}$ Tr_2 AND R exempt$^{\mathbf{a}}$ THEN w on$^{\mathbf{h}}$ R AND Tr_2 exempt$^{\mathbf{c}}$

R_6: IF w on$^{\mathbf{h}}$ R AND $\boxed{Tr_3 \text{ exempt}}$ THEN R exempt$^{\mathbf{a}}$

where: "s" and "w" denote the workpiece awaiting for machining and workpiece already completed.

The conditions put in a frame are always fulfilled (according to what has been assumed earlier). The letter notations **a** – **h** are abbreviations stating for the conditions considered.

Employing these notations and omitting the conditions always fulfilled (i.e., given in the frames), the set of production rules can be formulated as follows:

R_1: IF a THEN b,
R_2: IF b AND c THEN d AND a,
R_3: IF d AND e THEN f AND c, (1)
R_4: IF f AND c THEN g AND e,
R_5: IF g AND a THEN h AND c,
R_6: IF h THEN a.

The problem specification given above allows employing a Petri net model. An idea standing behind of the modeling concept assumes that each rule of the form (2) can be represented by a net shown in Fig. 6.

R_i: IF p_1 AND p_2 AND ... AND p_n THEN p_{n+1} AND ... AND p_m (2)

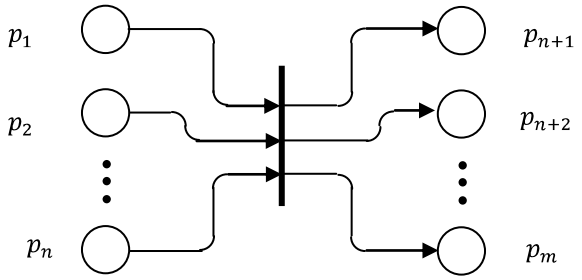
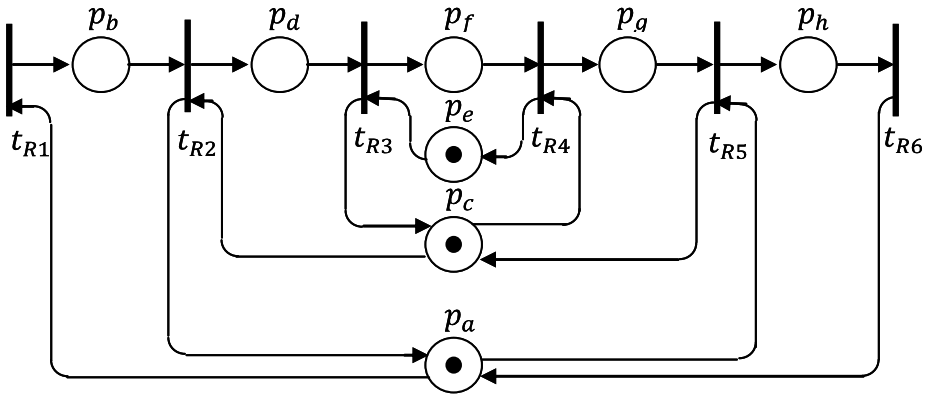


Fig. 6. The Petri net model of the rule R_i (2).

Taking into account possible graphical representation of the IF…THEN… rule the resultant Petri net model of the control procedure sought can be designed (see Fig. 7).



Legend:

p_a, p_c, p_e – places modeling conditions encompassing the resources state, i.e., the robot R, the conveyor Tr_2, the machine tool M, respectively

p_b, (p_h), p_d, (p_g), p_f – places modeling conditions encompassing the workpiece state, i.e., transported by robot R from Tr_1 to Tr_2 (from Tr_2 to Tr_3), transported by Tr_2 to (from) machine M, machined on M, respectively,

t_{R1}, t_{R2}, t_{R3}, t_{R4}, t_{R5}, t_{R6} – transitions modeling events encompassing actions corresponding to the workpiece picking up from Tr_1 and caring to Tr_2, the workpiece placing on Tr_2 and caring by Tr_2 to M, the workpiece placing to M and machining beginning, the workpiece releasing and placing on Tr_2, the workpiece picking up from Tr_2 and caring to Tr_3, the workpiece placing on Tr_3,

Fig. 7. The net model of the control procedure sought

The initial marking assumed in the Petri net model from Fig. 7 corresponds to the situation in which all the system resources are exempted (see the tokens in the places corresponding to particular resources), hence no technological operation is being executed. It should also be noted that such a state implies that the conditions "**a**", "**c**" and "**e**" are fulfilled. The transitions t_{R1}, t_{R2}, t_{R3}, t_{R4}, t_{R5}, t_{R6} correspond to IF…THEN… rules specifying the considered events.

The performance evaluation. The model obtained can be used then for analysis of the modeled cell functioning. A typical technique for such evaluation is a analysis of a reachability graph (or reachability tree). The reachability graph is a finite representation of a reachability set, i.e. the set containing the all markings reachable from the initial marking M_0. The nodes represent markings of the Petri net and the arcs represent the possible changes in the state resulting from the firing of transitions. Inspection of the graph will indicate, for instance, for a given marking *M*, if marking *M'* is reachable from *M*.

The reachability graph constructed for the model given in Fig. 7 is presented in Fig. 8. Results of analysis conducted show that the modeled behavior does not fulfill the requirements assumed, i.e. concerning the deadlock-free execution of production process. In other words,

it means that the control procedure mapping the behavior of this model can lead to the process deadlock.

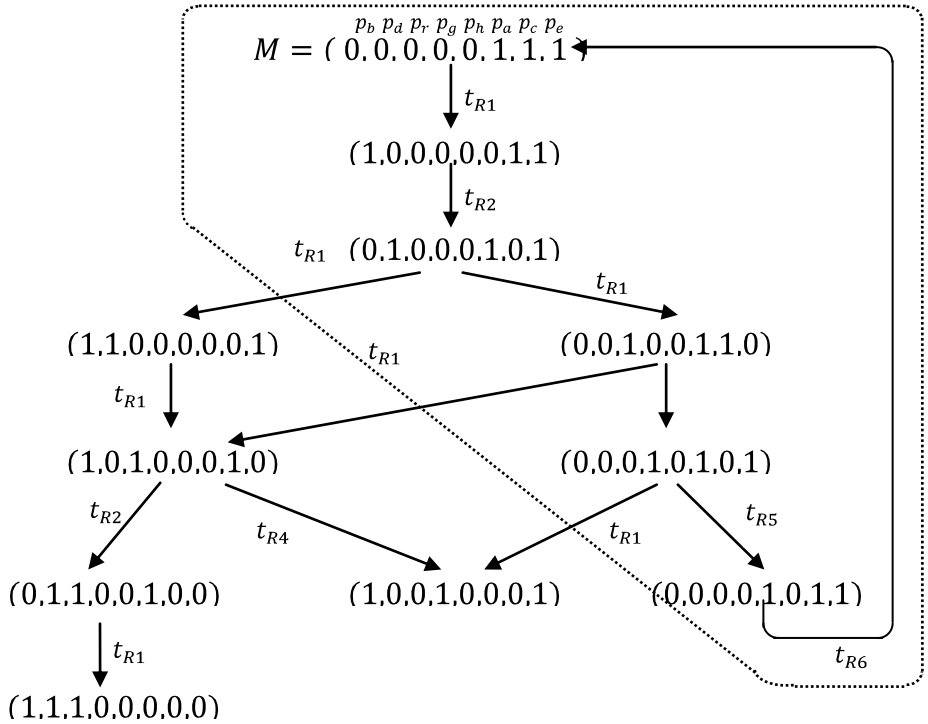


Fig. 8. Reachability graph of the Petri net model from Fig. 7

An example of such situation is the state that corresponds to the marking

$$M = (1,0,0,1,0,0,0,1)$$

with $p_b\ p_d\ p_r\ p_g\ p_h\ p_a\ p_c\ p_e$

in which a completed workpiece is waiting on Tr_2 for the robot R, while the next workpiece, picked up from Tr_1 and being kept by R, is waiting for Tr_1 to be released. So, the set of admissible states is that included in the area restricted by dashed line in Fig. 8.

The analysis of the reachability graph allows the model modification aimed at the development of such its version whose behavior would be represented only by feasible executions of the modeled process. In the example considered, it results from the graph analysis that the version modified should provide t_{R1} not to be enabled at any marking M such that $M(p_a) = 1$ and $M(p_c) = 1$ as well as at marking M' such that $M'(p_a) = 0$ and $M'(p_c) = 1$.

It should be noted that the problem specification manner presented is of heuristic nature and, in general case, it can lead to the construction of inadequate models, which do not allow the analysis of the system functioning. This implies the necessity of multiple problem

specifications and evaluation of the net models in a trial-and-error way. So, once again we have to emphasize our conclusion: "a problem is as much hard as much its model is sophisticated".

This observation support many different, available models following the Petri nets concept. So, besides fundamental classes of nets theory such as Condition/Event systems and Place/Transition systems, their subclasses, e.g. State Machines, Marked Graphs, Free-Choice nets, or their extensions, e.g. Inhibitor Petri nets, Predicate/Transition Petri nets, Timed Petri nets, Colored Petri nets, Stochastic Petri nets and so on, can be employed as a modeling tool. However, that is a matter of a compromise observed between the modeling power and decision power of the modeling framework.

It can be noted that extending the Petri net model usually increases the modeling power, however it may decrease their decision power. Also the subclasses of Petri nets might have good decision properties, however their modeling power may be decreased. So, an impact of any net model extension on its decision power must be recognized and evaluated from the specific of problem considered.

3. THE TURING MACHINE PERSPECTIVE

Talking about Turing machine let us recall the statement: Any algorithm can be expressed in terms of the relevant Turing machine. In order to show it let us return once again to the Petri net model shown in Fig. 7 The behavior of the modeled manufacturing cell has been encompassed by reachability digraph shown in Fig 8.

The same behavior can be encoded in the Turing machine shown in Table 1. The markings placed in the first column named States belong to so called reachability set of Petri net model considered. Other potentially possible markings are not included because they cannot enable any transition from considered set. The rest of markings are obtained due to the following state transition rule:

IF "the transition t_{Ri} is enabled at the initial marking M_j" **THEN** "the value of successive marking follows the production rule $R_{i''}$" **ELSE** "the value of successive marking equals to the initial one".

Table 1. Turing machine encompassing the behavior of the Petri net model from Fig. 7

Transitions / States M	t_{R1}	t_{R2}	t_{R3}	t_{R4}	t_{R5}	t_{R6}
0,0,0,0,1,0,1,1	0,0,0,0,1,0,1,1	0,0,0,0,1,0,1,1	0,0,0,0,1,0,1,1	0,0,0,0,1,0,1,1	0,0,0,0,1,0,1,1	0,0,0,0,0,1,1,1
0,0,0,1,0,1,0,1	1,0,0,1,0,0,0,1	0,0,0,1,0,1,0,1	0,0,0,1,0,1,0,1	0,0,0,1,0,1,0,1	0,0,0,0,1,0,1,1	0,0,0,1,0,1,0,1
0,0,1,0,0,1,1,0	**1,0,1,0,0,0,1,0**	0,0,1,0,0,1,1,0	0,0,1,0,0,1,1,0	0,0,0,1,0,1,0,1	0,0,1,0,0,1,1,0	0,0,1,0,0,1,1,0
0,0,0,0,0,1,1,1	1,0,0,0,0,0,1,1	0,0,0,0,0,1,1,1	0,0,0,0,0,1,1,1	0,0,0,0,0,1,1,1	0,0,0,0,0,1,1,1	0,0,0,0,0,1,1,1
1,0,0,0,0,0,1,1	1,0,0,0,0,0,1,1	0,1,0,0,0,1,0,1	1,0,0,0,0,0,1,1	1,0,0,0,0,0,1,1	1,0,0,0,0,0,1,1	1,0,0,0,0,0,1,1
0,1,0,0,0,1,0,1	**1,1,0,0,0,0,0,1**	1,0,0,0,0,0,1,1	0,0,1,0,0,1,1,0	0,1,0,0,0,1,0,1	0,1,0,0,0,1,0,1	0,1,0,0,0,1,0,1
1,1,0,0,0,0,0,1	1,1,0,0,0,0,0,1	1,1,0,0,0,0,0,1	**1,0,1,0,0,0,1,0**	1,0,0,1,0,0,0,1	1,1,0,0,0,0,0,1	1,1,0,0,0,0,0,1
1,0,1,0,0,0,1,0	1,0,1,0,0,0,1,0	**0,1,1,0,0,1,0,0**	1,0,1,0,0,0,1,0	1,0,1,0,0,0,1,0	1,0,1,0,0,0,1,0	1,0,1,0,0,0,1,0
0,1,1,0,0,1,0,0	1,1,1,0,0,0,0,0	0,1,1,0,0,1,0,0	0,1,1,0,0,1,0,0	0,1,1,0,0,1,0,0	0,1,1,0,0,1,0,0	0,1,1,0,0,1,0,0

By the grey shade the successive markings leading to the deadlock are distinguished. Underlined states are the deadlock ones, and then by the bold figures the admissible successive markings are distinguished. The rest of the next state markings are the same as initiating those ones.

Let us remind you also, that the problem we were faced with in the example considered was to find out the control procedure guaranteeing the asynchronously acting components of the manufacturing cell will be supervised in a way guaranteeing its deadlock and starvation free functioning. In broader sense, this issue regards the methods aimed at automatic synthesis of real-time concurrent control programs, i.e., methods that enable the synthesis of assumed quality control program on the basis of the given specification of the processes controlled. So, since the Petri net model of such procedure does not guarantee the required functioning, hence its correction aimed at preserving of the assumed quality behavior has to be adjusted.

For illustration of the possible solutions let us consider two cases: in the first one the model modification regards of its structure correction, and in the second one a new model created on the base of extended Petri nets is created.

In the first case shown in Fig. 9 a control place pad is added.

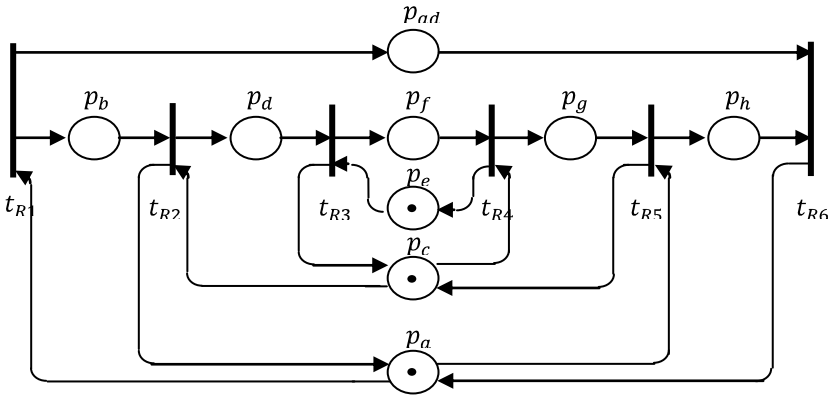


Fig. 9 Modified Petri net model of deadlock-free control procedure.

The reachability set of the new Petri net model does not contain any deadlock marking. The relevant Turing machine is specified by Table 2 , where markings are defined as follows:

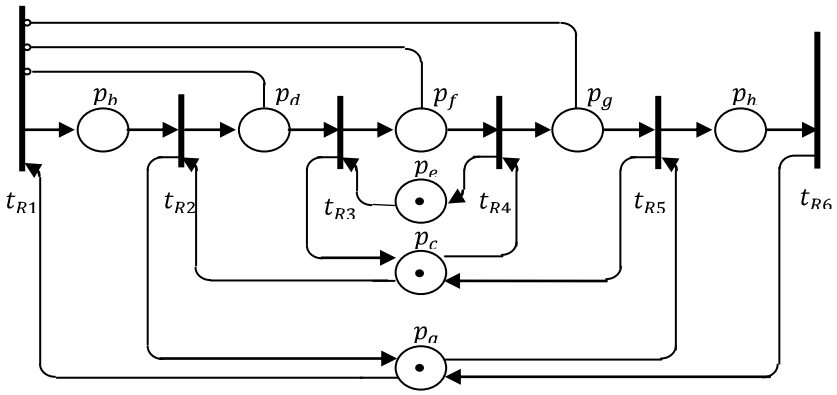
$$p_b\ p_d\ p_r\ p_g\ p_h\ p_a\ p_c\ p_e\ p_{ad}$$
$$M = (1,0,0,1,0,0,0,1,1)$$

Table 2. Turing machine encompassing the behavior of the Petri net model from Fig. 9.

Transitions / **States M**	t_{R1}	t_{R2}	t_{R3}	t_{R4}	t_{R5}	t_{R6}
0,0,0,0,1,0,1,1,1	0,0,0,0,1,0,1,1,1	0,0,0,0,1,0,1,1,1	0,0,0,0,1,0,1,1,1	0,0,0,0,1,0,1,1,1	0,0,0,0,1,0,1,1,1	**0,0,0,0,0,1,1,1,0**
0,0,0,1,0,1,0,1,1	0,0,0,1,0,1,0,1,1	0,0,0,1,0,1,0,1,1	0,0,0,1,0,1,0,1,1	0,0,0,1,0,1,0,1,1	**0,0,0,0,1,0,1,1,1**	0,0,0,1,0,1,0,1,1
0,0,1,0,0,1,1,0,1	0,0,1,0,0,1,1,0,1	0,0,1,0,0,1,1,0,1	0,0,1,0,0,1,1,0,1	**0,0,0,1,0,1,0,1,1**	0,0,1,0,0,1,1,0,1	0,0,1,0,0,1,1,0,1
0,0,0,0,0,1,1,1,0	**1,0,0,0,0,0,1,1,1**	0,0,0,0,0,1,1,1,0	0,0,0,0,0,1,1,1,0	0,0,0,0,0,1,1,1,0	0,0,0,0,0,1,1,1,0	0,0,0,0,0,1,1,1,0
1,0,0,0,0,0,1,1,1	1,0,0,0,0,0,1,1,1	**0,1,0,0,0,1,0,1,1**	1,0,0,0,0,0,1,1,1	1,0,0,0,0,0,1,1,1	1,0,0,0,0,0,1,1,1	1,0,0,0,0,0,1,1,1
0,1,0,0,0,1,0,1,1	0,1,0,0,0,1,0,1,1	0,1,0,0,0,1,0,1,1	**0,0,1,0,0,1,1,0,1**	0,1,0,0,0,1,0,1,1	0,1,0,0,0,1,0,1,1	0,1,0,0,0,1,0,1,1

By the gray shade the admissible successive states are distinguished. The rest of the successive markings are the same as initiating those ones.

In the second case an extended Petri net with inhibitor arcs is used. The firing rule applied in this kind of nets enables a transition to fire when it is enabled according to the earlier



mentioned firing rule and simultaneously if the input places linked to the transition by inhibitor arcs have no tokens. The relevant solution is shown in Fig. 10.

Legend:

- inhibitor arcs constraining the transition firing in case at least one of them is linked with the marked place

Fig. 10. Petri net with inhibitor arcs model of deadlock-free control procedure.

The reachability set of the new Petri net model does not contain any deadlock marking. The relevant Turing machine is specified by Table 3

Table 3. Turing machine encompassing the behavior of the Petri net model from Fig. 10

Transitions / **States M**	t_{R1}	t_{R2}	t_{R3}	t_{R4}	t_{R5}	t_{R6}
0,0,0,0,1,0,1,1	0,0,0,0,1,0,1,1	0,0,0,0,1,0,1,1	0,0,0,0,1,0,1,1	0,0,0,0,1,0,1,1	0,0,0,0,1,0,1,1	**0,0,0,0,0,1,1,1**
0,0,0,1,0,1,0,1	0,0,0,1,0,1,0,1	0,0,0,1,0,1,0,1	0,0,0,1,0,1,0,1	0,0,0,1,0,1,0,1	**0,0,0,0,1,0,1,1**	0,0,0,1,0,1,0,1
0,0,1,0,0,1,1,0	0,0,1,0,0,1,1,0	0,0,1,0,0,1,1,0	0,0,1,0,0,1,1,0	**0,0,0,1,0,1,0,1**	0,0,1,0,0,1,1,0	0,0,1,0,0,1,1,0
0,0,0,0,0,1,1,1	**1,0,0,0,0,0,1,1**	0,0,0,0,0,1,1,1	0,0,0,0,0,1,1,1	0,0,0,0,0,1,1,1	0,0,0,0,0,1,1,1	0,0,0,0,0,1,1,1
1,0,0,0,0,0,1,1	1,0,0,0,0,0,1,1	**0,1,0,0,0,1,0,1**	1,0,0,0,0,0,1,1	1,0,0,0,0,0,1,1	1,0,0,0,0,0,1,1	1,0,0,0,0,0,1,1
0,1,0,0,0,1,0,1	0,1,0,0,0,1,0,1	0,1,0,0,0,1,0,1	**0,0,1,0,0,1,1,0**	0,1,0,0,0,1,0,1	0,1,0,0,0,1,0,1	0,1,0,0,0,1,0,1

The succeeding markings are obtained due to the following state transition rule:

IF "the transition t_{Ri} is enabled at the initial marking M_j and for every inhibitor arcs $\forall\,(p, t_{Ri}) \in I$ the following condition holds $M(p) = 0$" **THEN** "the value of successive marking follows the production rule R_i" **ELSE** "the value of successive marking equals to the initial one".

Note that this solution provides the same set of reachable markings in cost of more complex rule for transition firing, while taking advantage in smaller set of places. So, the basic model modification itself is connected with difficulties arising from the complexity of the reachability graph analysis and the heuristic character of the available techniques for its modification. Simply speaking that means the problem of automatic programming still belongs to the open ones.

4. CONCLUSIONS

The supervisory control of large-scale compound discrete event systems involves computations on state spaces which grow exponentially with the number of system components. Programming languages allowing describing parallel processes are widely used nowadays, especially many general purpose and task oriented Petri net-based packages, e.g. for the specification and verification of concurrent control problems, have been developed so far. However problems of analysis and verification of models of concurrently flowing processes are far from being solved satisfactorily [1,4].

Therefore, although the cases have been discusses were simplified in scope, they provide an insight into the needs for methods aimed at real-time concurrent control programming, i.e. guaranteeing formal verification of designed control procedure while reducing state space generation/searching efforts.

REFERENCES

[1] BANASZAK Z. (Ed.): *Modelling and control of FMS. Petri net approach*. Wrocław Technical University Press, Wrocław 1991.

[2] BANASZAK Z.: *Modeling of manufacturing systems*. In: Modern Manufacturing, M.B. Zaremba B. Prasad (Eds) Springer-Verlag, London 1994, pp.253-286.

[3] KARATKIEVICH A.: *Dynamic analysis of Petri net-based discrete systems*. Springer Verlag, Berlin 2007.

[4] PERK S.: *Hierarchical Design of Discrete Event Controllers*: An Automated Manufacturing System Case Study http://www.rt.eei.uni-erlangen.de/FGdes/diplomarbeit2004_perk.pdf

[5] REISIG W., PETRI nets: *An Introduction*. Springer Verlag, Berlin 1985.

[6] SCHMIDT K., MOOR T., PERK S.: *Nonblocking Hierarchical Control of Decentralized Discrete Event Systems*. IEEE Transactions on Automatic Control, Vol. 53, No. 10, November 2008, pp. 2252-2265.

[7] VISWANDADHAM N., NARAHARI Y.: *Performance modeling of automated manufacturing systems*. Prentice Hall, New Jersey 1992

[8] http://www.rt.eei.uni-erlangen.de/FGdes/publications.html