UNIWERSYTET TECHNOLOGICZNO-PRZYRODNICZY
IM. JANA I JĘDRZEJA ŚNIADECKICH W BYDGOSZCZY
ZESZYTY NAUKOWE NR 260
TELEKOMUNKACJA I ELEKTRONIKA 15 (2011), 19-35

PETRI NET MODELS OF DISCRETE EVENT SYSTEMS AND STATE SEQUENCE GENERATION FOR CLOSED LOOP PLANT-CONTROLLER SYSTEM

Wiesław Zech[1], Tadeusz Puchałka[2]

[1]University of Technology and Life Sciences
Kaliskiego 7, 85-789 Bydgoszcz, Poland
wieslaw.zech@utp.edu.pl

[2]Poznań University of Technology
Piotrowo 3A, 60-965 Poznań, Poland
tadeusz.puchalka@put.poznan.pl

Summary: In this paper there has been provided a ladder diagram formal model as LD-P/T-system. Closed loop system which consists of the LD-controller model and the controlled object model is constructed. We propose an algorithm to construct a state-transition diagram of the system. A fault is detected when an unpredicted state is generated. Additional benefits from such an approach results in the fact that an abstraction of the transition diagram of this controller is possible, which can be used for LD-VHDL transformation into FPGA implementation.

Keywords: DES, Petri nets, PLC, ladder diagram

1. INTRODUCTION

Examples of Discrete Event System (DES) can be found in automated production. In such a system, programmable logic controllers (PLCs) have been widely used. Programming languages for PLC are standardized (IEC 61131-3 2003). Ladder diagram (LD) is one of them and is very popular. Several approaches have been proposed [5, 7, 12] for fault detection in a sequential control system. In this paper a ladder diagram formal model is provided as LD-P/T-system. Closed loop system which consists of the LD-controller model and controlled object model is constructed. We propose an algorithm to construct a state-transition diagram of the system. A fault is detected when an unpredicted state is generated. Additional benefits from such an approach results in the fact that an abstraction of the transition diagram of this controller is possible, which can be used for LD-VHDL transformation into FPGA implementation [4].

This article is structured as follows: section 2 presents some background on Petri net, and provides formal models: some class of DES as I-P/N-system, B-system, section 3 is devoted to the brief description of LD, in section 4 we define LD-P/T-system and model of closed loop system (controller-plant). Algorithms are printed in the appendix.

2. PETRI NETS

Petri nets are a mathematical and graphical tool that can find application in many fields of science and engineering. Their characteristic feature is the ability of modeling the concurrency. The occurrence actions, under some conditions, is a natural phenomenon due to which Petri nets are perceived as a formal tool for modeling discrete event systems (DES) [6].

Definition 1.[10]
A. Petri net is a 3-tuple PN = (P, T, F), where:
$P = \{p_1, p_2, ..., p_m\}$ is a finite set of places,
$T = \{t_1, t_2, ..., t_n\}$ is a finite set of transitions,
$F: (P \times T) \cup (T \times P) \rightarrow \{0,1\}$ is a flow function
The following conditions are met:
(i) $P \cap T = \emptyset$,
(ii) $P \cup T \neq \emptyset$,
(iii) $\forall x \in P \cup T, \exists y \in P \cup T: F(x, y) \neq 0 \vee F(y, x) \neq 0$.
B. For PN = (P, T, F) $^\bullet t = \{p \in P \mid F(p, t) = 1\}$, $t^\bullet = \{p \in P \mid F(t, p) = 1\}$
are pre-places set, post-places set of $t \in T$, respectively.
$^\bullet p = \{t \in T \mid F(t, p) = 1\}$, $p^\bullet = \{t \in T \mid F(p, t) = 1 \}$
are pre-transitions set, post-transitions set of $p \in P$, respectively.
C. PN = (P, T, F) is simple if $\forall x, y \in P \cup T: ({}^\bullet x = {}^\bullet y \wedge x^\bullet = y^\bullet) \Rightarrow x = y$.
In this paper only simple Petri nets have been considered.
Remark 1. Suitable form to describe PN is: PN = {t: ($^\bullet t, t^\bullet$) | for $t \in T$}

Definition 2.
A. [10] Marking of Petri net PN = (P, T, F) is mapping $m: P \rightarrow \mathbb{N}$ (or $m \in \mathbb{N}^P$), where $\mathbb{N} = \{0, 1, 2, \ldots\}$.
B. P/T-system is 3-tuple PS = (PN, δ, m_0), where: PN = (P, T, F) is Petri net, $m_0 \in \mathbb{N}^P \setminus \{\underline{0}\}$ is a initial marking (where $\forall p \in P: \underline{0}(p) = 0$), $\delta: \mathbb{N}^P \times T \rightarrow \mathbb{N}^P$ is a partial function which describes behavior of PS so, that:
$\forall(m, t) \in dom.\delta : \delta(m, t) = (m - {}^\bullet\underline{t}) + \underline{t}^\bullet$, where
$${}^\bullet\underline{t}: P \rightarrow \{0,1\},\ {}^\bullet\underline{t}(p) = \begin{cases} 1, p \in {}^\bullet t \\ 0, p \notin {}^\bullet t \end{cases}, \underline{t}^\bullet: P \rightarrow \{0,1\}, \underline{t}^\bullet(p) = \begin{cases} 1, p \in t^\bullet \\ 0, p \notin t^\bullet \end{cases},$$
$dom.\delta = \{ (m, t) \in \mathbb{N}^P \times T \mid {}^\bullet\underline{t} \leq m \}$.
C. (notation)
- $\forall t \in T: {}^{\bullet\bullet}t = \{m \in M \mid (m, t) \in dom.\delta\}$,
- $\forall t \in T: t^{\bullet\bullet} = \{m \in M \mid \exists m' \in {}^{\bullet\bullet}t: m = \delta(m', t)\}$,
- $\forall m \in M: {}^{\bullet\bullet}m = \{t \in T \mid m \in t^{\bullet\bullet} \}$,
- $\forall m \in M: m^{\bullet\bullet} = \{t \in T \mid m \in {}^{\bullet\bullet}t\}$,
- $\forall m \in M:]m[_P = \{p \in P \mid m(p) \geq 1\}$.

Definition 3. [10] For every P/T-system PS = (P, T, F, δ, m_0) set M of reachable markings is defined as a minimal set, so that:
(i) $m_0 \in M$,
(ii) $\forall m' \in \mathbb{N}^P: \forall t \in T: [((m', t) \in dom.\delta \wedge m' \in M) \Rightarrow \delta(m', t) \in M]$.

Definition 4. [10, 6] P/T-system PS = (P, T, F, δ, m_0) is called safe iff
$\forall m \in M: \forall p \in P: m(p) \leq 1$.
Only safe P/T-systems are used in this work.

Definition 5. TSP = (M, T, Δ, m_0) is T(PS)-system generated by P/T-system PS = (P, T, F, δ, m_0) if
(i) M is a set of reachable markings of PS,
(ii) T is a set of transitions of PS,
(iii) $\Delta = \{(m, t, m') \in M \times T \times M \mid T \cap (m^{\bullet\bullet} \cap {}^{\bullet\bullet}m') \neq \emptyset\}$,
(iv) m_0 is initial marking of PS**.**

An example of T(PS)-system is shown as follows in Fig. 1b.

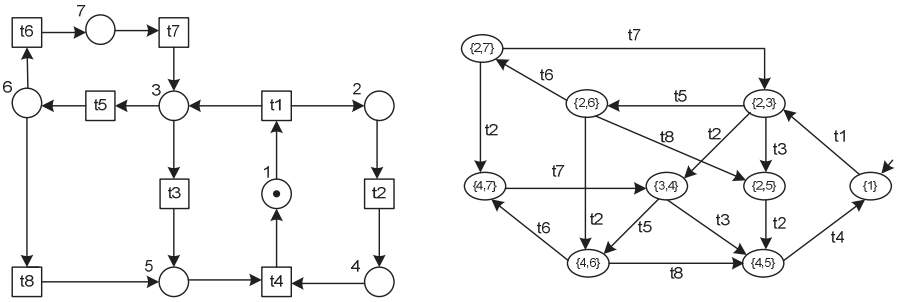


Fig. 1. Graphical representation of P/T-system PS = (PN, δ, m_0) with initial marking indicated as a dot placed at place 1,where PN = {t1: ({1},{2,3}), t2: ({2},{4}), t3: ({3},{5}), t4: ({4,5},{1}), t5: ({3},{6}), t6: ({6},{7}), t7: ({7},{3})} (a) and T(PS)- system generated by PS, where states (markings) are describe as sets of marked places (resp. to Def. 2.C) (b)

Theorem 1. [13] Every T(PS)-system PS = (P, T, F, δ, m_0) generated by P/T-system is a transition system in the sense of [8, 2, 3]**.**
For this reason, T is being considered as a set of events, M as a set of states and Δ as a set of transition between states. It is assumed that events in the system can be enabled or disabled. It is a result of the internal system structure. P/T-system is fulfilling this role. Nevertheless, this inner behavior can be subjected to the influence of outside events and generates changes in the environment.

Definition 6. Interpreted P/T-system (I-P/T-system) is a 7-tuple IPS = (PS, XB, YB, LX, LY, ∂, Λ), where PS = (P, T, F, δ, m_0) is a P/T-system, $XB = \{0,1\}^X$, $YB = \{0,1\}^Y$ are function sets describing appropriately the states of input signals X = { x_1, x_2,..., x_n}, and output signals Y = { y_1, y_2,..., y_k }. The set values of output signals Y is expanded to set {0,1,-},where – is 'don't-care' symbol[1], than $YB^- = \{0,1,-\}^Y$ is an expanded set of states of output signals.
LX: T → FB(X ×{0, 1}) is a labeling function of transitions, where for every t ∈T: LX(t) is a boolean formula which consists of variables (x, i) ∈ X × {0,1} and logical operators: and, or.

[1] IEEE std 1164 for VHDL. [11]

LY: P $\rightarrow$ {a $\subseteq$ Y×{0,1} | $\exists$u∈YB: a $\subseteq \bar{\underline{u}}$} is a labeling function of places such that $\forall m \in M, \exists u \in YB: \bigcup_{p \in]m[_P} LY(p) \subseteq \bar{\underline{u}}$, where: M is a set of reachable markings of PS, $\bar{\underline{u}} = \{(y, i) \in Y \times \{0, 1\} \mid i = u(y)\}$, $]m[_P = \{p \in P \mid m(p) \neq 0\}$

Function $\partial : M \times XB \rightarrow 2^M$ is an external transition of IPS. The value of ∂ is being calculated according to the pattern: $\forall (m, w) \in M \times XB$:

$$\partial(m, w) = \begin{cases} \{\underline{\delta}(m, sel(m, w))\}, & \text{if } stepTest(sel(m, w)) = True \\ \bigcup_{\tau \in max(2^{sel(m,w)} \cap step(m))} \{\underline{\delta}(m, \tau)\}, & \text{if } \begin{cases} sel(m, w) \neq \emptyset \wedge \\ stepTest(sel(m, w)) = False \end{cases} \\ \{m\}, & \text{if } sel(m, w) = \emptyset \end{cases}$$

Case of the non-deterministic behavior for (m, w) occurs, when:

$sel(m, w) \neq \emptyset \wedge stepTest(sel(m, w)) = False$, where: sel: M × XB $\rightarrow 2^T$, sel(m, w) = SEL(w) ∩ $m^{\bullet\bullet}$, SEL: XB $\rightarrow 2^T$, SEL(w) = {t∈T | EVAL(w, LX(t)) = True},

EVAL: XB × FB(X × {0,1}) $\rightarrow$ {True, False} is a function which evaluates labels of transitions. The value of EVAL is calculated according to formula:

$$EVAL(w, fb) = \begin{cases} True, & \text{if } fb = (\ ) \\ ev(w, fb), & \text{if } fb \in X \times \{0,1\} \cup \{True, False\} \\ EV(EVAL(w, fb1), EVAL(w, fb2), op), & \text{if others} \end{cases}$$

EV: $(\{True, False\}^2) \times Lop \rightarrow \{True, False\}$, $EV((val, val'), op) = val\ op\ val'$

ev: XB × $\big((X \times \{0,1\}) \cup \{True, False\}\big) \rightarrow \{True, False\}$,

$$ev(w, fb) = \begin{cases} True, \text{if } fb \in X \times \{0,1\} \wedge fb = (x, i) \wedge w(x) = i \\ False, \text{if } fb \in X \times \{0,1\} \wedge fb = (x, i) \wedge w(x) \neq i \\ fb, \text{if } fb \in \{True, False\} \end{cases}$$

, where Lop = {and, or}

$max(2^{sel(m, w)} \cap step(m)) =$

$\{\tau \in 2^{sel(m,w)} \cap step(m) \mid \forall \tau' \in 2^{sel(m,w)} \cap step(m): \tau \subseteq \tau' \Rightarrow \tau = \tau'\}$, stepTest: $2^{sel(m,w)} \rightarrow$ {True, False},

$$\forall \tau \in 2^{sel(m,w)}: stepTest(\tau) = \begin{cases} True, & \text{if } \forall (t, t') \in \tau \times \tau: (\ {}^\bullet t \cap {}^\bullet t' = \emptyset) \\ False, & \text{else} \end{cases}$$

$\underline{\delta}$: M × STEP(PS) $\rightarrow$ M, $\underline{\delta}(m, \tau) = \big(M - \sum_{t \in \tau} {}^\bullet \underline{t}\big) + \sum_{t \in \tau} \underline{t}^\bullet$

$STEP(PS) = \bigcup_{m \in M} step(m)$

$step(m) = \{\tau \in 2^T \mid \tau \subseteq m^{\bullet\bullet} \wedge stepTest(\tau) = True\}$ Optional: $\underline{\delta'}$: $]M[\times STEP(PS) \rightarrow]M[$, $\underline{\delta'}(\ ]m[_P, \tau\ ) = (\ ]m[_P \setminus pre(\tau)\ ) \cup post(\tau)$, where: $pre(\tau) = \bigcup_{t \in \tau} {}^\bullet t$, $post(\tau) = \bigcup_{t \in \tau} t^\bullet$, $]M[= \{\ ]m[_P \mid m \in M\ \}$ Λ: M $\rightarrow$ YB^- is an output function of IPS. The value of Λ is being calculated according to the formula:

$\forall m \in M, \forall y \in Y: \Lambda(m) = \Lambda'(m) \cup \{(y, -) \mid y \in Y \wedge \{(y, 0), (y, 1)\} \cap \Lambda'(m) = \emptyset\}$, where $\Lambda'(m) = \bigcup_{p \in]m[_P} LY(p)$.

Discrete values of signals X and Y can be expanded according to the needs, like in Definition 6: from {0, 1} to {0, 1, -}. Example of I-P/T-system is presented in Fig. 2.

Definition 7. A(IPS) = (M, XB, $YB^-, \partial, \Lambda, m_0$) is a Moore automaton generated by I-P/T-system IPS = (PS, XB, YB, LX, LY, ∂, Λ). A(IPS) describes the behavior of IPS.

The behavior of IPS is deterministic if A(IPS) is deterministic, non-deterministic otherwise. The A(IPS) which is deterministic is a special case defined automata. In

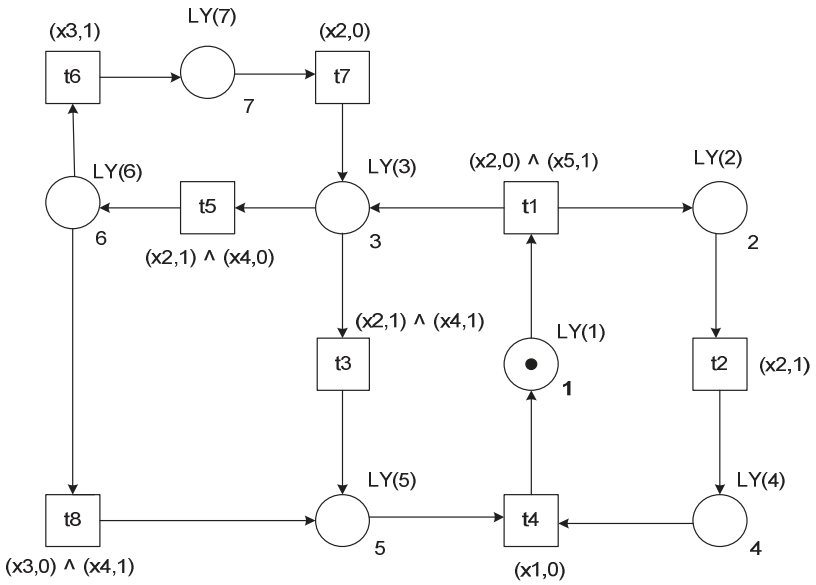
many applications, it is essential to make sure that the considered system is deterministic. So there is a need to make sure that the modeled system possesses this property. Diagram of Moore automaton generated by I-P/T-system specified in Fig. 2 is shown in Fig. 3.

In Definition 8 there is presented a new kind of the P/T-system, named B-system which is used for modeling ladder diagrams.

Remark 2. (notation) For every $w' \in XB$ and every $a \subseteq w'$: $\langle a \rangle = \{w \in XB \mid a \subseteq w'\}$. For example, if a = {(3,0), (4,1)}, than ⟨ a ⟩ = { {(1,0), (2,0), (3,0), (4,1)}, {(1,0), (2,1), (3,0), (4,1)}, {(1,1), (2,0), (3,0), (4,1)}, {(1,1), (2,1), (3,0), (4,1)} },
where $X = \{1, 2, 3, 4\}$. A shorter representation of the $w \in XB$ is possible. For example, the tuple w' = (0, 1, 0, 1) is representing of w = {(1,0), (2,1), (3,0), (4,1)}, at the condition: $(i, w(i)) = (i, w'[i-1])$, or if $(x_i , w(x_i)) \in XB$, $(x_i , w(x_i)) = (x_i , w'[i-1])$

Definition 8. [13] B-system is a pair BS = (PS, η) with a P/T-system PS = (P, T, F, δ, m_0) and bijection $\eta: P^0 \to P^1$, where $\{P^0, P^1\}$ is a partition of P. For η seen as relation $\eta \subseteq P^0 \times P^1$ the following conditions are met: (i) $\forall (p_j^0, p_j^1) \in \eta: |\{p_j^0, p_j^1\} \cap {}^\bullet t| = \forall (p_j^0, p_j^1) \in \eta: |\{p_j^0, p_j^1\} \cap t^\bullet| \leq 1$ (ii) $\forall (p_j^0, p_j^1) \in \eta: |\{p_j^0, p_j^1\} \cap]m_0[_P| = 1$

Definition 9. [10] For given P/T-system PS = (P, T, F, δ, m_0) set $P' \subseteq P$ is called P-invariant iff $\forall m \in M: \sum_{p \in P'} m(p) = \sum_{p \in P'} m_0(p)$

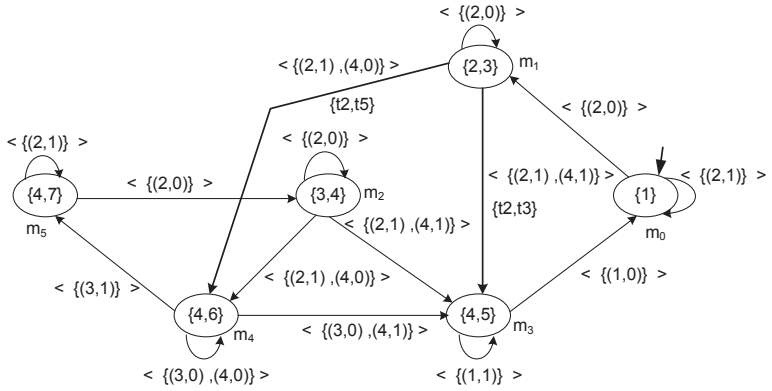


LY(1) = {(u1,0),(u2,0),(u3,0),(u4,0),(u5,0)},
LY(2) = {(u5,0)},
LY(3) = {(u1,1),(u2,0),(u3,0),(u4,0)},
LY(4) = {(u5,1)},
LY(5) = {(u1,0),(u2,0),(u3,1),(u4,0)},
LY(6) = {(u1,0),(u2,1),(u3,0),(u4,0)},
LY(7) = {(u1,0),(u2,0),(u3,0),(u4,1)}

Fig. 2. Graphical representation of I-P/T-system

Theorem 2. [13] For every B-system BS = (P, T, F, δ, m_0, η) the following propositions are true:

(1) $|\eta| = \frac{1}{2}|P|$, (2) $\forall(p_j^0, p_j^1) \in \eta: \{p_j^0, p_j^1\}$ is a P-invariant, (3) $\forall(p_j^0, p_j^1) \in \eta: p_j^{0\bullet} \cap {}^\bullet p_j^1 \neq \emptyset \vee p_j^{1\bullet} \cap {}^\bullet p_j \neq \emptyset$, (4) $\forall t \in T: |{}^\bullet t| = |t^\bullet|$, (5) PS is a 1-P/T-system.



Outputs

$\Lambda(m_0)$ = {(1,0),(2,0),(3,0),(4,0),(5,0)}, $\Lambda(m_1)$ = {(1,1),(2,0),(3,0),(4,0),(5,0)},
$\Lambda(m_2)$ = {(1,1),(2,0),(3,0),(4,0),(5,1)}, $\Lambda(m_3)$ = {(1,0),(2,0),(3,1),(4,0),(5,1)},
$\Lambda(m_4)$ = {(1,0),(2,1),(3,0),(4,0),(5,1)}, $\Lambda(m_5)$ = {(1,0),(2,0),(3,0),(4,1),(5,1)}

Fig. 3. Diagram of Moore automaton generated by I-P/T-system specified in Fig. 2

3. PLC LADDER DIAGRAM LANGUAGE

Ladder diagrams are an industrial programming language typically used on programmable logic controllers (PLC). The ladder diagram (LD), as a PLC language, consists of two vertical lines representing the power rails. Rungs of LD are circuits connected as horizontal lines between the power rails. The left part of the rang consists of symbols depicted by double vertical lines (resp. slashed), they are units which are characterized by the property that they turn conductible if the corresponding inputs are true (resp. false). The ellipse (or ellipses) placed on the right part of the rung represents the output. Each rung on the LD defines one operation in the control process.

The LD shown in Fig. 4 describes the control process of the neutralization tank system presented in Example 1. The p-th rung from the top of LD is denoted by lp. In this paper, it is assumed that the type of inputs and outputs is restricted to binary variables. For an LD consisting of n rungs, each output is calculated in PLC by scanning from the first rung to the n-th rung. The controlled plant is driven according to the output signals obtained immediately after each scanning. The scanning process is presented in Fig. 5.

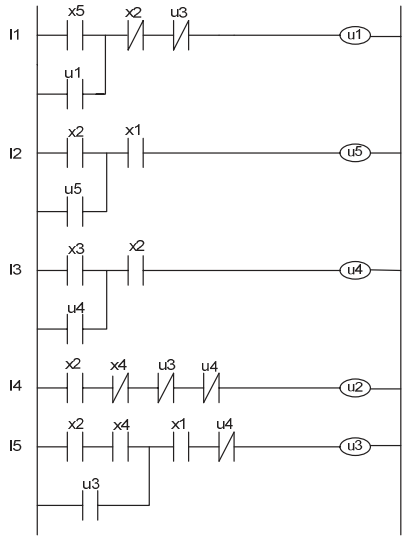


Fig. 4. [12] Ladder diagram for the neutralization tank system control process. The plant is shown in Fig. 6

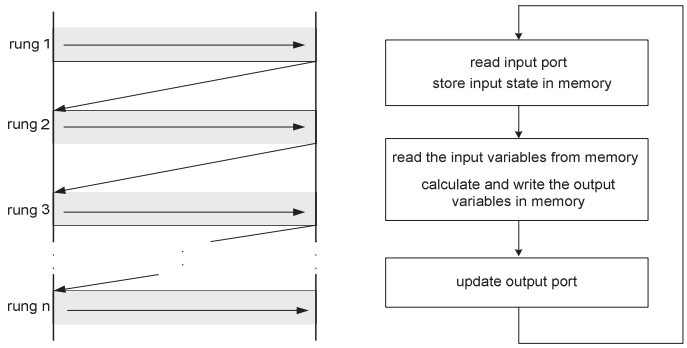


Fig. 5. Scanning process of the LD program

Example 1. Neutralization tank system [12]
The specification for the plant shown in Fig. 6 is given as follows:

1. At the initial state, all of the valves and the mixer should be off and the tank is empty.
2. When the starting switch turns on, open valve U1.
   Inject liquid from reservoir unless x2 = 1.
   2.a. Keep injecting liquid from reservoir. The mixer starts when x2 = 1.
   2.b. Sensor x4 detects pH of liquid. If the value does not reach the specified pH value, then open valve U2 to inject a neutralizing agent.
3. Keep injecting the neutralizing agent. If sensor x3 = 1, stop injecting the neutralizing agent and open valve U4 to drain the liquid until the liquid level is at the position of sensor x2. Close valve U4 again and go back to the process 2.b.
4. If pH of the liquid is satisfied (process completed), close valve U2 and drain the mixed liquid through valve U3. After that, if x1 turns to 0 due to decrease of the liquid level, close valve U3 and go to the process 1.

The LD satisfying the above specification is shown in Fig. 4.

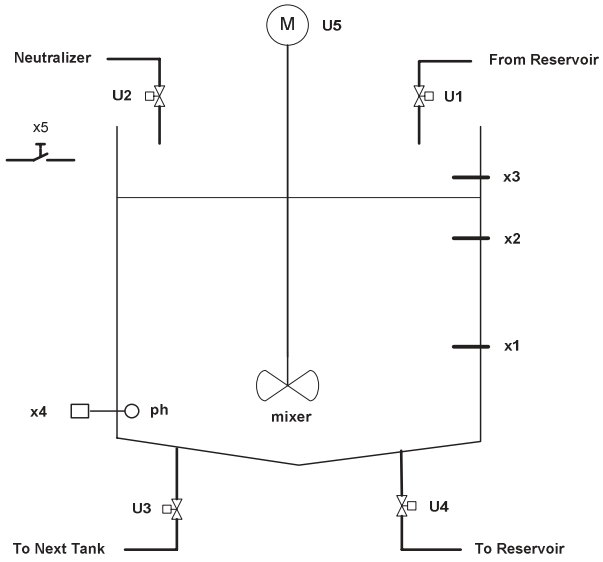


Fig. 6. Neutralization tank system

4. MODLING OF THE PLC LADDER DIAGRAMS

The defined below LD-P/T-system is based on the B-system from definition 8. It is a model of the behavior of PLC which was programmed in LD language. LD scanning was carried out with the help of the function ordscan. The subset of places named Pout represents these LD elements, whose state is being copied to the state of the PLC output port. The model can be enriched with additional elements for example timers, but this aspect has been omitted due to space limits.

Definition 8. LD- P/T-system is every eight-tuple LDPS = (BS, ordscan, XB,YB, LX, LY, $\delta^{\sim}$, Λ), in which BS = (P, T, F, δ, m_0,η) is a B-system, where: P = Pout $\cup$ Pint, and Pint = P \ Pout, Pout is an output places set,
ordscan:{1,2,…,|T|} $\rightarrow$ T is a bijection, which orderings transitions with respect LD scanning order, XB, YB, LX, EVAL are described in def. 6,
LY: Pout $\rightarrow$ {a $\subseteq$ Y×{0,1}| $\exists$u$\in$YB: a $\subseteq \bar{\bar{u}}$} is a Pout labeling function, which value is computed under formula: $\forall$m $\in$ M, $\exists$u$\in$YB: $\bigcup_{p\in]m[_{Pout}} LY(p) \subseteq \bar{\bar{u}}$, where M is a set of reachable markings of BS, $\bar{\bar{u}}$ = {(y, i) $\in$Y×{0, 1} | i = u(y)}.
$\delta^{\sim}$: M × XB $\rightarrow$ M is an external transition function of LDPS, which is computed according to formula: $\forall$(m, w) $\in$ M × XB: $\delta^{\sim}(m, w) = \delta_{|T|,w}\left(\delta_{|T|-1,w}\left(\ldots \delta_{2,w}\left(\delta_{1,w}(m)\right)\right)\right)$,

$\delta_{i,w}$: M $\rightarrow$ M, $\delta_{i,w}(m) = \begin{cases} m, & ordscan(i) \notin sel(m,w) \\ \delta(m, ordscan(i)), & ordscan(i) \in sel(m,w) \end{cases}$,

sel(m, w) = SEL(w) $\cap$ $m^{\bullet\bullet}$, SEL: XB $\rightarrow$ 2^T,
where SEL(w) = {t$\in$T | EVAL(w, LX(t)) = True}.
Λ: M $\rightarrow$ YB^- is an output function of LDPS. Value of the function is computed according to formula: $\forall$m$\in$M, $\forall$y$\in$Y: $\Lambda(m) = \bigcup_{p\in]m[_{Pout}} LY(p)$.

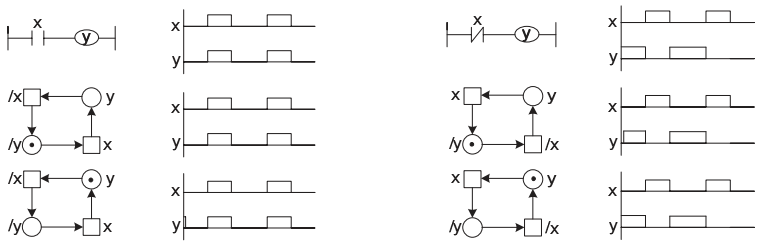


Fig. 7. Modeling semantic

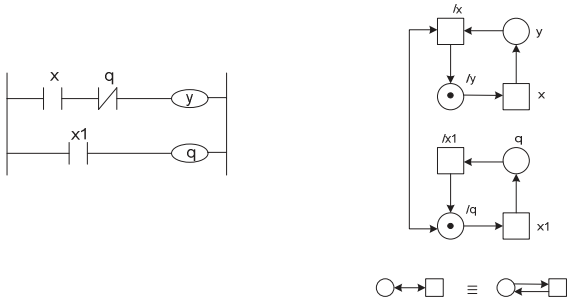


Fig. 8. LD inner feedback modeling example

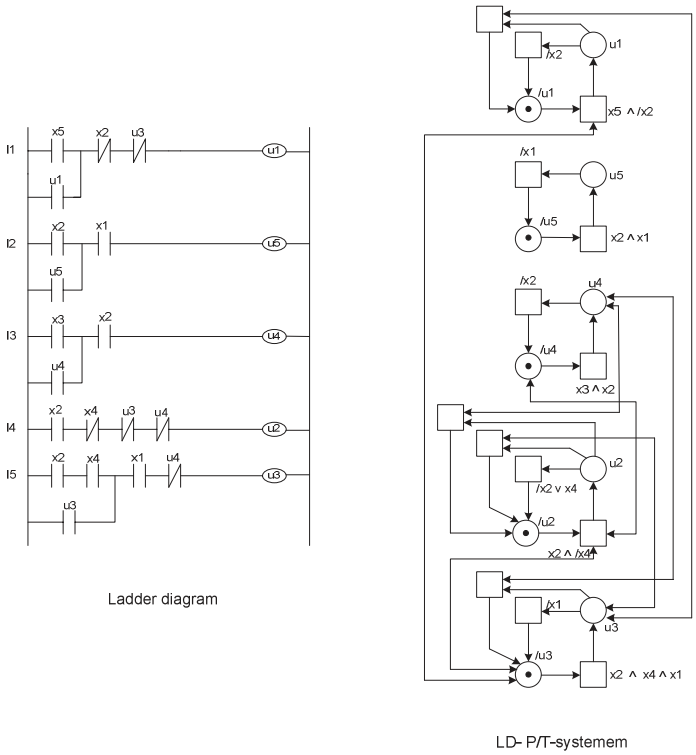


Fig. 9. Ladder diagram for the neutralization tank system from example 1 and its LD-P/T-system graphical model

Algorithm for generating global states transitions diagram of closed loop system consists of LD-P/T-system and the plant is presented in appendix B. State-transition diagram, as result of the algorithm, is presented in Fig. 12. Examined Example 1 was taken from [12]. The proposed there method of the generation state-transition diagram for closed loop system for controller-plant is based on the theory of difference equation. Our result for the same example differs in the fact, that transition ((1,1, 0,0), (0,1,0,0,1)) → ((1,1,0,1), (0,0,1,0,1)) from diagram presented in Fig. 12, is missing in [12]. The LD is simple, so it is easy to prove that the transition exists.

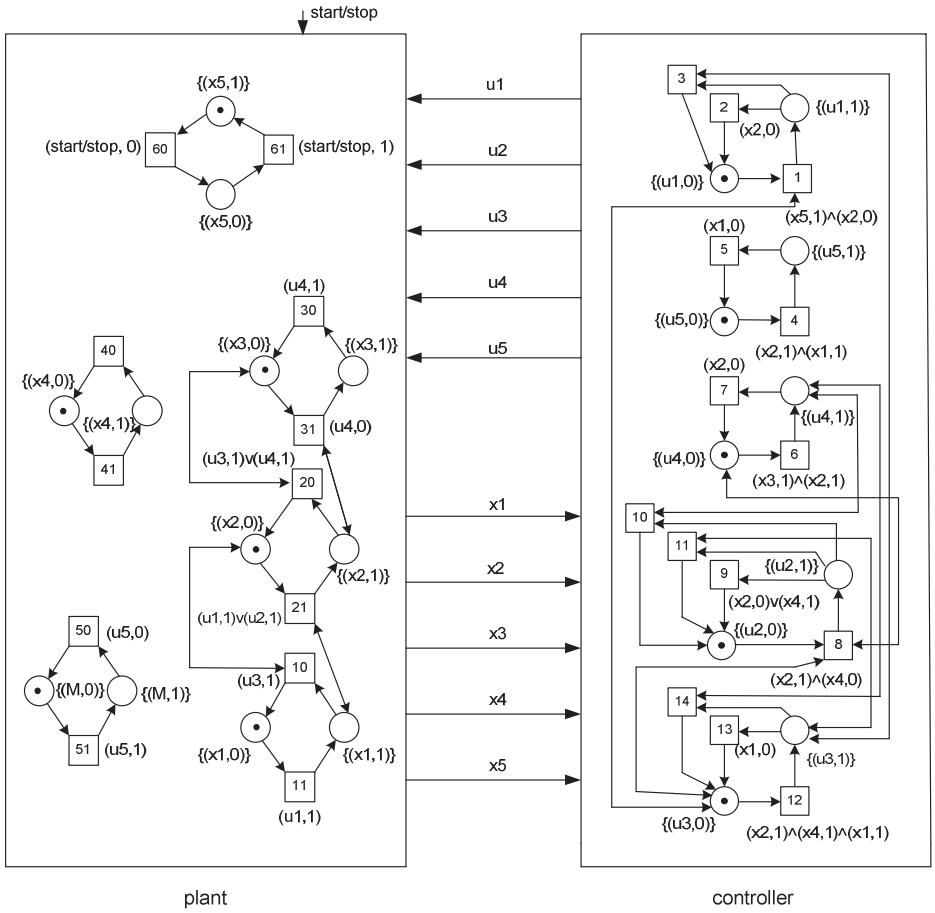


Fig. 10. Closed loop system controller-plant for example 1

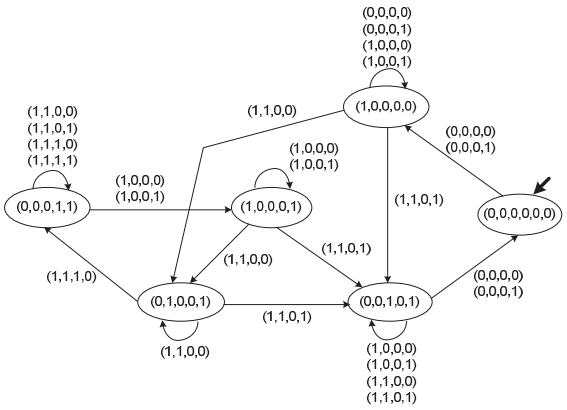


Fig. 11. Diagram of Moore automaton generated by LD-P/T-system of controller

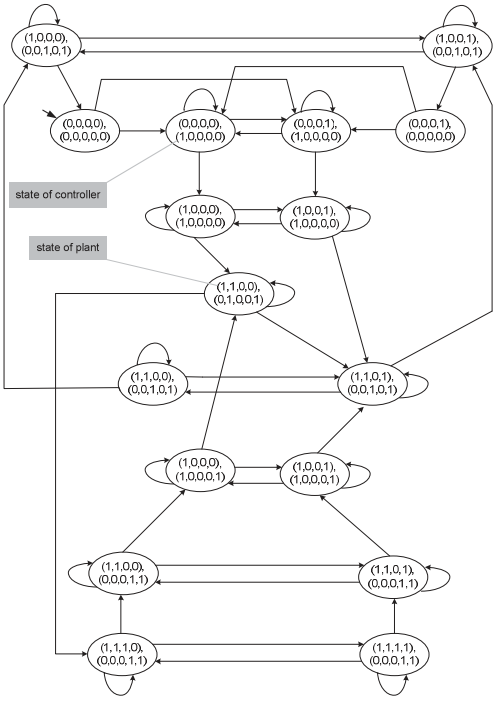


Fig. 12. Global states transitions diagram for closed loop system controller-plant from Fig. 10

The diagram shown in Fig. 11 represents Moore automaton generated by LD-P/T-controller system. It is simple to calculate – in frames of the same model – the automaton by supplementing a little the existing algorithm. This Moore automaton constitutes the base for the new I-P/N-system synthesis. The new system more precisely describes the behavior of controller, since it presents the information about the sequence of events in an open way. The I-P/N-system, for example 1, is presented in fig. 2. The behavior of this system is isomorphic to the behavior of LD-P/T-system presented in

Fig. 11. Description of the control algorithm in the form of I-P/N-system gives the possibility of its implementation via technology different than PLC technology, for example FPGA. [1, 13] It can be important when we deal with the embedded system.

5. CONCLUSION

In this paper, several models based on Petri nets have been considered. We propose a new model of PLC ladder diagram and algorithm to compute the state space of the closed loop system which consists of a controller and a plant. This result has a potential of being applied for fault handling and synthesis of new models of controllers which can be carried out on FPGA platforms[1, 13, 4]. Future subjects include LD-VHDL transformation for FPGA implementation.

APPENDIX

A. I-P/T-system

Specification of I-P/T-system from fig. 2:

```
PN={'t1':[{1},{2,3}],'t2':[{2},{4}],'t3':[{3},{5}],'t4':[{4,5},{1}],'t5':[{3},{6}],'
t6':[{6},{7}],
't7':[{7},{3}],'t8':[{6},{5}]}
P={1,2,3,4,5,6,7}
m=frozenset({1})
LX={'t1':(2,0),'t2':(2,1),'t3':((2,1),(4,1),'and'),'t4':(1,0),'t5':((2,1),(4,1),'and
'),'t6':(3,1),'t7':(2,0),'t8':((3,0),(4,1),'and')}
LY={ 1:{(1,0),(2,0),(3,0),(4,0),(5,0)}, 2: {(5,0),(4,1)},
3:{(1,1),(2,0),(3,0),(4,0)}, 4: {(5,1)}, 5:{(1,0),(2,0),(3,1),(4,0)}, 6:
{(1,0),(2,1),(3,0),(4,0)}, 7: {(1,0),(2,0),(3,0),(4,1)} }
XB={0: (0, 1, 0, 0), 1: (0, 1, 1, 0), 2: (0, 0, 1, 0), 3: (0, 0, 1, 1), 4: (0, 1, 1,
1), 5: (0, 1, 0, 1), 6: (0, 0, 0, 1), 7: (1, 0, 1, 1), 8: (1, 0, 0, 1), 9: (1, 1, 0,
1), 10: (1, 1, 0, 0), 11: (1, 0, 0, 0), 12: (1, 0, 1, 0), 13: (1, 1, 1, 0), 14: (0,
0, 0, 0), 15: (1, 1, 1, 1)}
X={1,2,3,4}
Y={1,2,3,4,5,6}
```

(modules are coded with using Python 3)

Module for EVAL: $XB \times FB(X \times \{0,1\}) \rightarrow \{True, False\}$

```
LX={'t1':(2,0),'t2':(2,1),'t3':(((2,1),(4,0),'and'),((3,1),(1,1),'and'),'or'),'t4':(
1,0),'t5':((2,1),(4,0),'and'),'t6':(3,1),'t7':(2,0),'t8':((3,0),(4,1),'and')}
XB={0:(0, 0, 0, 1),1: (1, 0, 0, 1), 2: (1, 1, 0, 1), 3: (1, 1, 0, 0), 4: (1, 0, 0,
0), 5: (1, 1, 1, 0), 6: (0, 0, 0, 0), 7: (1, 1, 1, 1)}
tranz='t3', number_w=2
fb =LX[tranz]= (((2,1),(4,0),'and'),((3,1),(1,1),'and'),'or')
w = XB[2]= (1, 1, 0, 1) # Remark 2
input: (1, 1, 0, 1), (((2,1),(4,0),'and'),((3,1),(1,1),'and'),'or')
output: False
def ev(w,fb1):
 if fb1==True:
 return True
 elif fb1==False:
 return False
 return fb1[1]==XB[w][fb1[0]-1]

def EV(fb2):
 if fb2[2]=='and':
 val=fb2[0] and fb2[1]
 elif fb2[2]=='or':
 val=fb2[0] or fb2[1]
 return val
def EVAL(w,fb):
```

```
 if fb==():
 return True
 elif fb!=() and len(fb)<=2:
 return ev(w,fb)
 elif len(fb)>2:
 temp1=EVAL(w,fb[0])
 temp2=EVAL(w,fb[1])
 temp3=(temp1,temp2,fb[2])
 return EV(temp3)
```

Module for sel: $M \times XB \to 2^T$

```
def fo(mark):
 return {t for t in sorted(PN) if PN[t][0] <= mark}

def SEL(w):
 return {i for i in sorted(LX) if EVAL(w,LX[i])==True}

def sel(mark,w):
 return SELw(w)&fo(mark)
```

fo(mark) = mark$^{\bullet\bullet}$

Module for stepTest: $2^{sel(m,w)} \to \{True, False\}$

```
def stepTest(TrSet):
 preTrSet=set()
 postTrSet=set()
 log=True
 temp=tuple(TrSet)
 temp1=list(temp)
 for i in range(len(temp)):
 preTrSet.update(PN[temp[i]][0])
 postTrSet.update(PN[temp[i]][1])
 temp1.remove(temp[i])
 for j in temp1:
 b=PN[temp[i]][0]&PN[j][0]==set()
 c=log and b
 log=c
 temp1=list(temp)
 return [log,preTrSet,postTrSet]
```

#stepTest(TrSet)[0] = log = stepTest(TrSet),
#PreTrSet = pre(TrSet) = $\cup_{t\in TrSet}\ {}^{\bullet}t$, PostTrSet = post(TrSet) = $\cup_{t\in TrSet}\ t^{\bullet}$

Module for ∂' : $M \times XB \to M \cup \{('NONDETERMINISM\ for', arg1, arg2)\}$

```
def ExtTrans(mark,w):
 TrSet=sel(mark,w)
 if TrSet==set():
 return mark
 elif TrSet!=set() and stepTest(TrSet)[0]:
 return (mark - stepTest(TrSet)[1]) | stepTest(TrSet)[2]
 elif not stepTest(TrSet)[0]:
 print('NONDETERMINISM for', mark, TrSet)
 return mark
```

Module for Λ: $M \to YB^-$

```
def lamb(mark):
 out_mark1=set()
 for p in mark:
 out_mark1.update(LY[p])
 return out_mark1
def LAMBDA(mark):
 out_mark=lamb(mark)
 temp={i[0] for i in out_mark}
 temp3=frozenset(Y)
 temp1=Y
 temp1.difference_update(temp)
 temp2={(j,'-') for j in temp1}
 out_mark.update(temp2)
 test={i for i in temp3 if ({(i,0),(i,1)} <= out_mark)}
```

```
 if test != set():
 print('LY incorrect for', test, mark)
 return out_mark
```

Module for A(IPS) generating (breadth_first_search method)[2]

```
From=RM={m}
Arcs={}
def XBnextM(From1,NM1,Arcs1):
 for curr_mark in From1:
 for i in sorted(XB):
 next_mark=ExtTrans(curr_mark,i)
 NM1.update({next_mark})
 aa=(curr_mark,next_mark)
 bb=XB[i]
 if aa in Arcs1.keys():
 Arcs1[aa].append(bb)
 else:
 Arcs1.update({aa:[bb]})
 return [NM1, Arcs1]
def XBreachM(From):
 NM=set()
 while len(From) > 0:
 From = XBnextM(From,NM,Arcs)[0].difference(RM)
 RM.update(From)
 return Arcs,RM
OutputMap={i:LAMBDA(i) for i in XBreachM(From)[1]}
print('Arcs =',XBreachM(From)[0])
print('Outputs =',OutputMap)
Arcs = {(frozenset({2, 3}), frozenset({4, 5})): [(0, 1, 1, 1), (0, 1, 0, 1), (1, 1,
0, 1), (1, 1, 1, 1), (0, 1, 1, 1), (0, 1, 0, 1), (1, 1, 0, 1), (1, 1, 1, 1)],
(frozenset({2, 3}), frozenset({2, 3})): [(0, 0, 1, 0), (0, 0, 1, 1), (0, 0, 0, 1),
(1, 0, 1, 1), (1, 0, 0, 1), (1, 0, 0, 0), (1, 0, 1, 0), (0, 0, 0, 0), (0, 0, 1, 0),
(0, 0, 1, 1), (0, 0, 0, 1), (1, 0, 1, 1), (1, 0, 0, 1), (1, 0, 0, 0), (1, 0, 1, 0),
(0, 0, 0, 0)], ……
……
Outputs = {frozenset({3, 4}): {(3, 0), (2, 0), (5, 1), (1, 1), (4, 0)},
frozenset({2, 3}): {(2, 0), (5, 0), (3, 0), (4, 1), (1, 1), (4, 0)}, frozenset({4,
6}): {(3, 0), (5, 1), (1, 0), (2, 1), (4, 0)}, frozenset({4, 7}): {(3, 0), (5, 1),
(1, 0), (4, 1), (2, 0)}, frozenset({1}): {(3, 0), (2, 0), (1, 0), (5, 0), (4, 0)},
frozenset({4, 5}): {(4, 0), (5, 1), (1, 0), (3, 1), (2, 0)}}
```

B. Listing for global states transitions diagram of closed loop system consists of LD-P/T-system and plant

(with using Python 3)

Specification of LD-P/T-system (controller) from fig. 10:

```
PNLD={1:[{(1,0),(3,0)},{(1,1),(3,0)}],2:[{(1,1)},{(1,0)}],3:[{(1,1),(3,1)},{(1,0),(3
,1)}],4:[{(5,0)},{(5,1)}],5:[{(5,1)},{(5,0)}],6:[{(4,0)},{(4,1)}],7:[{(4,1)},{(4,0)}
],8:[{(2,0),(4,0),(3,0)},{(2,1),(4,0),(3,0)}],9:[{(2,1)},{(2,0)}],10:[{(2,1),(4,1)},
{(2,0),(4,1)}],11:[{(2,1),(3,1)},{(2,0),(3,1)}],12:[{(3,0),(4,0)},{(3,1),(4,0)}],13:
[{(3,1)},{(3,0)}],14:[{(3,1),(4,1)},{(3,0),(4,1)}]}
Pl={(1,1),(1,0),(5,1),(5,0),(4,1),(4,0),(2,1),(2,0),(3,1),(3,0)}
ml=frozenset({(1,0),(5,0),(4,0),(2,0),(3,0)})
LXl={1:(2,0),2:(2,1),3:(),4:((1,1),(2,1),'and'),5:(1,0),6:((3,1),(2,1),'and'),7:(2,0
),8:((2,1),(4,0),'and'),9:((2,0),(4,1),'or'),10:(),11:(),12:(((2,1),(4,1),'and'),(1,
1),'and'),13:(1,0),14:()}
LYl={'11':{(1,1)},'10':set(),'51':{(5,1)},'50':set(),'41':{(4,1)},'40':set(),'21':{(
2,1)},'20':set(),'31':{(3,1)},'30':set()}
XBl={0:(0, 0, 0, 1),1: (1, 0, 0, 1), 2: (1, 1, 0, 1), 3: (1, 1, 0, 0), 4: (1, 0, 0,
0), 5: (1, 1, 1, 0), 6: (0, 0, 0, 0), 7: (1, 1, 1, 1)}
```

Specification of I-P/T-system (plant) from fig. 10:

```
PNPL={'11':[{(1,0)},{(1,1)}],'10':[{(1,1),(2,0)},{(1,0),(2,0)}],'21':[{(2,0),(1,1)},
{(2,1),(1,1)}],'20':[{(2,1),(3,0)},{(2,0),(3,0)}],'31':[{(3,0),(2,1)},{(3,1),(2,1)}]
,'30':[{(3,1)},{(3,0)}],'41':[{(4,0)},{(4,1)}],'40':[{(4,1)},{(4,0)}]}
Pp={(1,1),(1,0),(2,1),(2,0),(3,1),(3,0),(4,1),(4,0)}
```

[2] Idea of the method was adopted from [3], which was used in generating reachable markings of Petri nets

```
mp=frozenset({(1,0),(2,0),(3,0),(4,0)})
LXp={'11':(1,1),'10':(3,1),'21':((1,1),(2,1),'or'),'20':((3,1),(4,1),'or'),'31':(2,1
),'30':(4,1),'41':(),'40':()}
XBp={0: (0, 1, 0, 0), 1: (0, 1, 1, 0), 2: (0, 0, 1, 0), 3: (0, 0, 1, 1), 4: (0, 1,
1, 1), 5: (0, 1, 0, 1), 6: (0, 0, 0, 1), 7: (1, 0, 1, 1), 8: (1, 0, 0, 1), 9: (1, 1,
0, 1), 10: (1, 1, 0, 0), 11: (1, 0, 0, 0), 12: (1, 0, 1, 0), 13: (1, 1, 1, 0), 14:
(0, 0, 0, 0), 15: (1, 1, 1, 1)}
```

Signal x5 is switched on (x5, 1) and omitted, since it is the one which is accepted only in the initial state. For this reason transitions '60','61' are not presented in the specification. Transitions '50' and '51' are omitted also because the mixer does not exert a direct influence on the state of the controller.

```
def nextLDstate(NLD,s,LXl):
 #fitting marking of the LD-controller to state s[1]
 markl={(i+1,s[1][i]) for i in range(len(s[1]))}
 #-----------------------------------------------
 #LD-controller marking and output state updating according to
 #sequence of rungs
 TrList=sorted(LXl)
 tt=TrList[0]
 z=markl
 while tt <= TrList[len(TrList)-1]:
 if tt in sel(z,s[0],NLD,LXl):
 nn = (z - NLD[tt][0]) | NLD[tt][1]
 z=nn
 else:
 nn=z
 tt1=tt+1
 tt=tt1
 mn1=nn
 #-----------------------------------------------
 #fitting output of the LD-controller to new marking
 mn1list=list(mn1)
 OutLDdic={i[0]:i[1] for i in mn1list}
 OutLD=[OutLDdic[i] for i in sorted(OutLDdic)]
 OutLD1=tuple(OutLD)
 #-----------------------------------------------
 #global state updating
 s1=(s[0],OutLD1)
 return s1

def nextPL_LDstate(NLD,NPL,s,NewStates1,transArc1):
 #LD-controller reaction, storage of new global state and transition arc
 z0=nextLDstate(NLD,s,LXl)
 NN1.update({z0})
 transArc1.update({(s,z0)})
 #-------------------------
 #fitting marking of the plant to state s[0]
 markp={(i+1,s[0][i]) for i in range(len(s[0]))}
 TrSetp=sel(markp,s[1],NPL,LXp)
 for t in TrSetp:
 #next marking of plant
 #no steps, since the reaction of the plant is slower than the controller reaction
 new_markp=(markp - NPL[t][0]) | NPL[t][1]
 #fitting output of the plant to new marking
 new_marklList=list(new_markp)
 new_marklList.sort()
 OutPLdic={i[0]:i[1] for i in new_marklList}
 OutPL=[OutPLdic[i] for i in sorted(OutPLdic)]
 OutPL1=tuple(OutPL)
 #global state updating
 z2=(OutPL1,s[1])
 #----------------------------
 #-------------------------------------------------
 #LD controller reaction, storage of new global state and transition arc
 #(for every new state of output of the plant)
 z3=nextLDstate(NLD,z2,LXl)
 NewStates1.update({z3})
 transArc1.update({(s,z3)})
```

```
 return NewStates1,transArc1

From=ReachStates={s0}
transArc2=set()
def Simage(NLD,NPL,From):
 NewStates=set()
 transArc=set()
 for state in From:
 nextPL_LDstate(NLD,NPL,state,NewStates,transArc)
 return NewStates,transArc

def ReachS(NLD,NPL,From):
 while From!=set():
 temp=Simage(NLD,NPL,From)[1]
 transArc2.update(temp)
 From=Simage(NLD,NPL,From)[0].difference(ReachStates)
 ReachStates.update(From)
 return ReachStates,transArc2

print(ReachS(PNLD,PNPL,From)[1])

{(((1, 0, 0, 1), (1, 0, 0, 0, 1)), ((1, 0, 0, 0), (1, 0, 0, 0, 1))), (((1, 0, 0, 1),
(0, 0, 1, 0, 1)), ((0, 0, 0, 1), (0, 0, 0, 0, 0))), (((1, 1, 1, 1), (0, 0, 0, 1,
1)), ((1, 1, 1, 1), (0, 0, 0, 1, 1))), (((1, 1, 1, 1), (0, 0, 0, 1, 1)), ((1, 1, 1,
0), (0, 0, 0, 1, 1))),……..
……..
```

BIBLIOGRAPHY

[1] Adamski M., Monteiro J. L., 2000. From Interpreted Petri Net Specification to Reprogrammable Logic Controllers, Indusrial Electroncs, ISIE'2000. Proceedings of the 2000. Cholula, Pueblo: IEEE, 2000. 13-19.

[2] Bernardinello L., Ferigato C., Pomello L., 2003. An Algebraic model of observable properties in distributed systems, TCS.

[3] Cortadella J., Kishnievsky M., Lavagno L., Yakovlev A., 1998. Derevating Petri Nets From Finite Transitin Systems, IEEE Trans on Comp. Vol. 47, No. 8.

[4] Du D., Liu Y., Guo X., Yamazaki K., 2008. Study on LD-VHDL conversion for FPGA-based PLC implementation, Int. J. Adv. Manuf. Technol. Springer-Verlag.

[5] Fujimoto, Y., 2002. Design of Discrete time Polinominal Nonlinear Systems and Its Application to Sequential Control, T. IEE Japan, Vol 122-D, No. 9, 918-927.

[6] Girauld C., Valk R., 2001. Petri Nets for System Engineering, Berin, Springer-Verlag.

[7] Lee, Jin-Shyan, Chun-Chieh Chuang, 2009. Development of a Petri net-based fault diagnostic system for industrial processes, Industrial Electronics, IECON '09. 35th Annual Conference of IEEE, 4347-4352.

[8] Nielsen M., Rozenberg G., Thiagarajan P.S., 1992. Elementary Transition Systems, TCS 96.

[9] Pastor E., Cortadella J., Roig O., 2001. Symbolic Analysis of Bounded Petri Nets, IEEE Tras. on Comp. Vol. 50, No. 5.

[10] Reisig, W., 1988. Sieci Petiego. WNT Warszawa.

[11] Wright D., (Chair)., 2004. IEEE Standards for VHDL Register Transfer Level (RTL) Synthesis. IEEE Std 1076.6- 2004, New York, IEEE.

[12] Zanma T., Miyabayashi T., Ishida M., 2004. Sequence generation of discrete event system and logic controller and its applicability to fault detection, Advanced Motion Control, AMC '04. The 8th IEEE International Workshop on Digital Object Identifi,: 10.1109/AMC.2004.1297936, 601-606.

[13] Zech W., 2008. Właściwości struktury sieci Petriego i ich wykorzystanie do syntezy układów sterownia binarnego, rozprawa doktorska. Politechnika Poznańska.

MODELE SYSTEMÓW ZDARZEŃ DYSKRETNYCH SKONSTRUOWANE W OPARCIU O SIECI PETRIEGO I GENERACJA SEKWENCJI STANÓW ZAMKNIĘTEGO SYSTEMU OBIEKT-STEROWNIK

Streszczenie

W artykule przedstawiono formalny model diagramu drabinkowego (LD) jako LD-P/T-system. Skonstruowano model zamkniętej pętli sprzężenia między sterownikiem (LD) i sterowanym obiektem. Przedstawiono algorytm generacji diagramu przejść między stanami takiego systemu. Możliwa jest detekcja uszkodzenia, gdy wygenerowany zostanie nieprzewidziany stan. Dodatkowa korzyść z takiego podejścia wynika z faktu, że możliwa jest konstrukcje diagramu przejść samego sterownika, co może być wykorzystane do transformacji diagramów drabinkowych na model dający się opisać w języku VHDL i implementować w FPGA.

Słowa kluczowe: system zdarzeń dyskretnych, sieci Petriego, programowalne sterowniki sekwencyjne, diagramy drabinkowe