Control
and Cybernetics

VOL. 17 (1988) No. 4

Petri net approach to automatic real-time program synthesis[1)]

by

ZBIGNIEW BANASZAK
MOWAFAK HASSAN ABDUL-HUSSIN

Institute of Technical Cybernetics
Technical University of Wroclaw
Wrocław, Poland

This paper deals with a Petri net approach to the automatic design of control programs which are aimed at supervision of concurrent, pipeline-like flowing processes observed in large-variety, small-lot production systems. Its objective lies in the formal investigation of the conditions necessary for the design of a class of deadlock-free Petri nets. The nets considered, while modelling an admissible controls of processes flow, serve as control-programs representations. The results obtained allow for automatic conversion of a process specification, via a net model of a control flow, into the relevant control program.

KEYWORDS: Parallel processing flow control, Petri net, real-time programming, control systems synthesis.

1. Introduction

Problems arising in the field of the control system design for concurrently acting processes create a growing need for automatic tools supporting their programming. To achieve their objectives, the system designers use computer-assisted analysis and design methods based on the high-level programming languages [6, 7].

The design process of a software system consists of the following two stages [1]:

— formal specification of the functions of the system,
— proof of the correctness of this specification.

[1)] This research was supported by CPBP 02.13 grant BPL-IV-03/1837/85 "AI systems for mechanical shovels and vehicles".

While existing packages allow automation of the evaluation of derived control programs, for example by means of computer simulation, there is a lack of techniques allowing automatic program design. In the sequel, the problem of automatic software production lies in the automatic inversion of an input problem into a computer program suitable for the intended application. Its solution will help to reduce the time and effort involved in software completion as well as releasing programmers from debugging and from the program correctness examination.

As both the correctness and the performance are critical issues to be taken into account in automatic real-time program synthesis, an appropriate representation is needed. Petri nets have been proven to be convenient and multilevel applicable tools for the specification as well as for the verification of complex distributed industrial control systems. They provide not only a language for the design process but also a theory backing this process. Here, they are applied in the course of the formal investigation of the conditions necessary for the design of a net class respecting such basic dynamic properties as buffer overflow and deadlock avoidance. Conditions developed enable us to prove the correctness of an algorithm aimed at the control program synthesis. The considered algorithm transforms the given process specification into the net model describing the dynamics of the process flow.

The class of concurrent, pipeline-like flowing processes to be controlled is considered. It is assumed that system components are shared among different, concurrently flowing processes and that the process operations are executed asynchronously.

The composition of this paper is as follows. First, in section 2, we give basic definitions and present the formalism employed. Next, in section 3, we state the main problem, and then, in section 4, we present the modelling concept applied. In section 5, the conditions essential for the algorithm design, presented in the succeeding section 6, are provided. A short review of possible applications of our results is contained in section 7.

2. Formalism employed

A class of simple, self-loop free and safe place/transition nets is considered.

DEFINITION 1. We define simple, self-loop free and safe place-transition net (PT-net for short) by a quadruple $PN = (P, T, E, M_0)$, where P and T are finite set of places and transitions, respectively, such that $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$, $E \subset (P \times T) \cup (T \times P)$ is a flow relation, such that $\mathrm{dom}\,(E) \cup \cup \mathrm{cod}\,(E) = P \cup T$, $M_0 : P \to \{0, 1\}$ is an initial marking.

Graphically, a Petri net may be represented as a directed graph having two types of nodes: places (represented by circles) and transitions (represented by bars). These nodes, places and transitions are connected by directed arcs from places to transitions and from transitions to places.

In order to represent the net structure, the following description of the incidence matrix is used:

DEFINITION 2. Let $PN = (P, T, E, M)$, $n = \|P\|$, $m = \|T\|$, be a PT-net. A matrix C size of $m \times n$ is said to be an incidence matrix of PN if:

$$c_{ij} = \begin{cases} 1 & \text{if} \quad p_j \in t_i \\ -1 & \text{if} \quad p_j \in {}^{\cdot}t_i \\ 0 & \text{otherwise}, \end{cases}$$

while ${}^{\cdot}t = \{p | (p, t) \in E\}$ $(t^{\cdot} = \{p | (t, p) \in E\})$ is called a set of the input (resp. output) places of the transition t. The dynamic properties of a Petri net are represented by the position and movement of tokens (indicated by black dots) in the places of the net. The arrangement of tokens in a Petri net defines the state of the net and is called its marking. Every net is provided with an initial marking M_0. The marking may change as a result of the firing of a transition by the following execution rule:

1. Each transition which has one token in each of its input places and has not any token in each of its output places is enabled.

2. Any enabled transition may be chosen to fire.

3. The act of firing removes a token from each input places and puts a token in each output place.

The formal definitions are as follows:

DEFINITION 3. A transition $t \in T$ is enabled at M if $(\forall p \in {}^{\cdot}t)(M(p) = 1)$ and $(\forall p \in t^{\cdot})(M(p) = 0)$.

DEFINITION 4. The next-state function $\delta: \{0, 1\}^n \times T \to \{0, 1\}^n$ for a $PN = (P, T, E, M_0)$, $n = \|P\|$, with marking M and transition $t \in T$ is defined if and only if t is enabled at M. If $\delta(M, t)$ is defined, then $\delta(M, t) = M'$, where

$$M'(p) = \begin{cases} M(p) - 1 & \text{if} \quad p \in {}^{\cdot}t \\ M(p) + 1 & \text{if} \quad p \in t^{\cdot} \\ M(p) & \text{if} \quad p \in {}^{\cdot}t \cup t^{\cdot}. \end{cases}$$

Marking of a place represents the holding of a condition. The firing of a transition represents the occurrence of an event, which ends a set of holdings and begins a set of new holdings. Because of the space limitation definitions of the extended next-state function δ^*, of the firing sequence σ, and of the reachability set of markings $R(C, M_0)$, are omitted. They can be found in [8, 9].

Now, let us recall the definition of the net's liveness property playing the essential role in further considerations.

DEFINITION 5. A $PN = (P, T, E, M_0)$ is a live PT-net if $(\forall t \in T)(\forall M \in \in R(C, M_0))(\exists M' \in R(C, M))$ (t is enabled at M').

Besides the above well-known definitions, let us introduce new ones essential in further considerations.

DEFINITION 6. Consider $PN = (P, T, E, M_0)$ being a PT-net. $PN = (P, T, E, M_0)$ such that the following conditions hold is said to be a pipeline place/ /transition net (PPT-net for short).

(i) $T = \{t_i | i = \overline{1, m}\}$, $P = P^{\mathrm{I}} \cup P^{\mathrm{II}}$, $P^{\mathrm{I}} \cap P^{\mathrm{II}} = \emptyset$,

(ii) $P^{\mathrm{I}} = \{p_i | i = \overline{1, m-1}\}$,
$(\forall p_i \in P^{\mathrm{I}})(\exists ! t_i \in T)(\exists ! t_{i+1} \in T)(\dot{p}_i = \{t_i\} \ \& \ p_i^{\cdot} = \{t_{i+1}\})$,

(iii) $(\forall p_i \in P^{\mathrm{II}})(\exists t_j, t_{j+1} \in T)(t_j \in \dot{p}_i \ \& \ t_{j+1} \in p_i^{\cdot})$,
$(\forall t_i, t_{i+1} \in T)(\exists ! p_j \in P^{\mathrm{II}})(p_j \in t_i^{\cdot} \ \& \ p_j \in \dot{t}_{i+1})$,

(iv) M_0 is $(1 \times n)$ vector, where $n = \|P\|$ such that
$(\forall p \in P)(M_0(p) = 0)$,

where $p^{\cdot} = \{t | (p, t) \in E\}$ ($\dot{p} = \{t | (t, p) \in E\}$) — is a set of output (resp. input) transitions of a place p.

ASSUMPTION 1

Let $PN = (P, T, E, M_0)$ consist of a set $\{{}^kPN = ({}^kP, {}^kT, {}^kE, {}^kM_0) | k = \overline{1, v}\}$ of PPT-nets, such that the following conditions hold.

(i) $P = \bigcup_{i=\overline{1,v}} {}^iP$, $P = P^{\mathrm{I}} \cup P^{\mathrm{II}}$, $P^{\mathrm{I}} \cap P^{\mathrm{II}} = \emptyset$,

$P^{\mathrm{I}} = \bigcup_{i=\overline{1,v}} {}^iP^{\mathrm{I}}$, $P^{\mathrm{II}} = \bigcup_{i=\overline{1,v}} {}^iP^{\mathrm{II}}$,

$(\forall i = \overline{1, v})(\exists j = \overline{1, v})(i \neq j \ \& \ {}^iP^{\mathrm{II}} \cap {}^jP^{\mathrm{II}} \neq \emptyset)$,

(ii) $T = \bigcup_{i=\overline{1,v}} {}^iT$,

$(\forall i, j = \overline{1, v})(i \neq j \Rightarrow {}^iT \cap {}^jT = \emptyset)$,

(iii) $E = \bigcup_{i=\overline{1,v}} {}^iE$,

(iv) M_0 is a such $(1 \times n)$ vector that $(\forall p \in P)(M_0(p) = 0)$.

DEFINITION 7. A $PN = (C, M_0)$ satisfying Assumption 1 is said to be open queueing PT-net (QPT-net for short).

REMARK 1. From the above definition it follows that each PN being an element of the class of PPT-nets contains one source and one sink, and satisfies the following condition: $\delta(M_0, \sigma) = M_0$, where $\sigma = t_1\, t_2 \ldots t_i \ldots t_m$.

REMARK 2. For each PN being an element of the class of QPT-nets such that $PN = \{{}^kPN | k = \overline{1, v} \;\&\; {}^kPN \in PPT\}$ there exists $\sigma = \sigma_1\, \sigma_2 \ldots \sigma_j \ldots \sigma_v$, $\sigma_j = t_{l+1}\, t_{l+2} \ldots t_{l+r_j}$, $l = \sum_{i=1}^{j-1} r_i$, r_i — is a length of firing sequence σ_i, such that $\delta(M_0, \sigma) = M_0$.

REMARK 3. In the rest of the paper, in order to model the interprocesses cooperation, we use the following interpretation for places, transitions and tokens:

(i) Places represent operations or resources (machines).

(ii) If a place represents the resource, a token in it stands for a workpiece and if a place represents operation, a token in it represents the machines' operation being actually performed.

(iii) Transitions represent events reflecting changes of operations actually performed, i.e. transitions from one operation to the immediately following one or resources activity changes.

3. Problem statement

Consider a production system consisting of a finite number of machines which may function asynchronously. Machines are used to process the jobs, and every job can be processed on at most one machine at a time. Some machines may be shared among different routes. Each job is associated with a fixed sequence of machines; i.e. the production route it has to pass. Jobs cannot be divided and no more then one job may be processed on one machine at a given time. There is a finite set of routes along which different kinds of jobs are simultaneously processed. Along each route the pipeline-like flow of jobs is processed.

The control problem can be now formulated. Flows of jobs respect fixed sequence of the machines on which the processes are executed. No job can be moved to a machine with full entry store place. Machines act asynchronously, and the process performance is deadlock-free. Our task is then to design an algorithm transforming the given process specification; e.g. a set of production routes and buffers capacity constraints, into the related net model of its control flow.

In other words, the problem considered is two-fold. It contains a problem of processes modelling as well as a problem of extension of the obtained process models to the models of the corresponding deadlock-free control flows. So, the main task is to find a synchronization mechanism aimed at deadlock avoidance.

In order to illustrate its importance let us consider the machining cell as shown in Fig. 1. The cell consists of two machine tools M_1, M_2 and three industrial robots R_1, R_2 and R_3. Each machine is provided with a buffer consisting of one entry store place and one exit store place for a job to be processed next, and for a job having been processed waiting for the transportation, respectively. The robots are used to move the workpiece as follows from the input store I to the machine entry store places (robot R_1), from the exit store place of one machine to the entry store place of another one (robot R_2) as well as from exit store place of machines to the output store 0 (robot R_3). Workpieces can be transported and/or machined concurrently. The corresponding production routes are as follows: $MR_1 = R_1, M_1, R_2, M_2, R_3$; $MR_2 = R_1, M_2, R_2, M_1, R_3$. It is assumed that workpieces arrive in the cell randomly, and all components of the cell perform their tasks asynchronously. Following the above assumption, it can be easily examined that deadlocks may occur. One such situation may arise when all machines, including their store places are busy with jobs and a job waiting at the exit store place of machine M_1 waits for the releasing of the entry store place of machine M_2, while the job waiting at the exit store place of machine M_2 waits for the releasing of the entry store place of machine M_1.

4. Modelling of process interactions

All our further considerations will be restricted to the class of concurrent, pipeline-like flowing processes being executed on the common set of system components. Their objective lies in the statement of the models specification, and then, in the exploration (on the provided model base) of synchronization mechanisms considering only the deadlock-free variants of process cooperation.

The modelling of the control flow can be done using the Petri net representation. The problem we are facing now is how to find an algorithm transforming a given process specification into the relevant Petri net model. It means that the algorithm, we are looking for needs to be able to transform the given set $\{MR_i | i = \overline{1, v}\}$, where $MR_i = M_{j_1}, M_{j_2}, \ldots, M_{j_i}, \ldots, M_{j_{k_i}}$, $M_{j_i} \in \underline{M}$, $\underline{M}$ — is a set of system robots and machines, into a net model belonging to the class of *QPT*-nets. This problem can be solved for different demands concerning of a modelling abstraction level, i.e. assuming an existence of the one buffer for each machine tool, omitting from considerations working zones of machines and so on. The algorithm descriptions as well as some details can be found in [4, 5].

Here is an example of the algorithm implementation. Let us consider the machining cell shown in Fig. 1. The net model of the control flow (distinguished by continuous lines) is given in Fig. 2. In the model considered, all the transitions are interpreted as events reflecting either beginning, i.e. t_1, t_{11}, or ending, i.e. t_{10}, t_{20}, or changing, i.e. t_2-t_9, $t_{12}-t_{19}$, of operations. Places p_1-p_{18} are interpreted as operations occuring along the production routes. Places $p_{19}-p_{27}$ represent operations involved in transportations, i.e. p_{19}, p_{20}, p_{21}, and machining, i.e. p_{22}, p_{23}, and workpiece storing in machines buffers, i.e. p_{24}, p_{25}, p_{26}, p_{27}, as well. The order of storing operations, corresponding to $p_{19}+p_{27}$, occurring in the production routes is depicted by the labelled arcs in Fig. 1.

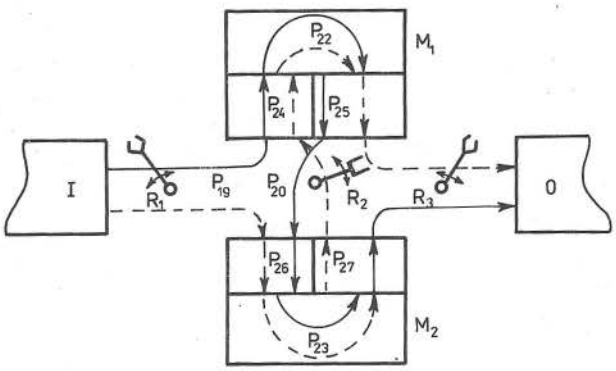

Fig. 1. Machining cell

For instance the state M, such that $M(p_2) = M(p_3) = M(p_{24}) = M(p_{22}) = 1$ and $(\forall p \in P \setminus \{p_2, p_3, p_{22}, p_{24}\})(M(p) = 0)$, is interpreted as follows. A job which has arrived first in the cell is machined on the machine M_1 while the next job is placed in the entry store place of the machine M_1 buffer. Therefore, the deadlock situation already mentioned can be interpreted as the state $M' \in R(PN, M_0)$ such that $M' = (01110000001110000000101111)$.

It should be pointed out, that in the case considered, deadlocks occur as direct consequence of constraints assumed for the stores capacity. There are two ways leading to the solution of the deadlock avoidance problem. The first method, taking into account imposed constraints, involves the algorithms aimed at searching of appropriate synchronization mechanisms. The problem related is an *NP*-complete one. The objective of the second approach is to find a required control flow encompassing some admissible realizations of the processes performance. Such a problem has a polynomial complexity. The latter case is the subject of our further considerations.

5. Principles of competing processes synchronization

In order to obtain a net model of the deadlock-free control flow some modifications need to be introduced to the relevant net model of interacting processes, i.e. to the model encompassing all interacting processes realizations. The modification considered should take into account preassumed buffers capacity constraints, while preserving the net's liveness property as well as the occurrence of a predetermined firing sequence.

Note that to avoid deadlocks it is enough to remove from the reachability set $R(C, M_0)$ all the states M such that $M_0 \notin R(C, M)$. This means that a QPT-net model can be properly modified (for instance using inhibitor arcs) so to meet the deadlock avoidance requirements.

Consider a PN belonging to the class of OPT-nets. Note that with each ${}^kPN$ (being a PPT-net) belonging to the QPT-net the relevant sequence ${}^kSQ = p_{j_1}, p_{j_2}, \ldots, p_{j_i}, \ldots, p_{j_{r_k}}$ is associated, where $p_{j_i} \in {}^kP^{\text{II}}$.

In each ${}^kSQ$ two kinds of alternately following subsequences are distinguished. Subsequences belonging to the first class contain the components uniquely occurring in ${}^kSQ$, $k = \overline{1, v}$, as well as the subsequences belonging to the second class which only contain the repeating (not uniquely occurring) components. Elements of the above classes are marked by ${}^kQU(l)$, ${}^kQE(l)$ respectively, where l denote the succeeding number of the given subsequence type in ${}^kSQ$. Note that with each set ${}^k\tilde{Q}E(l) = \{crd^i\, {}^kQE(l) | i = \overline{1, |{}^kQE|}\}$, where $|{}^kQE(l)|$ — stands for the length of ${}^kQE(l)$, $\underset{\sim}{\text{crd}}^i S = s_i$, $S = (s_1, s_2, \ldots, s_i, \ldots, s_n)$, the relevant set ${}^kTE(l) = \{t | t \in {}^{\cdot}p \;\&\; p \in {}^kQE(l)\}$ is associated. Using the above notations, the following theorem can be proved.

THEOREM 1. *Let $PN = (P, T, E, M_0)$ be a QPT-net. A $PN' = (P, T, E \cup E', M_0)$ such that the following condition holds is a live PT-net.* $E' = \underset{\sim}{=} \{[p, t] | t \in {}^kTE(l) \;\&\; p \in {}^kQE(l) \cup {}^kQU(l) \;\&\; p \in t^{\cdot} \cup {}^{\cdot}t \;\&\; k = \overline{1, v}\}$, *where* ${}^kQU(l) = \{crd^i\, {}^kQU(l) | i = \overline{1, |{}^kQU(l)|}\}$, $[p, t] \in P \times T$, *l — the number of the succeeding subsequence belonging to the first or second class.*

Proof. For the proof, we need to show that $(\forall M \in R(PN', M_0))(\exists t \in T)$ (t is enabled at M) and $(\exists \sigma \in T^*)\left(\delta(\delta(M, t), \sigma) = M_0\right)$ hold. Let us assume that $(\exists M \in R(PN', M_0))(\forall t \in T)$ (t is not enabled at M) holds. Note, that for the deadlock state M there exists the set $D \subset \{P | P \in Z\}$, where

$Z = \{p | p \in \{crd^j\, {}^kSQ | i = \overline{1, |{}^kSQ|}\} \;\&\; k = \overline{1, v}\}$, such that $(\forall p \in D)(M(p) = 1)$. Also, it should be pointed out that no one place from D cannot be released without another place belonging to D overflowing.

Consider the state M' and $t' \in T$ such that $\delta(M', t) = M$. Note that there exists $p' \in t'$, such that $M'(p') = 0$. At the state M there exists t such that $p' \in {}^{\cdot}t$ and $p'' \in t^{\cdot}$ where p', $p'' \in D$. Because t' is enabled at M', i.e. the

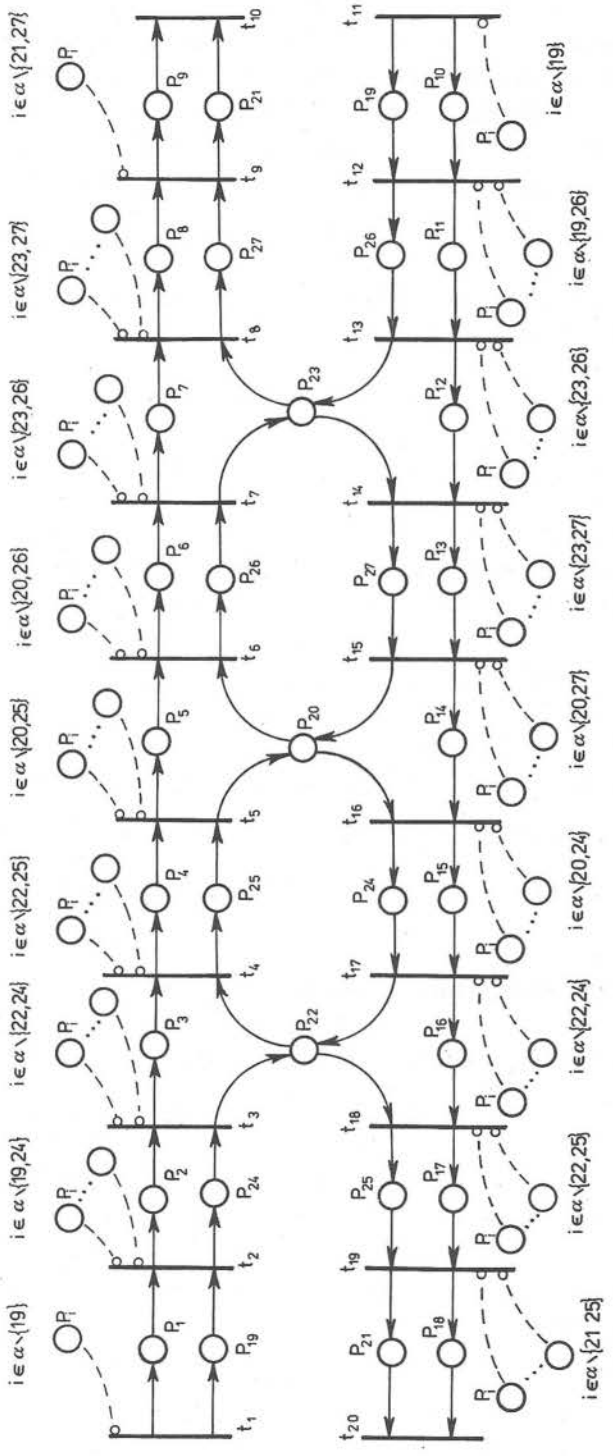

Fig. 2. Petri net model of process flow

following conditions hold $(\forall p \in \dot{t})(M'(p) = 1)$, $(\forall p \in t^{\cdot})(M'(p) = 0)$, $(\forall p \in \mathring{t})$ $(M'(p) = 0)$, where $\mathring{t} = \{p | [p, t] \in E'\}$, then according to the conditions determining the set E' design, $M'(p'') = 0$ holds. Since the firing of transition t' leads to $M(p'') = 0$ hence $p'' \notin D$. According to the occurring contradiction $(\forall M \in R(PN', M_0))(\exists t \in T)$ (t is enabled at M) holds.

Note that for each $PN = \{{}^k PN | k = \overline{1, v} \,\&\, {}^k PN \in PPT\}$ there exists $\sigma^* = \sigma_1 \sigma_2 \ldots \sigma_j \ldots \sigma_v$, $\sigma_j = t_{l+1} t_{l+2} \ldots t_{i+r_j}$, $l = \sum_{i=1}^{j-1} r_i$, r_i — is a length of firing sequence σ_i such that $\delta(M_0, \sigma^*) = M_0$. Thus because $R(PN, M_0)$ is finite there exists $\sigma \in T^*$ such that $\delta(M, \sigma) = M$.

Q.E.D. ■

The above theorem provides the conditions for avoiding the deadlock occurrence in QPT-nets.

For an illustration see Fig. 2, where the inhibitor arcs are distinguished by dashed lines.

From the analysis of the reachability tree corresponding to the net considered, i.e. $R(C', M_0)$, it can be easily found that $R(C', M_0) \subset R(C, M_0)$, i.e. besides deadlock states some safe states, for instance M such that

$$M(p_i) = \begin{cases} 1 & \text{for} \quad i \in H, \; H = \{3, 4, 20, 22, 25\}, \\ 0 & \text{for} \quad i \in \{1, 2, \ldots, 27\} \backslash H \end{cases}$$

are eliminated from $R(C, M_0)$.

It means that the algorithm considered, having its computational complexity determined by $O(n^2)$, where $n = \sum_{k=1}^{v} |{}^k SQ|$, $|{}^k SQ|$ — the length of ${}^k SQ$, provides a net model encompassing some subset of an all safe state set.

6. Algorithm for control flow model design

The algorithm transforming a set of production routes $PS = \{MR_i | i = \overline{1, v}\}$, $MR_i = M_{j_1}, M_{j_2}, \ldots, M_{j_i}, \ldots, M_{j_r}$, $M_{j_i} \in M$, into $PN = (C, M_0)$ being a net model of the control flow consists of three steps. As input data an auxiliary set $RB = \{BS_i | i = \overline{1, v}\}$, $BS_i = B_{i_1}, B_{i_2}, \ldots, B_{i_j}, \ldots, B_{i_r}$, $B_{i_j} \in B$ — is interpreted as a robot or a machine tool or as machine tool buffer, is considered.

STEP 1

For each BS_i set the matrix C^i of size $m_i \times n_i$, where $m_i = |BS_i| + 1$, $n_i = |BS_i|$, such that

$$c^i_{kl} = \begin{cases} 1 & \text{if} \quad k = l \\ -1 & \text{if} \quad k - 1 = l. \\ 0 & \text{otherwise} \end{cases}$$

Set the matrix $\bar{C}$ of size $m \times n'$, where $m = \sum_{i=1}^{v} m_i$, $n' = \sum_{i=1}^{v} n_i$, such that

$$\bar{c}_{ij} = \begin{cases} c^r_{kl} & \text{if} \quad i = \sum_{w=1}^{r-1} m_w + k, \; j = \sum_{w=1}^{r-1} n_w + l, \\ 0 & \text{otherwise.} \end{cases}$$

STEP 2

Set the matrix $\hat{C} = \hat{C}^1 \hat{C}^2 \ldots \hat{C}^i \ldots \hat{C}^{n'}$ of size $m \times n''$, where $n'' = \|F\|$, $F = \bigcup_{l=\overline{1,v}} \{crd^k BS_l | k = \overline{1, |BS_l|}\}$, such that each column vector $\hat{C}^i$ follows

$$\hat{c}^i_k = \begin{cases} 1 \text{ for } k = \sum_{l=1}^{w-1} |BS_l| + w - 1 + j, \text{ if } B_i = crd^j BS_w, w = \overline{1, v}, \\ -1 \text{ for } k = \sum_{l=1}^{w-1} |BS_l| + w - 1 + j + 1, \text{ if } B_i = crd^j BS_w, w = \overline{1, v}, \\ 0 \text{ otherwise.} \end{cases}$$

STEP 3

On the basis of the matrix $\hat{C}$ set the matrix C^* of size $m \times n''$ such that

$$c^*_{ij} = \begin{cases} 2 & \text{for} \quad [p_j, t_i] \in E' \\ \hat{c}_{ij} & \text{otherwise.} \end{cases}$$

Then set the matrix $\tilde{C} = \bar{C}C^*$ of size $m \times n$, where $n = n' + n''$. The structure of the matrix $\tilde{C}$ obtained according to this algorithm is shown in Fig. 3.

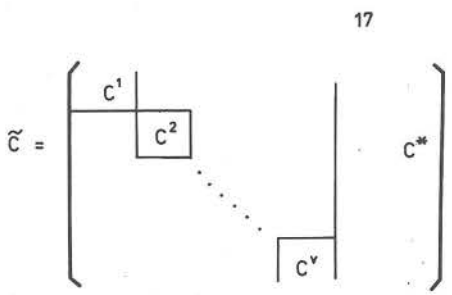

Fig. 3. Structure of the matrix $\tilde{C}$

In order to illustrate an operation of the abovementioned algorithm let us consider the following two production routes $MR_1 = R, M_1, R, M_2, R$, $MR_2 = R, M_2, R, M_1, R$, where R — the industrial robot, M_i — the i-th machine tool. It is assumed that with robot R and each machine M_1, M_2 the relevant buffers B_3 and B_1, B_2 are associated, and to each production route the relevant sequence of buffers is corresponding, i.e. $BS_1 = B_3, B_1, B_3, B_2, B_3$, $BS_2 = B_3, B_2, B_3, B_1, B_3$ respectively. According to

the first step of the algorithm considered the matrix C is designed, on the base of the matrices C^1 and C^2

$$\bar{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix},$$

$$C^1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}, \quad C^2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}.$$

The matrix $\hat{C}$ constructed according to the step 2 has the following form:

$$\hat{C} = \begin{bmatrix} 2 & 2 & 1 \\ 1 & 2 & -1 \\ -1 & 2 & 1 \\ 2 & 1 & -1 \\ 2 & -1 & 1 \\ 2 & 2 & -1 \\ 2 & 2 & 1 \\ 2 & 1 & -1 \\ 2 & -1 & 1 \\ 1 & 2 & -1 \\ -1 & 2 & 1 \\ 2 & 2 & -1 \end{bmatrix}.$$

The resultant matrix C^* provided by step 3 has the following form:

$$C^* = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 1 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & -1 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & 1 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 1 & -1 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 2 & -1 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 2 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 2 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 2 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & -1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & 2 & -1 \end{bmatrix}.$$

Assuming the initial marking M_0 such that $(\forall\, i = \overline{1,n})\,(M(p_i) = 0)$, the net $PN = (C^*, M_0)$ can be treated as the net model of a control flow. Such form is well suited for net behaviour analysis [8] using expression $M' = M + U_t C$ where:

$$c_{ij} = \begin{cases} 0 & \text{if} \quad c^*_{ij} = 2 \\ c^*_{ij} & \text{otherwise} \end{cases}.$$

In consequence, the algorithm provided allows the automatic designing of simulation programs of the modelled systems control flows.

7. Concluding remarks

The presented approach, being our contribution to the automatic programming problem, may be applied to designing computer aided planning systems as well as to designing real-time industrial controllers.

Early results have been implemented [2] in the system aimed at the variety of concurrent processes performance evaluation. The package helps to examine the different dispatching rules, usually used in the course of the synchronization and scheduling of processes. The approach having been introduced here allows us to avoid the laborious and time-consuming programming of admissible synchronization of system component activities.

Other implementation of such an approach to the automatic programming of concurrently flow processes has been presented in [3]. The programming of the real-time controllers should take into account both actual changes of operation processing times and controls ensuring the desirable order of the process performance. Such requirements cause the occurrence of loops and condition branching both in the course of programming and in the examination of the resultant program correctness. The presented approach helps to realize the automatic conversion of the input data

(production routes) into the control procedure of workflows performed along the assumed routings. Automatically obtained procedures (represented in machine-level programming language) satisfy the requirement of deadlock-free performance of asynchronously flowing processes. Also, what may be inferred from the above considerations is that our approach can be successfully applied to the design of adaptive controllers.

However, our considerations concern only sequential processes. Still much research remains to be done towards the generalization of these results to the processes specified by partially ordered sets of operations.

References

[1] BACHMANN P., RICHTER K. Systems analysis and simulation. In: A. Sydow, M. Thoma and R. Vichnevetsky (eds.), Adaptive control of flexible manufacturing systems. Akademie-Verlag, 28 (1985), 290–293.

[2] BANASZAK Z., MAZUR M. Computer aided planning system of concurrent processes in Polish. Proc. of the 1st Nat. Conf. on Robotics. Scientific Papers of the Institute of Technical Cybernetics, Wrocław, Technical University, 27 (1985), 105–116.

[3] BANASZAK Z., MAZUR M. Self programmable controller of concurrent processes. Proc. of the 1st Int. School: MICROCOMPUTERS' 85. Scientific Papers of the Institute of Technical Cybernetics, Wrocław Technical University, 28 (1985), 5–12.

[4] BANASZAK Z. Coordination of concurrent processes: automatic program synthesis. In: R. Trappl (ed.), Cybernetics and Systems'86. D. Reidel, Dordrecht, 1986, pp. 709–716.

[5] BANASZAK Z. Perspectives of automatic real-time program synthesis. In: Real-Time Programming 1986, J. Szlanko (ed.). Pergamon Press, Frankfurt, pp. 19–26.

[6] BOSSY J. C., BRARD P., FAUGERE P., MERLAUD C. Le GRAFCET. Educalivre, Paris 1984.

[7] MARTIN T. Digital computer applications to process control. In: R. Iserman, H. Kaltenecker (eds.), Industrial experience with process control programming language PEARL. Pergamon Press, New York 1981, pp. 505—509.

[8] PETERSON J. L. Petri net theory and the modelling of systems. Prentice-Hall, Englewood Cliffs, New York 1981.

[9] REISIG W. Petrinetze. Springer-Verlag, Berlin 1982.

Received, July 1987.

Metoda sieci Petriego w automatycznej syntezie programów w czasie rzeczywistym

Artykuł dotyczy metody sieci Petriego w automatycznym projektowaniu programów sterowania zorientowanych na nadzorowanie równoległych procesów przepływowych spotykanych w systemach produkcyjnych o krótkich seriach i dużym zróżnicowaniu. Celem artykułu jest zbadanie warunków dostatecznych dla projektowania pewnej klasy sieci Petriego. Otrzymane wyniki pozwalają na automatyczne przetworzenie specyfikacji procesu, poprzez sieciowy model przepływu sterowań na odpowiedni program sterowania.

Метод сети Петри в автоматическом синтезе программ в реальном масштабе времени

Статья касается метода сети Петри в автоматическом проектировании программ управления, ориентированных на дозор параллельных поточных процессов, встречаемых в производетвенных системах с короткими сериями и значительной разнообразностью. Целью статьи является исследование достаточных условий для проектирования некоторого класса сетей Петри. Полученные результаты позволяют автоматически преобразовать спецификацию процесса, посредством сетевой модели потока управлений, в соответствующую программу управления.