Modeling and Race Detection of Ladder Diagrams via Ordinary Petri Nets

Jiliang Luo, *Member, IEEE*, Qi Zhang, Xuekun Chen, and MengChu Zhou, *Fellow, IEEE*

***Abstract*—This paper presents an ordinary Petri net (PN)-based approach to the modeling and race-detection problems of programs for programmable logic controllers (PLCs). First, a PLC program is formalized by a graph where nodes represent contacts and coils. Second, an algorithm is proposed to translate this graph into an ordinary PN. Third, a method is presented to detect whether there exists a race in a program by using a reachability graph technique, to locate a race by introducing the race path, and to correct a race by analyzing the subnet that contains it. An example is utilized to illustrate the theoretic results.**

***Index Terms*—Discrete event systems, formal methods, IEC61131-3 standard, industrial automation, ladder diagram (LD), Petri nets (PNs), race detection, supervisory control.**

I. Introduction

As AN important industrial control device, programmable logic controllers (PLCs) play a significant role in automatic control systems. A ladder diagram (LD) is the most widely used programming language for PLCs, which is transparent and intuitive, since the variables are represented as graphical symbols and each instruction is graphical. Along with the increasing number of sensors and actuators in a plant, the complexity for guaranteeing the correctness of programs increases sharply. Consequently, a formal verification technique attracts more and more attention in order to use it to validate and verify LD programs. In the published work, the use of formal tools is divided into two categories. One is to develop a formal model fulfilling the control objects and then to design an LD program according to this model. The other one is to design a given LD program and then to validate it by analyzing properties of its corresponding model. Timed automata [1], statecharts [2], sensor graphs [3], and Petri nets (PNs) are emerging as effective formal tools to validate and verify LD programs.

Manuscript received March 30, 2016; accepted December 5, 2016. Date of publication March 14, 2017; date of current version June 14, 2018. This work was supported by the National Natural Science Foundation of China under Grant 61573158. A prior version of this paper was presented at the 2012 IEEE Conference on Decision and Control, Maui, HI, USA. This paper was recommended by Associate Editor S. Tong. *(Corresponding author: MengChu Zhou.)*

J. Luo and Q. Zhang are with the Department of Control Science and Engineering, Huaqiao University, Xiamen 361021, China (e-mail: jlluo@hqu.edu.cn; luojiliang77@gmail.com).

X. Chen is with Fujian Electrical Power Technical College, Quanzhou 362000, China (e-mail: 33201268@qq.com).

M. Zhou is with the Institute of Systems Engineering, Macau University of Science and Technology, Macau 999078, China and also with the Helen and John C. Hartmann Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102 USA (e-mail: zhou@njit.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2016.2647219

The researchers have verified PLC programs by a model checking method. Ngalamou and Myers [4] extracted a control process by using an Extensible Markup Language (XML) model and generating a symbolic model verifier (SMV) code from an XML model. Pavlovic and Ehrich [5] performed a process of verifying a function block diagram with the NuSMV model checker. In [6] and [7], the race, where some outputs change from scan to scan under fixed inputs, is addressed. It is a threatening condition since, if a race occurs, then a corresponding relay tends to be destroyed, tremendous waste may be caused, and human safety may be menaced. The work [6] uses constraint-based program analysis techniques to prove whether there exists any flaw in a program in the standard meta-language. The work [7] puts forward the formula for verifying the race by using linear temporal logic. However, the translation algorithm between LD programs and automata is lacking, and mathematical foundation and rich-experience about model checking are required for the programmers if these methods are used. Like the model checking method, the above methods all suffer from the state space explosion of a PLC system.

PNs attract more and more researchers since they are well applicable to model and analyze various discrete event systems. A large body of their supervisory control theories [8]–[19] is well-developed to design a logic controller to prevent them from reaching illegal states. Peng and Zhou [20] survyed the PN and LD-based methods and their conversion for the controller design from specifications. They emphasize that PNs are an effective tool to establish a more flexible and understandable system than LDs do, since PNs are more easily modifiable and, hence, maintainable than LDs. Venkatesh *et al.* [21] introduced a real timed PN, and propose an efficient method to design a sequential PLC controller based on Real-Time PNs. The other useful methods for designing PLC programs based on PNs are proposed in [20] and [22]–[32]. Giua and Seatzu [33] modeled railway networks with PNs so as to apply the theory of supervisory control to automatically design their system controller. Hsieh and Kang [34] presented a control-based PN modeling method that translated a high-level AGV system flow path net into a low-level ON/OFF control model systematically. Bender *et al.* [7] performed the translation by using timed PNs with read-arcs. Despite the clear description of a working process, the reachability analysis of timed PNs with read-arcs is of high computational complexity. The work [35] translates LD programs into a class of extended PNs that have inhibitor arcs and some newly defined arcs. Wightkin *et al.* [36] translated sequential function chart (SFC) programs into timed PNs with flags indicating activities of steps in an SFC. Tsai and Teng [37] introduced a Boolean PN to model an LD program by attaching a Boolean equation with a transition, and representing each contact or coil by a place. Such nets may be small, but is too abstract to trace logic mistakes in programs. The use of Boolean equations greatly increases the analysis complexity of the net. The work [38] converted LD programs into colored PNs. Chen *et al.* [39] presented a method modeling an LD as an ordinary PN.

The main contributions of this paper with respect to the state of the art are summarized as follows.

1) A method is proposed to translate LDs into ordinary PNs instead of extended PNs as done in the published work via the introduced LD graph. Compared with the extended PN methods, an LD is described as graphically as possible by our method, and, consequently, our obtained model is more intuitional and explicit than existing methods [35], [38], which is useful for the analysis and verification of programs. Especially promising is that the theory of ordinary PNs is better developed than most of extended ones.
2) Only one-to-one mapping rules between simple instructions and PN structures are presented in the published methods, while our proposed algorithm is systemic since it can translate a whole LD into an ordinary PN. This makes it easier to design a software to automatically translate an LD into a PN.
3) A PN-based method is proposed to detect, locate and correct races in an LD. The PN structure can directly tell whether there is a relation between two instructions or two variables that need be considered for the existence of race. Consequently, through the PN structure, we can divide an LD into several parts that can be separately verified about the race. Evidently, this greatly reduce the number of states that need be checked, thereby avoiding state space explosion issues faced by the existing methods. As a result, this method is much more efficient than the others by verifying an LD as a whole.
4) A subnet related to a race can be singled out from the whole net by our method. It is useful to find the reason of this race by its structure that is very intuitional. This gives engineers a good opportunity to correct such race issues easily.

The rest of this paper is organized as follows. Section II introduces LDs, graphs, and PNs. Section III defines the LD graph for an LD program, and presents a translation algorithm from an LD graph to an ordinary PN, and an algorithm for reducing a PN structure. Section IV presents a method to detect, locate, and correct races in an LD. Section V concludes this paper.

II. Preliminaries

A. Ladder Diagram

IEC61131-3 standard [40] defines four kinds of programming languages for PLCs, which are instruction lists, structured texts, LDs, and function block diagrams.

For an LD program, an input variable represents the states of a sensor or switch, which is periodically sampled and input into a PLC system; an output variable represents the states of an actuator such as a motor or relay, which is periodically output from a PLC to control an actuator; and an internal variable does not represent any external equipment including sensors and actuators, but is used for the computation of an LD only.

An LD consists of six kinds of elements that are the left power rail, right power rail, contact, coil, horizonal line, and vertical line. The left and right power rails are represented as |. There are two kinds of static contacts that are the normally open contact represented as | | and the normally closed contact, |/|. They are often used to denote the input, output, and internal variables in a PLC system. There are two kinds of momentary coils that are the coil represented by ( ), and negated coil, (/), and there are also two kinds of latched coils that are the set coil, (*S*), and the reset coil, (*R*). They are often used to denote internal and output variables. Graphically, an LD is a collection of rungs between the left and right power rails. Each rung is a set of contacts and coils that are connected by horizonal and vertical lines. The contact and coil, which denote the same Boolean variable, can appear in the same rung. In this paper, coils are supposed to be directly connected with the right power rail.

The execution of a PLC keeps periodical scanning. In each scan, an LD program is executed in the similar manner of a relay electrical connect chart. The power flows from top to down, and from left to right, and consequently, each output is computed and refreshed.

Fig. 1(a) shows a dosing tank, which is a part of a chemical plant. This system is equipped with three liquid level float senors $S1$–$S3$, starting and stopping buttons $PB1$ and $PB2$, two feeding valves A and B, a discharging valve C, and a stirring motor M. The internal relay $M0$ controls the whole system to startup or stop. The level of liquid in the tank is detected by sensors $S1$–$S3$ to indicate the tank's liquid levels, i.e., empty, half and full, respectively. When the tank is empty, valve A is opened and the first kind of liquid is filled into the tank. When the tank is half with the first kind of liquid, the second kind of liquid is filled into the tank as soon as valve B is opened. When the tank is full, valves A and B are closed. The stirring motor is turned on when the second kind of liquid has been filled into it. Finally, the discharging valve C is opened after the liquid has been stirred and closed until the level of liquid is lower than $S1$.

Fig. 1(b) shows the LD program, where contacts $I0$–$I4$ capture the signals from $PB1$, $PB2$, $S1$, $S2$, and $S3$, respectively, the coils $Q0$–$Q3$ control valves A–C and motor M through outputting brake signals, respectively, and $M0$ is an internal variable.

B. Graph

A graph is a two-tuple $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$, where $\mathcal{V} = \{v_1, \dots, v_n\}$, $n \in \mathbb{Z}$, is a finite, nonempty set of nodes, called the vertex set; $\mathcal{E} = \{e_1, e_2, \dots, e_m\}, m \in \mathbb{Z}$ is a set of edges, where

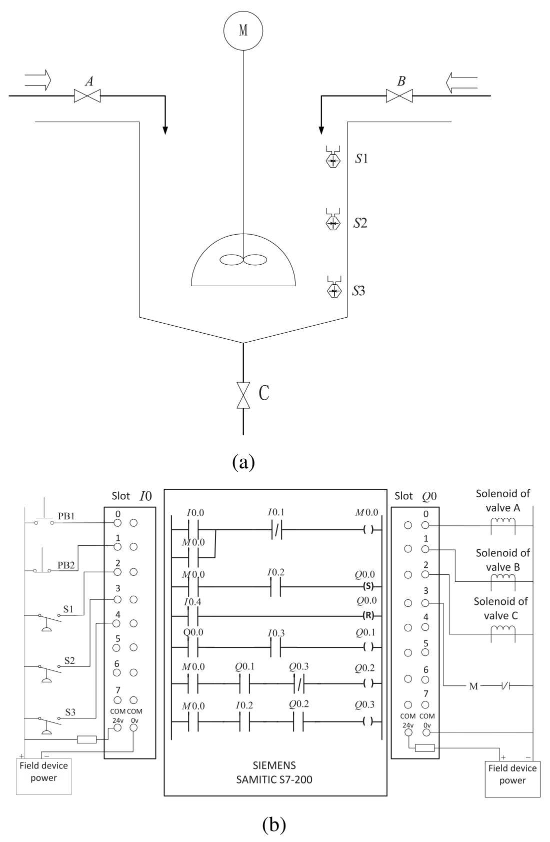


Fig. 1. (a) Dosing tank and (b) its PLC program.

$e_i = \langle v_j, v_t\rangle$, $i, j, t \in \mathbb{Z}$, $\forall 1 \le i \le m, v_j, v_t \in \mathcal{V}$, is a pair denoting the edge between v_j and v_t.

A node sequence in which there is an edge between any two adjacent nodes is called a path. A path is called a simple path if each node appears at most once in it. It is supposed that each path is a simple path in this paper.

C. Ordinary Petri Nets

An ordinary PN [41] is a three-tuple $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F})$, where $\mathcal{P}$ (places) and $\mathcal{T}$ (transitions) are finite, nonempty and disjoint sets representing the nodes, and $\mathcal{F} \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ is a set of directed arcs connecting nodes. A marking m is a column vector indicating a distribution of tokens among the places. The number of tokens in place p is denoted by $m(p)$. m_0 is the initial marking. $(\mathcal{N}, m_0)$ is called a net system or marked net.

Given two nodes $x, y \in \mathcal{P} \cup \mathcal{T}$, if $(x, y) \in \mathcal{F}$, x is the input of y, and y is the output of x. ${}^\bullet x$ and $x^\bullet$ represent the set of inputs and the set of outputs of node x, respectively. If $\forall p \in {}^\bullet t$, $m(p) \ge \mathcal{W}(p, t)$, then t is enabled; this is denoted by $m[t\rangle$. If t is enabled at marking m, then it can fire, and a new marking m' is reached such that each of its input places loses a token and each of its output places receives a token if it fires. This is denoted as $m[t\rangle\, m'$.

If a sequence of transitions can fire from m and results in m', then we say that m' is reachable from m. The PN is said to be reversible if m_0 is reachable from all markings that are reachable from m_0. The PN is said to be live if $\forall t \in \mathcal{T}$, under any reachable marking from m_0, can be made enabled by firing some sequence of transitions. A place p and a transition t make up a self-loop if p is both the input and output place of t, i.e., $p \in {}^\bullet t \cap t^\bullet$.

A P-invariant stands for a set of places which the weighted sum of tokens is constant for all markings reachable from any given initial marking.

III. Translating LD Programs Into Ordinary PNs

A. LD Graph

Definition 1: Let G be the graph obtained by removing the left and right power rails from an LD. If a connected subgraph of G is not contained by any other connected subgraph of G, then it is called a rung of the LD.

Definition 2: Given an LD with $n \in \mathbb{Z}^+ = \{1, 2, \ldots\}$ rungs and $q \in \mathbb{Z}^+$ coils, the LD graph is defined as $\mathcal{G}_{LD} = \langle \mathcal{V}, \mathcal{E}\rangle$ that meets the following conditions.

1) $\mathcal{V} = \mathcal{V}_L \cup \mathcal{V}_R \cup \mathcal{V}_v$ is the vertex set such that $\mathcal{V}_L = \{v_{l1}, \ldots, v_{ln}\}$ where v_{li}, $1 \le i \le n$, represents the junction of the left power rail and the ith rung in the LD; $\mathcal{V}_R = \{v_{r1}, \ldots, v_{rn}\}$ where v_{ri}, $1 \le i \le n$, represents the junction of the right power rail and the ith rung in the LD; and $\mathcal{V}_v$ is the set of vertices such that v_{s_1,s_2,s_3} is its element iff the following conditions are met.
   a) There exists a contact or coil whose name is s_1 that is a character string variable.
   b) If $s_2 = 0$, then there exists a contact named by s_1 that is normally open; if $s_2 = 1$, then there exists a contact named by s_1 that is normally closed; if $s_2 = 2$, then there exists a coil named by s_1; if $s_2 = 3$, then there exists a negated coil named by s_1; if $s_2 = 4$, then there exists a set coil named by s_1; and if $s_2 = 5$, then there exists a reset coil named by s_1.
   c) If $s_3 = 0$, then there exists a contact named by s_1; if $s_3 = j, (1 \le j \le q)$, then there exists a coil named by s_1 and this coil is the jth coil from the top to bottom.
2) $\langle v_j, v_k\rangle$, $\forall v_j, v_k \in \mathcal{V}$, is an element of $\mathcal{E}$ if there is a link line between two nodes in the LD that are corresponding to v_j and v_k, respectively.

In Definition 2, the implications of the subscripts of a vertex are summarized in Table I.

Example 1: For the LD shown in Fig. 1(b), its LD graph $\mathcal{G}_{LD} = \langle \mathcal{V}, \mathcal{E}\rangle$, shown in Fig. 2, is obtained according to Definition 2. The vertex set $\mathcal{V}$ includes $\mathcal{V}_L = \{v_{l1}, v_{l2}, v_{l3}, v_{l4}, v_{l5}, v_{l6}\}$, $\mathcal{V}_R = \{v_{r1}, v_{r2}, v_{r3}, v_{r4}, v_{r5}, v_{r6}\}$ and $\mathcal{V}_v$ in which the vertices are converted from the symbols of LD according to Definition 2. For example, $v_{I0.0,0,0}$ represents the variable $I0.0$ that appears as a normally open contact in LD. Table II shows that the vertices in $\mathcal{V}_v$ and what they represent based on their subscripts.

B. Translation Algorithm From LD Graph to PN

After an LD is expressed by an LD graph, we design Algorithm 1 for translating an LD graph to an ordinary PN.

TABLE I
IMPLICATIONS OF A VERTEX'S SUBSCRIPTS

	value	implication
s_1	character string variable	the name of the coil or contact represented by this vertex
s_2	0	a normally open contact
	1	a normally closed contact
	2	a coil
	3	a negated coil
	4	a set coil
	5	a reset coil
s_3	0	a contact
	$1 \le j \le q$	the jth coil from top to down

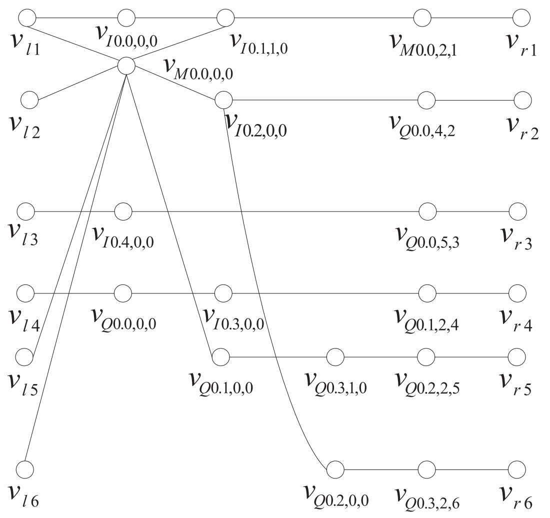


Fig. 2. LD graph of Fig. 1(b).

TABLE II
VERTICES OF THE LD GRAPH OBTAINED IN EXAMPLE 1

nodes	s_1	s_2	s_3
$v_{I0.0,0,0}$	$I0.0$	normally open	contact
$v_{I0.1,1,0}$	$I0.1$	normally closed	contact
$v_{I0.2,0,0}$	$I0.2$	normally open	contact
$v_{I0.3,0,0}$	$I0.3$	normally open	contact
$v_{I0.4,0,0}$	$I0.4$	normally open	contact
$v_{M0.0,0,0}$	$M0.0$	normally open	contact
$v_{M0.0,2,1}$		coil	the first coil
$v_{Q0.0,0,0}$	$Q0.0$	normally open	contact
$v_{Q0.0,4,2}$		set coil	the second coil
$v_{Q0.0,5,3}$		reset coil	the third coil
$v_{Q0.1,0,0}$	$Q0.1$	normally open	contact
$v_{Q0.1,2,4}$		coil	the fourth coil
$v_{Q0.2,0,0}$	$Q0.2$	normally open	contact
$v_{Q0.2,2,5}$		coil	the fifth coil
$v_{Q0.3,1,0}$	$Q0.3$	normally closed	contact
$v_{Q0.3,2,6}$		coil	the sixth coil

The translation is performed according to the following rules. First, each variable, which represents a contact or coil, is modeled by a pair of places. One place that is marked represents that this variable is **ON**(1), and the other place that is marked represents that this variable is **OFF**(0). Second, if a variable is an input variable for a PLC which corresponds to a contact, then two transitions are used to connect its two places such that these transitions and places make up a directed circuit, and to model that this contact turns **ON** and **OFF**, respectively. Third, for an output variable, which corresponds to a coil adjacent to the right power rail, several transitions are introduced to represent the processes of this coil turning **ON** or **OFF**. To define these transitions, we need the following definitions.

Definition 3: Given an LD graph, let $v_{s_1,s_2,i}$, $i \in \mathbb{Z}$, be a node corresponding to a coil connected with the right power rail v_{ri}, i.e., $2 \le s_2 \le 5$. Then:

1) a path from v_{li} to v_{ri} is called a rung path of $v_{s_1,s_2,i}$ if it contains $v_{s_1,s_2,i}$;
2) the path that is obtained by removing v_{li} and v_{ri} from a rung path of $v_{s_1,s_2,i}$ is called an instruction path of $v_{s_1,s_2,i}$;
3) the set of instruction paths of $v_{s_1,s_2,i}$ is denoted by $\Pi(v_{s_1,s_2,i})$.

Example 2 (Example 1 Continued): According to Definition 3, the instruction path set of $v_{M0.0,2,1}$ is

$$\Pi\big(v_{M0.0,2,1}\big) = \big\{\big(v_{I0.0,0,0}, v_{I0.1,1,0}, v_{M0.0,0,0}, v_{I0.1,1,0}\big)\big\}. \quad (1)$$

Similarly, we have

$$\begin{cases} \Pi\big(v_{Q0.0,4,2}\big) = \big\{\big(v_{M0.0,0,0}, v_{I0.2,0,0}\big)\big\} \\ \Pi\big(v_{Q0.0,5,3}\big) = \big\{\big(v_{I0.4,0,0}\big)\big\} \\ \Pi\big(v_{Q0.1,2,4}\big) = \big\{\big(v_{Q0.0,0,0}, v_{I0.3,0,0}\big)\big\} \\ \Pi\big(v_{Q0.2,2,5}\big) = \big\{\big(v_{M0.0,0,0}, v_{Q0.1,0,0}, v_{Q0.3,1,0}\big)\big\} \\ \Pi\big(v_{Q0.3,2,6}\big) = \big\{\big(v_{M0.0,0,0}, v_{Q0.2,0,0}, v_{I0.2,0,0}\big)\big\}. \end{cases} \quad (2)$$

According to the execution theory of LDs, it is the path $\pi \in \Pi(v)$, where each contact is on, that makes the coil v on. Therefore, such a path should be modeled by a transition that is enabled if each contact in it is on and v is off. This is just the reason that we introduce Definition 3.

Definition 4: Let $v_{s_1,s_2,s_3} \in \mathcal{V}_{Oi}$, $i \in \mathbb{Z}$, be an assigned-node that is not a set or reset coil, i.e., $s_3 \neq 4 \wedge s_3 \neq 5$, connected with v_{ri} in a given LD graph. For v_{li} and v_{s_1,s_2,s_3}, a cut-node set of v_{s_1,s_2,s_3}, denoted by ψ, is defined as a set of nodes that satisfies the following.

1) Each node in ψ is in a path between v_{li} and v_{s_1,s_2,s_3}.
2) If all nodes in ψ are deleted, then there does not exist a path between v_{li} and v_{s_1,s_2,s_3}.
3) ψ is not contained in any other cut-node set of v_{s_1,s_2,s_3}.

The set of the cut-node sets of v_{s_1,s_2,s_3} is denoted by $\Psi(v_{s_1,s_2,s_3})$, and called as its cut-set.

Example 3 (Example 2 Continued): From Definition 4, the cut-sets of $v_{M0.0,2,1}$, $v_{Q0.1,2,4}$, $v_{Q0.2,2,5}$, and $v_{Q0.3,2,6}$ are

$$\begin{cases} \Psi\big(v_{M0.0,2,1}\big) = \big\{\psi_{1,1}, \psi_{1,2}\big\} \\ \Psi\big(v_{Q0.1,2,4}\big) = \big\{\psi_{4,1}, \psi_{4,2}\big\} \\ \Psi\big(v_{Q0.2,2,5}\big) = \big\{\psi_{5,1}, \psi_{5,2}, \psi_{5,3}\big\} \\ \Psi\big(v_{Q0.3,2,6}\big) = \big\{\psi_{6,1}, \psi_{6,2}, \psi_{6,3}\big\} \end{cases} \quad (3)$$

where

$$\begin{cases} \psi_{1,1} = \{v_{I0.1,1,0}\}, \psi_{1,2} = \{v_{I0.0,0,0}, v_{M0.0,0,0}\} \\ \psi_{4,1} = \{v_{Q0.0,0,0}\}, \psi_{4,2} = \{v_{I0.3,0,0}\} \\ \psi_{5,1} = \{v_{M0.0,0,0}\}, \psi_{5,2} = \{v_{Q0.1,0,0}\} \\ \psi_{5,3} = \{v_{Q0.3,1,0}\}, \\ \psi_{6,1} = \{v_{M0.0,0,0}\}, \psi_{6,2} = \{v_{Q0.2,0,0}\} \\ \psi_{6,3} = \{v_{I0.2,0,0}\}. \end{cases} \tag{4}$$

There is not a cut-node set of $v_{Q0.0,4,2}$ or $v_{Q0.0,5,3}$ since they are a set and reset coil, respectively.

In an LD graph, it is a cut-node set where each contact is off that can cut all the paths between the left power rail and a coil v, adjacent to the right power rail, and consequently, coil v turns off. Therefore, a transition is used to model such a cut-node set, and its firing is controlled by the nodes in this set. According to Definition 4, if an assigned-node v_{s_1,s_2,s_3} represents a set or reset coil, i.e., $s_2 = 4 \vee s_2 = 5$, then its cut-node sets should be ignored. The reason is that a set or reset coil remains on even if all the contacts or coils connected to it turn off.

Algorithm 1 is proposed to convert an LD graph into an ordinary PN. It has the following five parts. The first one, from steps 2–13, adds a pair of places for each variable whose name is just the first subscript of a node in the given LD graph, and marks one of these two places. The second one, from steps 14–15, for each rung, finds the assigned-node set. The third one, steps 16–25, for each node v in the assigned-node set, finds the instruction path set of v, $\Pi(v)$, according to Definition 3, and for each instruction path $\pi \in \Pi(v)$, adds a transition from which there is an arc to the place representing that this assigned-node is on, and to which there is an arc from the place representing that this assigned-node is off. Furthermore, steps 26–33 adds a bidirectional arc between this transition and the place representing that each node in this instruction path is on. The fourth one, steps 34–41, finds all cut-sets of v, i.e., $\Psi(v)$. For each cut-node set $\psi \in \Psi(v)$, it designs a transition from which there is an arc to the place representing that this assigned-node is on, and to which there is an arc from the place representing that this assigned-node is off. The fifth one, steps 42–51, adds a bidirectional arc between this transition and the place representing that each node in this cut-node set is off. So far, the input or output transitions of the places, which are corresponding to the output variables representing the coils, have been designed. Finally, according to the properties of contacts, steps 52–57, search every pair of places without any input or output, which are corresponding to an input variable, and add two transitions such that they can make up a loop with this pair of places.

Example 4 (Example 3 Continued): By Algorithm 1, the LD graph shown in Fig. 2 is translated to the PN shown in Fig. 3. Places $p_{I0.0,0}$ and $p_{I0.0,1}$, $p_{I0.1,0}$ and $p_{I0.1,1}$, $p_{I0.2,0}$ and $p_{I0.3,1}$, $p_{I0.4,0}$ and $p_{I0.4,1}$, $p_{M0.0,0}$ and $p_{M0.0,1}$, $p_{Q0.0,0}$ and $p_{Q0.0,1}$, $p_{Q0.1,0}$ and $p_{Q0.2,1}$, and $p_{Q0.3,0}$ and $p_{Q0.3,1}$, represent OFF and ON states of variable $I0.1$, $I0.2$, $I0.3$, $I0.4$, $M0.0$, $Q0.0$, $Q0.1$, $Q0.2$, and $Q0.3$, respectively. From expressions (1), (2), and (4), $t_{1,1,1}$ and $t_{1,2,1}$ model the instruction

Algorithm 1 Translating an LD Graph Into an Ordinary PN

```
Input: $\mathcal{G}_{\mathcal{LD}} = \langle \mathcal{V}, \mathcal{E} \rangle$ with $n$ rungs and $q$ coils, $n, q \in \mathbb{Z}$, where
       $\mathcal{V} = \mathcal{V}_L \cup \mathcal{V}_R \cup \mathcal{V}_v$
Output: $(\mathcal{N}, m_0)$, $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F})$
1: $\mathcal{P} = \emptyset, \mathcal{T} = \emptyset, \mathcal{F} = \emptyset$;
2: for all $v_{s_1,s_2,s_3} \in \mathcal{V}_v$ do
3:   if $p_{\bar{s}_1} \notin \mathcal{P}$ then
4:     $\mathcal{P} = \mathcal{P} \cup \{p_{\bar{s}_1}, p_{s_1}\}$;
5:   end if
6: end for
7: for all $p_{\bar{s}_1}, p_{s_1} \in \mathcal{P}$ do
8:   if $\exists v_{x,s_2,s_3} \in \mathcal{V}_v \wedge x = s_1 \wedge (s_2 = 1 \vee s_2 = 3)$ then
9:     $m_0(p_{\bar{s}_1}) = 1, m_0(p_{s_1}) = 0$;
10:  else
11:    $m_0(p_{\bar{s}_1}) = 0, m_0(p_{s_1}) = 1$;
12:  end if
13: end for
14: for $(j = 1$ to $n)$ do
15:   $\exists v_{s_1,s,j} \in \mathcal{V}, 2 \le s \le 5$;
16:   $\Pi(v_{s_1,s,j}) = \{\pi_1, \pi_2, \cdots, \pi_K\}$, $K \in \mathbb{Z}$, according to
      Definition 3;
17:   for all $1 \le k \le K$ do
18:     $\mathcal{T} = \mathcal{T} \cup \{t_{j,k,1}\}$;
19:     if $s = 3 \vee s = 5$ then
20:       $\mathcal{F} = \mathcal{F} \cup \{(p_{s_1}, t_{j,k,1}), (t_{j,k,1}, p_{\bar{s}_1})\}$;
21:     else
22:       $\mathcal{F} = \mathcal{F} \cup \{(p_{\bar{s}_1}, t_{j,k,1}), (t_{j,k,1}, p_{s_1})\}$;
23:     end if
24:     for all $v_{a,s_2,s_3} \in \pi_k$ do
25:       if $s_2 = 0$ then
26:         $\mathcal{F} = \mathcal{F} \cup \{(p_{\bar{a}}, t_{j,k,1}), (t_{j,k,1}, p_a)\}$;
27:       else
28:         $\mathcal{F} = \mathcal{F} \cup \{(p_a, t_{j,k,1}), (t_{j,k,1}, p_{\bar{a}})\}$;
29:       end if
30:     end for
31:     $\Psi(v_{s_1,s,j}) = \{\psi_1, \psi_2, \cdots, \psi_K\}$, $K \in \mathbb{Z}$, according to
        Definition 4;
32:     for all $1 \le k \le K$ do
33:       $\mathcal{T} = \mathcal{T} \cup \{t_{j,k,0}\}$;
34:       if $s = 2$ then
35:         $\mathcal{F} = \mathcal{F} \cup \{(p_{s_1}, t_{j,k,0}), (t_{j,k,0}, p_{\bar{s}_1})\}$;
36:       else
37:         $\mathcal{F} = \mathcal{F} \cup \{(p_{\bar{s}_1}, t_{j,k,0}), (t_{j,k,0}, p_{s_1})\}$;
38:       end if
39:       for all $v_{a,s_2,s_3} \in \psi_k$ do
40:         if $s_2 = 0$ then
41:           $\mathcal{F} = \mathcal{F} \cup \{(p_{\bar{a}}, t_{j,k,0}), (t_{j,k,0}, p_{\bar{a}})\}$;
42:         else
43:           $\mathcal{F} = \mathcal{F} \cup \{(p_a, t_{j,k,0}), (t_{j,k,0}, p_a)\}$;
44:         end if
45:       end for
46:     end for
47:   end for
48: end for
49: for $p_{\bar{s}_1}, p_{s_1} \in \mathcal{P}$ do
50:   if $(p_{\bar{s}_1}^{\bullet} \cap^{\bullet} p_{s_1} = \emptyset) \wedge (p_{s_1}^{\bullet} \cap^{\bullet} p_{\bar{s}_1} = \emptyset)$ then
51:     $\mathcal{T} = \mathcal{T} \cup \{t_{s_1}, t_{\bar{s}_1}\}$;
52:     $\mathcal{F} = \mathcal{F} \cup \{(p_{s_1}, t_{\bar{s}_1}), (t_{\bar{s}_1}, p_{\bar{s}_1}),$— $(p_{\bar{s}_1}, t_{s_1}), (t_{s_1}, p_{s_1})\}$;
53:   end if
54: end for
```

paths $\pi_{1,1}$ and $\pi_{1,2}$, respectively. Similarly, $t_{2,1,1}$, $t_{3,1,1}$, $t_{4,1,1}$, $t_{5,1,1}$, and $t_{6,1,1}$ model $\pi_{2,1}$, $\pi_{3,1}$, $\pi_{4,1}$, $\pi_{5,1}$, and $\pi_{6,1}$. The cut-node sets $\psi_{1,1}$, $\psi_{1,2}$, $\psi_{4,1}$, $\psi_{4,2}$, $\psi_{5,1}$, $\psi_{5,2}$, $\psi_{5,3}$, $\psi_{6,1}$, $\psi_{6,2}$, and $\psi_{6,3}$ are modeled by $t_{1,1,0}$, $t_{1,2,0}$, $t_{4,1,0}$, $t_{4,2,0}$, $t_{5,1,0}$, $t_{5,2,0}$, $t_{5,3,0}$, $t_{6,1,0}$, $t_{6,2,0}$, and $t_{6,3,0}$.

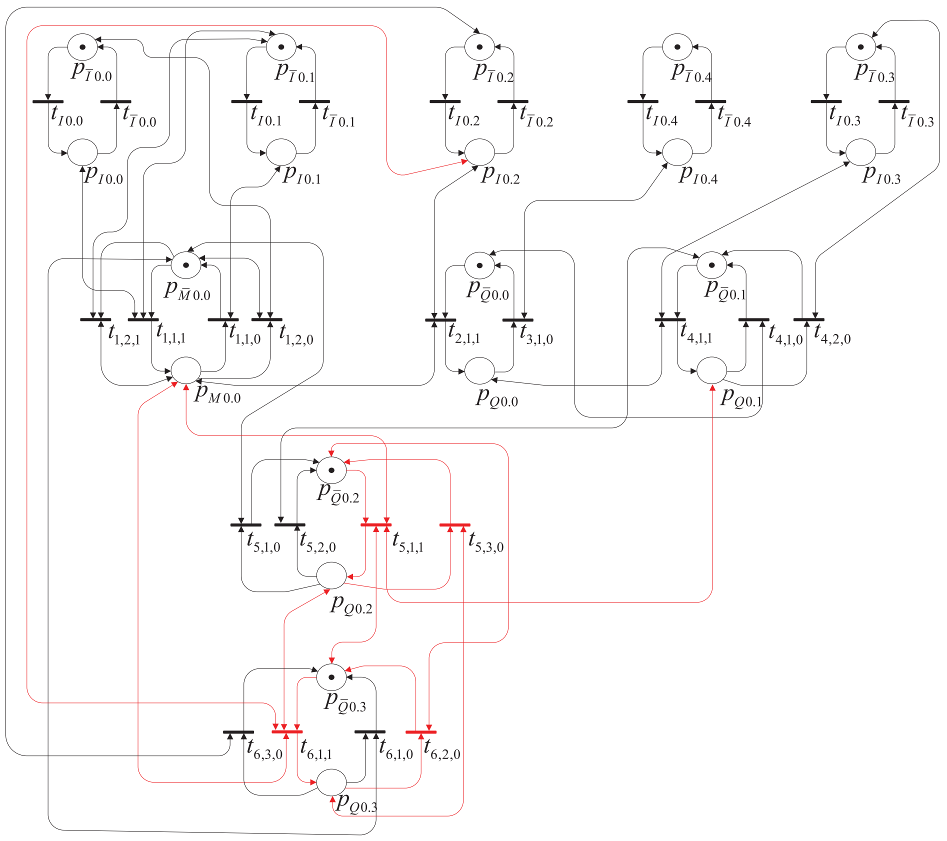


Fig. 3. PN model of the LD program in Fig. 1(b).

In a PN obtained by Algorithm 1, there may exist some transitions that are disabled all the time. For example, $t_{1,2,1}$ and $t_{1,2,0}$ in Fig. 3 are such cases. To remove such meaningless transitions, we need to classify the transitions into two types.

Definition 5: Let $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F})$ be a PN model of an LD program. A transition $t \in \mathcal{T}$ is called a sensing-transition if it has only one input place and only one output place, which are a pair of places corresponding to an input variable in the LD. Otherwise, it is a computing transition. $\mathcal{T}_s$ and $\mathcal{T}_c$ denote the sets of sensing and computing-transitions, respectively.

Example 5 (Example 4 Continued): In the PN model, which is shown in Fig. 3, the set of sensing-transitions is

$$\mathcal{T}_s = \{t_{\bar{I}0.0}, t_{I0.0}, t_{\bar{I}0.1}, t_{I0.1}, t_{\bar{I}0.2}, t_{I0.2}, t_{\bar{I}0.3}, t_{I0.3}, t_{\bar{I}0.4}, t_{I0.4}\}$$

and the set of computing-transitions is

$$\begin{aligned}\mathcal{T}_c = \{&t_{1,1,1}, t_{1,2,1}, t_{2,1,1}, t_{3,1,1}, t_{4,1,1}, t_{5,1,1},\\ &t_{6,1,1}, t_{1,1,0}, t_{1,2,0}, t_{4,1,0}, t_{4,2,0}, t_{5,1,0},\\ &t_{5,2,0}, t_{5,3,0}, t_{6,1,0}, t_{6,2,0}, t_{6,3,0}\}.\end{aligned}$$

Lemma 1: $(\mathcal{N}, m_0)$ is the PN model obtained by Algorithm 1 for an LD graph $\mathcal{G}_{LD}$. If $p_{\bar{s}_1}$ and p_{s_1} are a pair of places for v_{s_1,s_2,s_3} in $\mathcal{G}_{LD}$, then they constitute a P-invariant.

Proof: According to Algorithm 1, if a transition is an output of $p_{\bar{s}_1}$ or p_{s_1}, then it is also an input of $p_{\bar{s}_1}$ or p_{s_1}. Clearly, the firing of any transition does not change the sum of tokens in them. Therefore, $p_{\bar{s}_1}$ and p_{s_1} constitute a P-invariant. ■

The redundant transitions are partly caused by the self-holding instructions in an LD. In the LD graph of such self-holding instructions, an assigned-node corresponding to an output variable (a coil) is also in an instruction path (or a cut-node set) of another node corresponding to this same variable. According to Algorithm 1, the pair of places for this output variable are both inputs of the transition that models this instruction path (or a cut-node set). Consequently, a redundant transition appears.

Definition 6: Let $(\mathcal{N}, m_0)$ be a PN model of an LD program. For a computing-transition, $t \in \mathcal{T}_c$, if there exist a pair of places that are corresponding to the same variable in the LD and are both the input and output places of t, then it is called a redundant transition.

Algorithm 2 is proposed to remove redundant transitions, where N_v is used to represent the set of names of contacts and coils.

Example 6: From Definition 6, we can figure out that transitions $t_{1,2,1}$ and $t_{1,2,0}$ in the PN model, as shown in Fig. 3, are redundant. According to Algorithm 2, they and their arcs should be deleted. The reduced PN model is shown in Fig. 4.

Theorem 1: Let $(\mathcal{N}, m_0)$ be a PN model of an LD program. If a reduced PN model $(\mathcal{N}', m_0)$ is obtained by removing the redundant transitions from $(\mathcal{N}, m_0)$ using Algorithm 2, then the reachability graph of the reduced PN model is the same as that of the original PN model $(\mathcal{N}, m_0)$.

Proof: From Definition 6, there exists a pair of places that are input places of a redundant transition and are

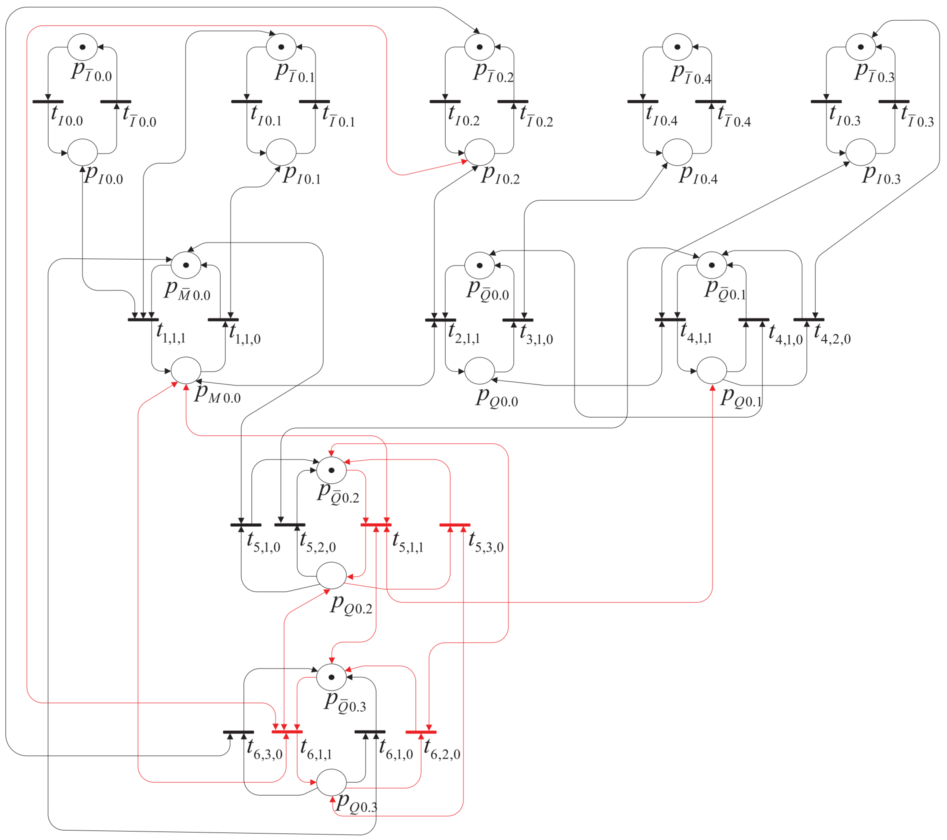


Fig. 4. PN obtained by reducing the PN in Fig. 3 by Algorithm 2.

Algorithm 2 Reducing Redundant Structures of PNs

```
Input: $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F})$, $N_v$ /*a set of the names of all variables in the LD program*/
Output: $\mathcal{N}$
1: From Definition 5, obtain the set of computing-transitions $\mathcal{T}_c$;
2: for all $t \in \mathcal{T}_c$ do
3:     if $\exists s_1 \in N_v, (p_{\bar{s}_1} \in^{\bullet} t) \wedge (p_{s_1} \in^{\bullet} t)$ then
4:         $\mathcal{T} = \mathcal{T} - \{t\}$;
5:         $\mathcal{F} = \mathcal{F} - \cup_{\forall p \in \mathcal{P}} \{(p, t), (t, p)\}$;
6:     end if
7: end for
```

corresponding to the same variable of the LD. From Lemma 1, this pair of places make up a P-invariant. Therefore, the sum of tokens in this pair of places is always 1 since it is 1 at the initial marking from Algorithm 1. Consequently, a redundant transition will never fire according to the firing rules of PN. Thus, the reduced PN model $(\mathcal{N}', m_0)$, which is obtained by deleting the redundant transitions in the PN model $(\mathcal{N}, m_0)$, has the same reachability graph as that of $(\mathcal{N}, m_0)$. ■

To analyze the computational complexity of the modeling method, we suppose that there are $x \in \mathbb{Z}$ contacts and $y \in \mathbb{Z}$ coils in an LD. From Algorithm 1, there are $2(x + y)$ places, $2x$ transitions corresponding to x contacts, and the number of transitions corresponding to coils is a polynomial function of the number of coils since it equals the sum of the number of

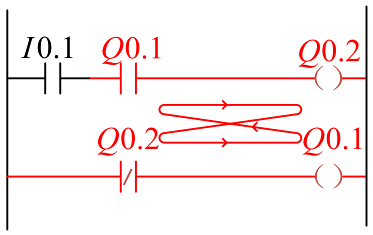


Fig. 5. Race LD.

paths and that of cut-sets for coils. Therefore, the size of PN model is a polynomial function of the numbers of contacts and coils while the sate space may increase exponentially. But this method cannot apply to LDs with timers.

IV. Detecting Races in LD Programs

A race in an LD program is an undesirable condition that causes that some outputs keep changing their values under fixed inputs. For example, as shown in Fig. 5. Evidently, if $I0.1$ is on, $Q0.1$ and $Q0.2$ make up a race since they set and reset each other cycle by cycle. If a race occurs, the devices or circuits become unstable, and may result in great loss. Therefore, it is of great importance to make sure PLC programs are free of race.

All execution processes of an LD can be obtained by using a reachability-graph technique since we have obtained its PN

Algorithm 3 Reachable Tree of an LD With K Rungs

```
Input: $\mathcal{N}, m_0$, which is the PN model of an LD with $K$ rungs
Output: Reachable tree $\mathcal{G}_{LD}$ of an LD
1: Denote $m_0$ as a circle in real line, and mark it as a new node;
2: while there exists a node denoted by a circle in real line and marked as a new node, do
3:    Select any new node $m$ that is denoted by a circle in solid line,
4:    for all $t \in \mathcal{T}_s$ do
5:       if $t$ is enabled at $m$ then
6:          $m'$ is the marking that is reached after $t$ fires;
7:          if there does not exist a marking denoted as a circle in solid line that equals to $m'$, then
8:             Denote $m'$ as a circle in solid line, and mark it as a new node;
9:          end if
10:      end if
11:   end for
12:   $m' = m$;
13:   $k = 1$;
14:   while $k \leq K$ do
15:      if There exists a transition $t$ in $\mathcal{T}_i$ enabled at $m'$ then
16:         $m''$ is the marking that is reached from $m'$ after $t$ fires;
17:         if $k \neq K$ then
18:            Draw a circle in dashed line to denote $m''$, a directed arc in dashed line from $m'$ to $m''$, and label this arc with $t$;
19:            $m' = m''$;
20:         else
21:            if There exists a node $\bar{m}$ denoted by a circle in solid line such that $\bar{m} = m$ then
22:               Draw a directed arc from $m'$ to $\bar{m}$, and label this arc with $t$;
23:            else
24:               Draw a circle in solid line to denote $m''$ and a directed arc from $m'$ to $m''$, and label this arc with $t$;
25:            end if
26:         end if
27:      end if
28:      $k = k + 1$;
29:   end while
30:   Mark $m$ as an old node;
31: end while
```

model, and can be used to detect races. However, when computing a reachability graph, additional firing-rules should be taken into account since an LD program is sequentially executed in each scanning period of PLCs. To do so, we need introduce several notations.

Definition 7: Let $\mathcal{N} = \{\mathcal{P}, \mathcal{T}, \mathcal{F}\}$ be the PN model of a given LD with $I \in \{1, 2, \ldots\}$ rungs. $\mathcal{T}_i$, $1 \leq i \leq I$, is the set of computing transitions corresponding to the ith rung in the LD, i.e., the set of transitions whose first subscript equals i.

According to Definitions 5 and 7, the transition set is divided into the set of sensing transitions $\mathcal{T}_s$, and K sets of computing transitions which are $\mathcal{T}_i$, $1 \leq i \leq K$. Consequently, $\mathcal{T} = \mathcal{T}_s \cup \mathcal{T}_c = \mathcal{T}_s \cup \mathcal{T}_1 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_K$.

Algorithm 3 is designed according to the reachability-graph algorithm of ordinary PNs and the execution characteristics of a PLC, i.e., a PLC executes in cycles, and each cycle includes the input-scan where inputs are read into the PLC memory, logic-scan where instructions are consequently executed to operate the input data and results (outputs) are saved in temporary memory, and output-scan where outputs are updated according to temporary memory. It is to compute a graph describing all possible executing processes of a PLC, which is called the PLC r-graph. It has one outer loop and two inner loops. The first inner loop is to address the state-involving process in the input-scans, while the second one does so in the logic-scans. Evidently, a directed arc in solid line denotes a transition from one state to another due to the change of an input, while a directed arc in dashed line is that due to an instruction execution. As is also straightforward that a node denoted by circle in dashed-line is a state in the logic-scans, and one denoted by circle in solid line that is also the ending node of a directed arc denoted by dashed-line is a state in the output-scans.

TABLE III
NODES IN FIG. 6

Node	Marking
m_0	$(0,0,0,0,0,0,0,0,0,0)^T$
m_1	$(1,0,0,0,0,0,0,0,0,0)^T$
m_2	$(1,0,0,0,0,1,0,0,0,0)^T$
m_3	$(1,0,1,0,0,1,0,0,0,0)^T$
m_4	$(1,0,1,0,0,1,1,0,0,0)^T$
m_5	$(1,0,1,1,0,1,1,0,0,0)^T$
m_6	$(1,0,1,1,0,1,1,1,0,0)^T$
m_7	$(1,0,1,1,0,1,1,1,1,0)^T$
m_8	$(1,0,1,1,0,1,1,1,1,1)^T$
m_9	$(1,0,1,1,0,1,1,1,0,1)^T$
m_{10}	$(1,0,1,1,1,1,1,1,1,1)^T$
m_{11}	$(1,0,1,1,1,1,0,1,1,1)^T$
m_{12}	$(1,0,1,1,1,1,1,0,1,1)^T$
m_{13}	$(1,0,1,1,0,1,1,1,0,1)^T$
m_{14}	$(1,0,1,1,1,1,0,0,0,0)^T$
m_{15}	$(1,0,1,1,1,1,0,0,0,0)^T$
m_{16}	$(1,0,1,1,1,1,0,0,0,0)^T$
m_{17}	$(1,0,1,1,1,1,0,0,0,0)^T$
m_{18}	$(1,0,1,0,0,1,1,0,1,1)^T$

Example 7: The PLC r-graph of the LD shown in Fig. 2 can be computed via Algorithm 3, as partially illustrated in Fig. 5 since it is too large to be completely shown. The marking (state) corresponding to each node can be found in Table III.

Definition 8: Given a PLC r-graph of an LD, a node in solid line, x, is called a racing node if there exists an arc in dashed line pointing to x and one in dashed line leaving from x.

For example, m_5, m_8, m_{14}, and m_{17} in Fig. 5 are racing nodes according to Definition 8.

Definition 9: In a PLC r-graph of an LD, a sequence of nodes is called a race path if:

1) there is a directed arc in dashed line from one node to next one;
2) each node appears once;
3) it has two racing nodes; and
4) it is a circuit.

For example, $m_{16}m_5m_6m_7m_8$ in Fig. 5 is a race path according to Definition 9.

Theorem 2: There is a race in an LD iff there exists a race path in the PLC r-graph of its PN model $(\mathcal{N}, m_0)$.

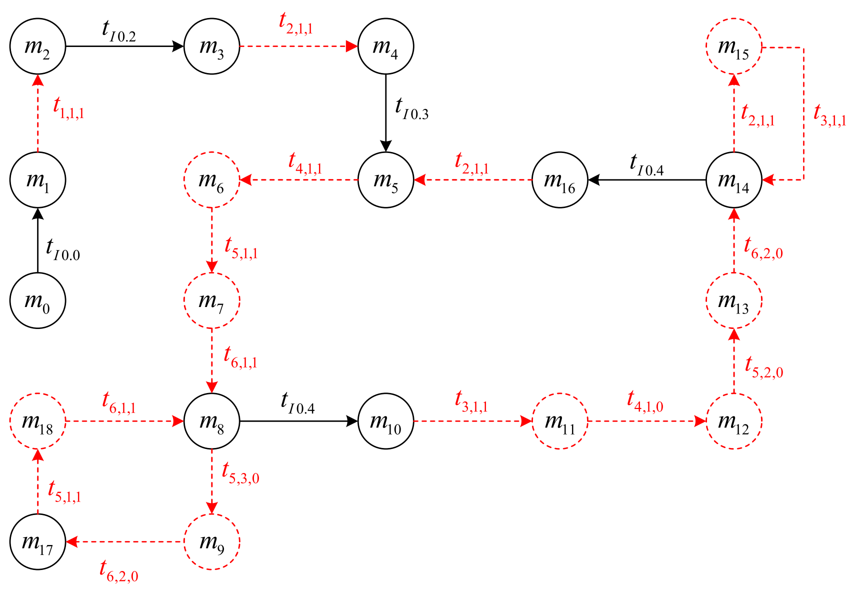


Fig. 6. Part of the PLC reachable-graph of the LD in Example 1.

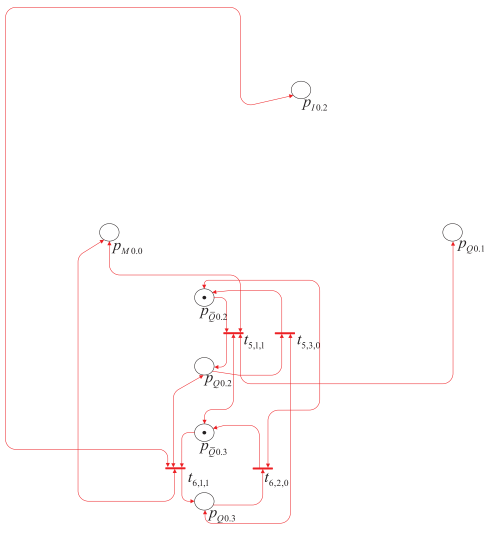
Proof: (if) If there exists a race path π in the PLC r-graph, there are at least two racing nodes x_1 and x_2 in it according to Definition 9. Each one of its arcs is in a dashed line, and denotes a computing transition from Definition 9 and Algorithm 3. This implies that no input changes when PLC executes along π. However, x_1 and x_2 are in solid lines from Definition 8, and denote two states with different outputs in output-scans, respectively. Therefore, outputs changes while inputs are constants. This means there is a race if there exists a race path π in the PLC r-graph.

(only if) If there is a race in this LD, then outputs keep changing even if all input variables are fixed. From Algorithm 3, any execution process of the PLC can be represented by a path in the PLC r-graph. Therefore, there exists a path in the PLC r-graph to represent the process where this race occurs. Since there exist at least two states under which there are different outputs, there are at least two racing nodes in this path from Definition 8. Further, since the inputs are fixed in this process, each arc in this path denotes a computing transition, and, hence, is in a dashed line. This means this path is a race path from Definition 9. Therefore, there exists a race path in the PLC r-graph if there is a race. ■

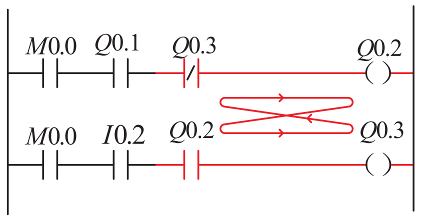
From Theorem 2, we can detect if there exists a race in an LD, and locate it if so by finding a race path.

The procedure to detect and locate races in a given LD is designed as follows.

1) Obtain the LD graph for a given LD according to Definitions 1 and 2.
2) Obtain the PN model for the LD graph by Algorithm 1.
3) Reduce the PN model by Algorithm 2.
4) Compute the PLC r-graph by entering the PN model into Algorithm 3 as its inputs.
5) According to Definition 9, search race paths in the PLC r-graph. This procedure ends, if none is found; go to next step, otherwise.
6) For each race path, determine the subnet of the PN model that contains it, and, in turn, locate the rungs of instructions in the LD that contain the race via this subnet. Furthermore, find the reason of this race by analyzing this subnet.



(a)



(b)

Fig. 7. Location of the race. (a) Subnet for the race path. (b) Instructions contain the race.

Next, the LD shown in Fig. 1(b) is taken as an example to illustrate this procedure. By Step 1, the LD graph, as shown in

Fig. 2, is obtained according to Definitions 1 and 2. Steps 2–4 are done in Examples 3, 6, and 7, respectively. In Step 5, we can find a race path, which is $m_8m_9m_{17}m_{18}$ shown in Fig. 6. This means that there is a race in the LD program. By the sequence of transitions $t_{5,3,0}t_{6,2,0}t_{5,1,1}t_{6,1,1}$ along this race path, we can obtain the subnet that just contains this race, which is shown in Fig. 7(a). This subnet tells that $Q0.2$ and $Q0.3$ keep changing from cycle to cycle when $I0.2$ and $Q0.1$ are fixed as 1, and this race lies in the fifth and sixth rungs of the LD, which is shown in Fig. 7(b). Further, from the structural analysis of this subnet, we can conclude that the reason of this race is that $Q0.2$ and $Q0.3$ are used to alter their values by each other. Therefore, this race mistake can be corrected by removing $Q0.2$ from the sixth rung or $Q0.3$ from the fifth rung.

V. Conclusion

By the proposed modeling method, the graphical, intuitional and explicit structures of ordinary PNs are obtained to express the abstract execution of an LD program, and, consequently, the analysis, simulation and verification of LDs according to the theory of ordinary PNs are possible. Furthermore, we show how to detect, locate and correct races in an LD via ordinary PN structures.

In the future, we intend to compress the size of the PN model via various reduction methods [42], to get rid of race directly in PN structures via the techniques in [29] and [43] and to investigate the ways for completing the translation algorithm for more complex instructions. Besides, we will investigate more efficient methods to verify LD programs using PNs and other model-checking tools [44]–[49].

References

[1] M. Zhou, F. He, M. Gu, and X. Song, "Translation-based model checking for PLC programs," in *Proc. IEEE Int. Comput. Softw. Appl. Conf.*, Seattle, WA, USA, Jul. 2009, pp. 553–562.

[2] S. Seidel, T. Klotz, U. Donath, and J. Haufe, "Modelling the real-time behaviour of machine controls using UML statecharts," in *Proc. IEEE Conf. Emerg. Technol. Factory Autom. (ETFA)*, Bilbao, Spain, Sep. 2010, pp. 1–8.

[3] T. Alenljung, B. Lennartson, and M. N. Hosseini, "Sensor graphs for discrete event modeling applied to formal verification of PLCs," *IEEE Trans. Control Syst. Technol.*, vol. 20, no. 6, pp. 1506–1521, Nov. 2012.

[4] L. Ngalamou and L. Myers, "Combining software methods for effective deployment of programmable logic controllers (PLCs)," *Int. J. Comput. Sci. Netw. Security*, vol. 10, no. 12, pp. 134–146, Dec. 2010.

[5] O. Pavlovic and H.-D. Ehrich, "Model checking PLC software written in function block diagram," in *Proc. Int. Conf. Softw. Test. Verification Validation*, Los Alamitos, CA, USA, Apr. 2010, pp. 439–448.

[6] A. Aiken, M. Fähndrich, and Z. Su, "Detecting races in relay ladder logic programs," *Int. J. Softw. Tools Technol. Transf.*, vol. 3, no. 1, pp. 93–105, 2000.

[7] D. F. Bender *et al.*, "Ladder metamodeling and PLC program validation through time Petri nets," in *Proc. Eur. Conf. Model Driven Archit. Found. Appl.*, Berlin, Germany, Jun. 2008, pp. 121–136.

[8] L. E. Holloway, B. H. Krogh, and A. Giua, "A survey of Petri net methods for controlled discrete event system," *Discr. Event Dyn. Syst. Theory Appl.*, vol. 7, no. 2, pp. 151–190, 1997.

[9] M. V. Iordache and P. J. Antsaklis, "Supervision based on place invariants: A survey," *Discr. Event Dyn. Syst.*, vol. 16, no. 4, pp. 451–492, Dec. 2006.

[10] J. Luo, W. Wu, H. Su, and J. Chu, "Supervisor synthesis for enforcing a class of generalized mutual exclusion constraints on Petri nets," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 39, no. 6, pp. 1237–1246, Nov. 2009.

[11] J. Luo and K. Nonami, "Approach for transforming linear constraints on Petri nets," *IEEE Trans. Autom. Control*, vol. 56, no. 12, pp. 2751–2765, Dec. 2011.

[12] J. Luo, H. Shao, K. Nonami, and F. Jin, "Maximally permissive supervisor synthesis based on a new constraint transformation method," *Automatica*, vol. 48, no. 6, pp. 1097–1101, Jun. 2012.

[13] J. Luo, H. Ni, and M. Zhou, "Control program design for automated guided vehicle systems via Petri nets," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 1, pp. 44–55, Jan. 2015.

[14] J. Luo, H. Ni, W. Wu, S. Wang, and M. Zhou, "Simultaneous reduction of Petri nets and linear constraints for efficient supervisor synthesis," *IEEE Trans. Autom. Control*, vol. 60, no. 1, pp. 88–103, Jan. 2015.

[15] F. Basile, P. Chiacchio, and A. Giua, "Suboptimal supervisory control of Petri nets in presence of uncontrollable transitions via monitor places," *Automatica*, vol. 42, no. 6, pp. 995–1004, Jun. 2006.

[16] Z. Li and M. Zhou, "Control of elementary and dependent siphons in Petri nets and their application," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 38, no. 1, pp. 133–148, Jan. 2008.

[17] J. O. Moody and P. J. Antsaklis, "Petri net supervisors for DES with uncontrollable and unobservable transitions," *IEEE Trans. Autom. Control*, vol. 45, no. 3, pp. 462–476, Mar. 2000.

[18] H. Hu, Y. Liu, and M. Zhou, "Maximally permissive distributed control of large scale automated manufacturing systems modeled with Petri nets," *IEEE Trans. Control Syst. Technol.*, vol. 23, no. 5, pp. 2026–2034, Sep. 2015.

[19] H. Hu and M. Zhou, "A Petri net-based discrete-event control of automated manufacturing systems with assembly operations," *IEEE Trans. Control Syst. Technol.*, vol. 23, no. 2, pp. 513–524, Mar. 2015.

[20] S. S. Peng and M. C. Zhou, "Ladder diagram and Petri-net-based discrete-event control design methods," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 34, no. 4, pp. 524–531, Nov. 2004.

[21] K. Venkatesh, M. Zhou, and R. J. Caudill, "Comparing ladder logic diagrams and Petri nets for sequence controller design through a discrete manufacturing system," *IEEE Trans. Ind. Electron.*, vol. 41, no. 6, pp. 611–619, Dec. 1994.

[22] M. C. Zhou and E. Twiss, "Design of industrial automated systems via relay ladder logic programming and Petri nets," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 28, no. 1, pp. 137–150, Feb. 1998.

[23] S. Peng and M. Zhou, "Sensor-based stage Petri net modeling of PLC logic programs for discrete-event control design," *Int. J. Prod. Res.*, vol. 41, no. 3, pp. 629–644, Feb. 2003.

[24] M. Uzam, A. H. Jones, and N. Ajlouni, "Conversion of Petri net controllers for manufacturing systems into ladder logic diagrams," in *Proc. IEEE Conf. Emerg. Technol. Factory Autom.*, Kauai, HI, USA, Nov. 1996, pp. 649–655.

[25] M. Uzam and A. H. Jones, "Discrete event control system design using automation Petri nets and their ladder diagram implementation," *Int. J. Adv. Manuf. Syst. Special Issue Petri Nets Appl. Manuf. Syst.*, vol. 14, no. 10, pp. 716–728, Oct. 1998.

[26] M. Uzam, A. H. Jones, and I. Yücel, "Using a Petri-net-based approach for the real-time supervisory control of an experimental manufacturing system," *Int. J. Adv. Manuf. Technol.*, vol. 16, no. 7, pp. 498–515, 2000.

[27] G. Frey, "Automatic implementation of Petri net based control algorithms on PLC," in *Proc. Amer. Control Conf.*, Chicago, IL, USA, Jun. 2000, pp. 648–655.

[28] T. Suesut, P. Inban, P. Nilas, P. Rerngreun, and S. Gulphanich, "Interpretation Petri net model to IEC 1131-3: Ld for programmable logic controller," in *Proc. IEEE Conf. Robot. Autom. Mechatronics*, Singapore, Dec. 2004, pp. 1107–1111.

[29] G.-B. Lee, H. Zandong, and J. S. Lee, "Automatic generation of ladder diagram with control Petri net," *J. Intell. Manuf.*, vol. 15, no. 2, pp. 245–252, 2004.

[30] J.-S. Lee and P.-L. Hsu, "A systematic approach for the sequence controller design in manufacturing systems," *Int. J. Adv. Manuf. Technol.*, vol. 25, no. 7, pp. 754–760, 2005.

[31] D. Thapa, S. Dangol, and G.-N. Wang, "Transformation from Petri nets model to programmable logic controller using one-to-one mapping technique," in *Proc. Int. Conf. Comput. Intell. Model. Control Autom.*, Vienna, Austria, Nov. 2005, pp. 228–233.

[32] T. Perme, "Translation of extended Petri net model into ladder diagram and simulation with PLC," *J. Mech. Eng.*, vol. 55, no. 10, pp. 608–622, Sep. 2009.

[33] A. Giua and C. Seatzu, "Modeling and supervisory control of railway networks using Petri nets," *IEEE Trans. Autom. Sci. Eng.*, vol. 5, no. 3, pp. 431–445, Jul. 2008.

[34] S. Hsieh and M.-Y. Kang, "Developing AGVS Petri net control models from flowpath nets," *J. Manuf. Syst.*, vol. 17, no. 4, pp. 237–250, 1998.

[35] J. Lee and J. S. Lee, "Conversion of ladder diagram to Petri net using module synthesis technique," *Int. J. Model. Simulat.*, vol. 29, no. 1, pp. 79–88, Jan. 2009.

[36] N. Wightkin, U. Buy, and H. Darabi, "Formal modeling of sequential function charts with time Petri nets," *IEEE Trans. Control Syst. Technol.*, vol. 19, no. 2, pp. 455–464, Mar. 2011.

[37] J.-I. Tsai and C.-C. Teng, "Constructing an abstract model for ladder diagram diagnosis using Petri nets," *Asian J. Control*, vol. 12, no. 3, pp. 309–322, May 2010.

[38] E. A. da Silva Oliveira, L. D. da Silva, K. Gorgônio, A. Perkusich, and A. F. Martins, "Obtaining formal models from ladder diagrams," in *Proc. IEEE Int. Conf. Ind. Inform.*, Lisbon, Portugal, 2011, pp. 796–801.

[39] X. Chen, J. Luo, and P. Qi, "Method for translating ladder diagrams to ordinary Petri nets," in *Proc. IEEE Conf. Decis. Control*, vol. 45. Dec. 2012, pp. 6716–6721.

[40] *International Standard 61131-3 Programmable Controllers—Part 3: Programming Languages*, IEC Standard 61131-3, Jan. 2003.

[41] Z. W. Li and M. C. Zhou, *Deadlock Resolution in Automated Manufacturing Systems: A Novel Petri Net Approach*. New York, NY, USA: Springer, 2009.

[42] J. Li, M. C. Zhou, and X. Dai, "Algebraic reduction and refinement of Petri nets," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 42, no. 5, pp. 1244–1255, Sep. 2012.

[43] M. V. Moreira, D. S. Botelho, and J. C. Basilio, "Ladder diagram implementation of control interpreted Petri nets: A state equation approach," in *Proc. 4th IFAC Workshop Discr. Event Syst. Design*, vol. 42. Gandia, Spain, Oct. 2009, pp. 78–83.

[44] Y. Shi, C. Tian, Z. Duan, and M. Zhou, "Model checking Petri nets with MSVL," *Inf. Sci.*, vol. 363, pp. 274–291, 2016.

[45] W. Yu, C. Yan, Z. Ding, C. Jiang, and M. C. Zhou, "Modeling and verification of online shopping business processes by considering malicious behavior patterns," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 2, pp. 647–662, Apr. 2016.

[46] Z. Ding, Y. Zhou, M. Jiang, and M. C. Zhou "A new class of Petri nets for modeling and property verification of switched stochastic systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 7, pp. 1087–1100, Jul. 2015.

[47] Z. Ding, C. Jiang, and M. C. Zhou, "Design, analysis and verification of real-time systems based on time Petri net refinement," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 1, pp. 1–18, Jan. 2013.

[48] G. Liu, W. Reisig, C. Jiang, and M. C. Zhou, "A branching-process-based method to check soundness of workflow systems," *IEEE Access*, vol. 4, pp. 4104–4118, Aug. 2016.

[49] S. G. Wang, M. D. Gan, M. C. Zhou, and D. You, "A reduced reachability tree for a class of unbounded Petri nets," *IEEE/CAA J. Automatica Sinica*, vol. 2, no. 4, pp. 353–360, Oct. 2015.

Jiliang Luo (M'14) received the B.E. and M.S. degrees in thermal power engineering from Northeast Dianli University, Jilin, China, in 2000 and 2003, respectively, and the Ph.D. degree in control science and engineering from Zhejiang University, Hangzhou, China, in 2006.

He joined Huaqiao University, Xiamen, China, in 2006, where he is currently a Professor of Control Science and Engineering and the Vice Dean of the College of Information Science and Engineering. He was a Visiting Researcher with Chiba University, Chiba, Japan, from 2009 to 2010 and the New Jersey Institute of Technology, Newark, NJ, USA, from 2014 to 2015. His current research interests include supervisory control of discrete event systems, Petri nets, programmable logic controller, and intelligent manufacturing systems and robots.

Dr. Luo was a recipient of Ho-Pan-Qing-Qi Award of 2012 and *Asian Journal of Control's* Outstanding Reviewer of 2011.

Qi Zhang received the B.S. degree in electronics engineering from Huaqiao University, Xiamen, China, in 2014, where she is currently pursuing her master's degree in electrical engineering.

Her research interests include Petri net theory and application, programmable logic controllers (PLCs), and the design of PLC program.

Xuekun Chen received the B.S. degree in electrical engineering from North China Electric Power University, Beijing, China, in 2010, and the M.S. degree in electrical engineering from Huaqiao University, Xiamen, China, in 2013.

She is currently an Lecturer with Fujian Electrical Power Technical College, Quanzhou, China. Her research interests include Petri net theory and application, programmable logic controllers, and distribution automation systems.

MengChu Zhou (S'88–M'90-SM'93–F'03) received the B.S. degree in control engineering from the Nanjing University of Science and Technology, Nanjing, China in 1983, and the M.S. degree in automatic control from the Beijing Institute of Technology, Beijing, China in 1986, and the Ph.D. degree in computer and systems engineering from Rensselaer Polytechnic Institute, Troy, NY, USA, in 1990.

He joined the New Jersey Institute of Technology, Newark, NJ, USA, in 1990, and he is currently a Distinguished Professor of Electrical and Computer Engineering. He has over 700 publications including 12 books, over 360 journal papers (over 260 in IEEE TRANSACTIONS), and 28 book-chapters. His current research interests include Petri nets, Internet of Things, big data, Web services, manufacturing, transportation, and energy systems.

Dr. Zhou was a recipient of the Humboldt Research Award for U.S. Senior Scientists, the Franklin V. Taylor Memorial Award, and the Norbert Wiener Award from the IEEE Systems, Man and Cybernetics Society. He is a Founding Editor of IEEE Press Book Series on Systems Science and Engineering. He is a Life Member of Chinese Association for Science and Technology-USA and served as its President in 1999. He is a Fellow of International Federation of Automatic Control and American Association for the Advancement of Science.