IMPLEMENTATION OF CONCURRENT CONTROL ALGORITHMS USING PLC LADDER DIAGRAMS

Liudmila Cheremisinova

Abstract: *The problem of mapping a concurrent control algorithm onto a program structure for Programmable Logic Controller is discussed. The systematic method to derive Ladder Diagram programs from a parallel automaton that is a functional model of logic control device is presented. The mapping process is decomposed into a sequence of optimizing transformations of mathematical models of a parallel control algorithm specified in a formal language PRALU. The suggested procedures of optimizing the mathematical models are based on providing the proper control for a specific object under the control, i.e. within the restricted domain.*

Keywords: *programmable logic controllers, parallel algorithms, control systems, state assignment.*

ACM Classification Keywords: *D. Software; D.1 PROGRAMMING TECHNIQUES; D.1.3 Concurrent Programming; D. Software; D.2 SOFTWARE ENGINEERING; D.2.2 Design Tools and Techniques.*

Introduction

Sequential control allows processing sequential and parallel operations in a discrete mode with respect to time or events. It is used to coordinate different continuous functions and to control complex process sequences. The behavior of the control system under discussion is characterized by complex interaction, asynchronism and concurrency. The widespread case is considered when a complex requires control in which inputs and outputs are on/off signals. The functions of a control of such a system are concentrated in one block – logic control unit that should provide proper synchronization of interaction between the components. In recent years the use of Petri net formalism has been gaining popularity for abstract description of the behavior of concurrent systems [Karatkevich, 2007, 2015].

The success of the control of a multiple component system greatly depends on the efficiency of the synchronization among its processing elements. The functions of a control of such a system are concentrated in a logic control device that should provide a proper synchronization of interaction between the components. In order to represent clearly the interaction involved in concurrent engineering system it is necessary to describe formally its functional and structural properties. This is becoming the usual industrial way to represent the control logic on the logical control level.

There exist many languages for description of logical control algorithms suited for these purposes. They can be divided into two classes: the languages formalizing methods of description of system functionality applied in industrial practice and languages based on formal mathematical models [Cheremisinova, 2002]. Keeping a logical control algorithm representation in mind a special language PRALU [Zakrevskij, 1989, 1999] has been chosen for these purposes. PRALU language combines the best features of languages from the mentioned two groups: it is clear enough for a designer and at the same time it has its background in Petri net theory (expanded nets of free choice – EFC–nets investigated by Hack [Hack, 1972]). The language has means for representation of asynchronous, sequential and concurrent processes. One of the proposed standard forms of PRALU–algorithms released from technical details was named as a parallel automaton [Zakrevskij, 1984]. This form is well suited for the purposes of hardware implementation of PRALU–algorithms and can be easily got from an initial PRALU–algorithm by its transforming [Zakrevskij, 1999].

The programmable logic controllers (PLC) [Bolton, 2015] are being used in many places where some kind of control is needed to run a real-time technical system. PLC-s [PLC, 1993] are digitally operating electronic systems designed for the use in an industrial environment and they are now widely used in many technical "real world" applications such as complex petro-chemical plants, robotic centers, automobile production lines.

The problem of design of logic control devices for distributed discrete-event systems is considered. The widespread case is considered when a technical system requires control in which inputs and outputs are on/off signals. The problem of mapping a parallel control algorithm onto a program structure for PLC is discussed. We use the most popular way of PLC programming based on the use of Ladder Diagram (LD) language. The functional mathematical model of LD program is suggested and it is shown that at the heart of the suggested method the asynchronous hardware realization of parallel automaton lies. The paper presents a systematic method to derive LD programs from a parallel automaton that is a functional model of logic control device. The mapping process is decomposed into a sequence of optimizing transformations of mathematical models of a parallel control algorithm. The proposed optimization procedures are based on providing the proper control for a specific object under the control. This possibility for optimization arises when considering the control unit behavior together with the behavior of the controlled object.

Programmable Logic Controllers

PLC is a special form of microprocessor-based controller that uses a programmable memory to store instructions and to implement functions such as logic, sequencing, timing, counting and arithmetic in order to control machines and processes. PLC-s were invented to replace the sequential electro-

magnetic relay circuits. They work by looking at their inputs and depending upon their state, switching on/off their outputs. To get the desired behavior of a control device a designer has to write a program on some of the available languages that is then translated into a list of instructions that have one-to-one correspondence to each symbol of the language.

Many programming languages have been developed for programming PLCs. Each manufacturer has its own language. To improve the software used in PLCs, the International Electrotechnical Commission (IEC) has defined a standard for programming languages which is called IEC 1131-3 [PLC, 1993]. The IEC 1131-3 standard tries to bring together the languages from the PLC and the computing worlds. The following is a list of programming languages specified by this standard:

1) Ladder Diagram (LD) that has been developed to mimic relay logic;

2) Sequential Function Charts (SFC) that is similar (but much more powerful) to flowcharts, it was developed to accommodate the programming of more advanced systems;

3) Function Block Diagram (FBD) that represent PLC programs as connection of different function blocks;

4) Structured Text (ST) that is a textual (PASCAL or BASIC like) programming language;

5) Instruction List (IL) that is an assembly like language;

The first three languages are graphical ones and the last two are textual languages.

A very commonly used method of programming PLCs is based on the use of ladder diagrams [Bolton, 2015]. Writing a program is then equivalent to drawing a switching circuit. Each LD diagram is composed of two vertical lines representing the power rails. The power rails are connected as horizontal lines called as rungs of the ladder, between these two verticals. In drawing a ladder diagram, certain conventions are adopted:

1) The vertical lines of the diagram represent the power rails between which circuits are connected. The power flow is taken to be from the left-hand vertical across a rung.

2) Each rung on the ladder defines one operation in the control process.

3) A ladder diagram is read from left to right and from top to bottom.

4) When the PLC is in its run mode, the rungs of an LD program are executed until an end rung is reached. The procedure of going through all the rungs of the program is termed a cycle.

5) Each rung must start with one or more inputs and end with at least one output.

6) A particular device can appear in more than one rung of a ladder. A relay may switch on one or more devices.

PLC language instructions can be classified into two categories: logic instructions and block instructions. Practically all PLCs have the same set of logic instructions having one-to-one correspondence with coils and contacts of the relay circuit. Accordingly to that the program variables can be divided into logic (Boolean) (including input and output ones) and arithmetic variables. As we aimed to consider the control part of a technical system the set of logic instructions is of the main interest (the other operations are translated to logical ones when transforming a control algorithm into a parallel automaton [Cheremisinova, 1988]. Practically all PLCs has the same set of logic instructions, the typical set of logic instructions includes "examine if on (or off)", "set on (or off)", "set on (or off) and preserve". Table 1 shows the typical set of basic logic instructions (as in [Allen-Bredly, 1976]). An example of a LD-program rung is shown in Figure 1.

In the most basic form a PLC is a microprocessor-based controller that executes an application program by interpreting its instructions. When executing an program a PLC continuously repeats a single loop called a cycle which consist of 3 steps: sampling input signals, executing an program to update the PLC's internal registers and delivering new output signal.

We mentioned before that an LD program is a list of rungs. From a syntactic point of view we can decompose a rung into two parts: a front and a rear. A front corresponds to a logic formula over the rung inputs. An input can be normally open or close. The sequential and parallel connections of the inputs represent respectively conjunction and disjunction. So, the left part of the logic formula is a multilevel Boolean expression over AND, OR, NOT operations. A rear of a rung is one or more outputs connected in parallel. In a rear, an output can be preceded by a front circuit like in the example of Figure 1.

Table 1. Basic logic instructions of LD language

Name	Graphic symbol	Operation	Result of instruction: is TRUE if	Relay analog
Load	–| |–	Examine if on	Input is TRUE	Normally open contact
LoadNot	–|/|–	Examine if off	Input is FALSE	Normally closed contact
Out	–( )–	Set on	There is a path of TRUE instructions on the rung	Relay coil
OutBar	–(/)–	Set off	There is a path of FALSE instructions on the rung	Doesn't exists
Set	–(L)–	Set on and preserve	There is a path of TRUE instructions on the rung	Latching relay
Reset	–(U)–	Set off and preserve	There is a path of TRUE instructions on the rung	

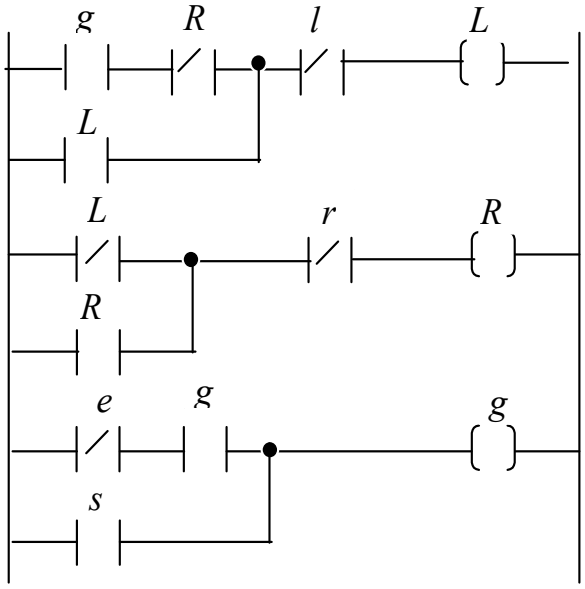

Figure 1. An example of LD-program

Ideally all rungs or as well as Boolean equations are independent and thus they should be evaluated simultaneously but in practice rungs are evaluated in a top-to-bottom order as they appear in a program. LD rung as well as a Boolean equation may be regarded as an if-then statement in programming languages. If-clauses are composed of logic instructions or compare instructions. Then-clauses are composed of output or arithmetic instructions. When a condition of an if-clause becomes true the corresponding then-clause is executed.

To answer the question what are the values of variables, defined by Boolean expressions represented by a system of Boolean equations or LD-program we have to think about the scanning sequence. The first thing in the scan cycle is to read the values of input signal from PLC sensors into the memory. Next, the PLC executes a program starting from the top left to bottom, changing the values of Boolean variables and using these new values when executing then-clauses following just after their setting.

Parallel and sequent automata

The process of a control algorithm mapping into PLC program can be understood as a sequence of transformations of mathematical models of the control PRALU-algorithm. When solving the problem of a control algorithm implementation one faces the necessity of getting first a formal finite-state model of the algorithm called as a parallel automaton [Zakrevskij, 1984]. This automaton model has structural input and output states and abstract internal ones. An essential difference of parallel automaton model from sequential one is that it can be in more than one partial state simultaneously. In that case the partial states are called parallel. All parallel partial states, a parallel automaton is in at some moment, form its

global state. Any transition of automaton defines changes of partial states that cause the current global state change. As well as a customary sequential automaton parallel automaton may be of Moore type or Mealy type.

For the purposes of PLC implementation of a control algorithm it is useful (though more difficult) to get a Moore type automaton. It is obtained by cutting long chains of the PRALU-algorithm on a set of strings (interpreted as automaton transitions) in the form:

$$\mu_n : -k_n^{in} \rightarrow \nu_n / k_n^{out},$$

where k_n^{in} and k_n^{out} are elementary conjunctions of Boolean variables, μ_n and ν_n are initial and terminal labels: subsets of natural numbers – states. The expression "$-k_n^{in}$" denotes the waiting operation of the PRALU-language: to wait until the term k_n^{in} takes the value 1. "$\rightarrow k_n^{out}$" denotes the acting operation: to give such values to the variables from conjunction k_n^{out} it to be equal to 1. k_n^{in} and k_n^{out} are interpreted as the input condition of the transition and the output signals respectively, μ_n and ν_n are interpreted as subsets of partial states s_i. Such a transition should be understood as follows: if the current global state of the parallel automaton contains all the partial states from μ_n and the variables in the conjunction term k_n^{in} assume values providing k_n^{in} = 1, then (as the result of the transition) the automaton goes to the next global state involving from initial one by substituting partial states from μ_n for the states from ν_n.

The values of inner and output variables of Moore type automaton are specified by partial states. Note that the conjunction k_n^{out} is divided into $|\nu_n|$ parts k_{ni}^{out}: each $k_{ni}^{out} \in k_n^{out}$ corresponds to some partial state $s_i \in \nu_n$. Taking into account the most used behavioral interpretation of output variables of initial PRALU-algorithm it can be said that automaton considered is inertial [Zakrevskij, 1999] over the set of its output variables, i.e. the variables that are absent in k_n^{out} preserve their values.

For instance, let us consider the process of transforming the following PRALU-algorithm (from [Zakrevskij, 1999]):

$$1: -u \rightarrow a\,b - \bar{u} \rightarrow 2.3$$

$$2: -\bar{v}w \rightarrow \bar{b}c - \bar{w} \rightarrow b\ \bar{c} \rightarrow 2$$

$$-v \rightarrow \bar{a}c \rightarrow 4.5$$

$$3: -uw \rightarrow d \rightarrow 6$$

$$4: -u \rightarrow a\ \bar{b} \rightarrow 7$$

$$5: -\bar{v}w \rightarrow \bar{c} \rightarrow 8$$

$$6.7.8: \bar{u}v \rightarrow \bar{a}\ \bar{d} \rightarrow 1$$

into a parallel automaton of Moore type:

$$1: |_1 - u\,|_9 \to a\,b - \overline{u} \to 2.3$$

$$2: |_2 - \overline{v}\,w\,|_{10} \to \overline{b}\,c - \overline{w}\,|_{11} \to b\ \overline{c} \to 2$$

$$- v\,|_{12} \to \overline{a}\,c \to 4.5$$

$$3: |_3 - u\,w\,|_{13} \to d \to 6$$

$$4: |_4 - u\,|_{14} \to a\ \overline{b} \to 7$$

$$5: |_5 - \overline{v}\,w\,|_{15} \to \overline{c} \to 8$$

$$6.7.8: \overline{u}\,v\,|_{16} \to \overline{a}\ \overline{d} \to 1$$

After simplifying the automaton and renumbering its partial states [Cheremisinova, 2002] we have:

u $\quad s_1 \to s_9 / a\,b$

$\overline{u}$ $\quad s_9 \to s_2 \cdot / b\ \overline{c}.\ s_3 /-$

$\overline{v}\,w$ $\quad s_2 \to s_{10} /\ \overline{b}\,c$

$\overline{w}$ $\quad s_{10} \to s_2 / b\ \overline{c}$

v $\quad s_2 \to s_4 /\ \overline{a}.\ s_5 / c$

$u\,w$ $\quad s_3 \to s_6 / d$

u $\quad s_4 \to s_7 / a\ \overline{b}$

$\overline{v}\,w$ $\quad s_5 \to s_8 /\ \overline{c}$

$\overline{u}\,v$ $\quad s_6 . s_7 . s_8 \to s_1 /\ \overline{a}\ \overline{d}$

Traditionally, the next step on the way to control device hardware implementation is state assignment. A peculiarity of this process for parallel automaton is that there are parallel states in it. It was suggested [Zakrevskij, 1989] to code partial states with ternary vectors which should be non-orthogonal for parallel partial states (but orthogonal for non-parallel). After encoding partial states an initial parallel algorithm can be transformed from its abstract form into a structural one – sequent automaton that can be directly hardware implemented.

A sequent automaton is a system of sequents [Zakrevskij, 2000] $f_i(V) \mid\!- k_i$, where $f_i(V)$ is a Boolean function over variables from $V = X \cup Y \cup Z$ and k_i is some elementary conjunction of variables from $W = Y \cup Z$, where X, Y and Z are sets of input, output and inner variables. Such an expression is specified semantically by the following manner: once the Boolean function $f_i(V)$ equals 1 then immediately after

that the variables from k_i should be set to be such that k_i equals 1. A set of values of all variables from V defines a total variable vector-state $\boldsymbol{v}$ of the sequent automaton. Below a sequent $f_i(V) \mid\!- k_i$ is called active on the state $\boldsymbol{v}^k$ if $f_i(\boldsymbol{v}^k)$ is true.

Further sequent automata inertial in relation to a subset $U \subseteq V$ will be considered. A sequent $f_i(V) \mid\!- k_i$ of such an automaton defines the following relationship on the set U: if the current state $\boldsymbol{v}^k$ of sequent automaton is such that some sequent is active on it then in the next state $\boldsymbol{v}^{k+1}$ all variables from U except for those mentioned in the conjunction k_i keep their values. Thus the variables from U keep their values, that have got, until any active sequent changes them. This property implies the interpretation of acting operations of the PRALU-language and forces to implement all output variables with flip-flops as well as inner variables.

Two types of sequent automata will be dealt with: a simple and a functional sequent automata [Zakrevskij, 1999]. The first one is a system of simple sequents $k_i' \mid\!- k_i''$. The functional sequent automaton consists of the sequents having the form $f_i^1(V) \mid\!- w_i$ or $f_i^0(V) \mid\!- \overline{w}_i$, where $w_i \in W$ and $f_i^1(f_i^0)$ are in the disjunctive normal form. In fact such an automaton is a system of Boolean equations. Having in view a Moore type automaton any its transition $\mu_n : -k_n^{in} \to \nu_n / k_n^{out}$ will be converted into $1 + |\nu_n|$ simple sequents, i.e. it generates the following sequents:

$$k_n^{in'} \wedge k^{\mu} \mid\!- k^{\nu}, \quad k^{\nu}{}_i \mid\!- k_{ni}^{out}, \ i = 1, 2, \ldots, |\nu_n|,$$

where k^{μ} and k^{ν} are conjunctive terms defining unions of compatible codes of partial states from the sets μ_n and ν_n respectively, $k^{\nu}{}_i$ and k_{ni}^{out} are conjunctive terms defining the code of the partial state $s_i \in \nu_n$ and output signals k_n^{out} implied by s_i.

Deriving LD realizable Boolean equations

As we focus our discussion on design of a control device let us consider a control part of the LD program. Omitting technical details it can be said that the control part of LD program is an ordered set of logic equations $F_i(V) = w_j^{\sigma}$, where $w_j^{\sigma} \in W = Y \cup Z$ ($w_j^{\sigma} = w_j$ when $\sigma = 1$ or $\overline{w}_j$ when $\sigma = 0$). $F_i(V)$ defines multilevel Boolean expression over AND, OR, NOT operations. Omitting brackets in the expressions $F_i(V)$ converts them into a sum of products form (disjunctive normal form). Thus a LD program can be represented syntactically by a functional sequent automaton.

When PLC executes a program the discipline of changing internal states should be maintained by input signals. In other words, there must be a stable state on W for any input state on X. In a similar, an asynchronous sequent automaton finding itself in a stable state $\boldsymbol{w}^k$ passes into another stable state $\boldsymbol{w}^{k+1}$ after changing an input state $\boldsymbol{x}^k$. A transition between stable states could be fulfilled as a result of execution of one or more active sequents. The execution order of sequents defines a variant of a

sequence of intermediate states $\boldsymbol{w}_i^{k+1}$ that take place when leaving $\boldsymbol{w}^k$ for $\boldsymbol{w}^{k+1}$. This order is random for asynchronous automata.

As to the LD program, Boolean equations (or LD-rungs) are executed in turn and the result of their execution is used in the next equations. Strictly speaking, the derived sequence of intermediate states is not random – one can find it for any given $\boldsymbol{w}^k$ and $\boldsymbol{x}^k$. But to provide an automaton to function properly the assumption, that the sequence of changes of inner variable values (as the result of input state modification) is purely arbitrary, is sufficient for proper modeling control algorithm by LD program. Thus the first and the basic step that should be done to convert parallel automaton into LD program is to synthesize an asynchronous inertial functional sequent automaton.

It is possible to synchronize changes of inner variables values during scanning the LD program by means of reduplication of inner variables. That is, one more inner variable z_i' is introduced for every variable $z_i \in Z$. Then z_i is used only in the left parts of Boolean equations, so it preserves invariably its value during the whole cycle of LD program scanning in top-to-bottom. z_i' is used only in the right parts of Boolean equations, so it can get new value in accordance with LD program flow. At the end of the LD program cycle z_i is set to have a value of z_i' and these new values of inner variables are used to set values of output variables. Such synchronization redoubles the number of inner variables but makes it possible to use state assignment methods developed for automaton synchronous implementation.

State assignment of asynchronous parallel automaton

When automaton asynchronous implementation is considered the additional condition has to be fulfilled to avoid the influence of races between memory elements during hardware operation. One of the ways to avoid them is to order memory elements switches so as to eliminate critical races [Cheremisinova, 2004]. The drawback of this approach seems to be the "density" of the codes obtained.

By using PLC for control device implementation it seems to be convenient to use a trivial method of encoding the partial states – 1-hot encoding. For example, such a coding had been used for PLC implementation of Graphset [Chevalier, 1980]. In this case the number of encoding variables is equal to the number of internal partial states of an automaton. The code of any partial state of a parallel automaton will have the only unit component. Any other component will be 0 if it corresponds to a partial state incompatible with considered one or don't care ("-") if it corresponds to a partial state parallel to considered one. But for some moment it might be more than one memory elements set to 1 due to existence of parallel partial states.

Considering a transition between two partial states it should be noted that it passes through one intermediate state encoded by the code with two unit components. This unstable state is alternated with

the force of the same input signal to resulting one. In general case a transition from n partial states to m ones takes place over $n + m$ -1 unstable states.

Having in mind an asynchronous implementation of parallel automaton it is desirable to agree on which of two output signals will be displayed during unstable states taking place in the transition $\mu_n \rightarrow \nu_n$: one implied by μ_n or the other one implied by ν_n. For distinctness let us agree that the output signals begin to change over their values in the state imposing ν_n immediately after switching on encoding variables of all partial states from ν_n. Then only those states can cause the output variable settings that are at the end of a LD program scan. So, if there exists an unstable state that takes place and that is changed within a scan cycle it cannot do necessary output variable setting. A partial state s_i is an unstable if there exist the following two transitions in the automaton:

$$\mu_n : -k_n^{in} \rightarrow \nu_n / k_n^{out}, \ \mu_p : -k_p^{in} \rightarrow \nu_p / k_p^{out} \ (s_i \in \nu_n, s_i \in \mu_p)$$

having compatible (nonorthogonal) input conditions k_n^{in} and k_p^{in}. If there exist such an unstable state the appropriate encoding variable should be doubled when implementing the automaton with LD program. Otherwise it is not necessary.

Sequent automaton mapping into LD program

Let us suppose that each partial state s_i of an automaton is encoded by inner variable z_i. And let the global state and the input state $\boldsymbol{x}_t$ of the automaton are such that the transition $t_n = \mu_n : -k_n^{in} \rightarrow \nu_n / k_n^{out}$ takes place. When converting a parallel automaton to functional sequent one it should be paid attention to following.

1. The inner variable z_i ($s_i \in \nu_n$) is switched on immediately after the transition t_n becomes active.

2. The inner variable z_j ($s_j \in \mu_n$) is switched off only after all inner variables z_i ($s_i \in \nu_n$) take the values 1.

3. The output variable y_p preserves its value implied by $s_i \in \mu_n$ until the automaton reaches at least one of the partial states from ν_n controlling y_p.

4. The output variable y_q switches the proper value implied by $s_j \in \nu_n$ immediately after reaching s_j.

For any variable $w_p \in W$ we form two functions f_p^1 and f_p^0, which present in fact its on- and off-functions. Let us choose for every partial state s_p sets T_p^{in} and T_p^{from} of all transitions into and from it respectively: $t_i = \mu_i : -k_i^{in} \rightarrow \nu_i / k_i^{out} \in T_p^{in}$ if $s_p \in \nu_i$ and $t_i \in T_p^{from}$ if $s_p \in \mu_i$. Then in accordance with what was said the functions f_p^1 and f_p^0 have the following general forms:

$$f_p^1 = \bigvee_{t_j \in T_p^{in}} (k_j^{in} \bigwedge_{s_i \in \mu_j} z_i) \quad f_p^0 = \bigvee_{t_j \in T_p^{from}} (\bigwedge_{s_i \in \nu_j} z_i) \,.$$

As to the output variables let us pick up all partial states implying the values 1 and 0 of the variable $y_q \in Y$ into the sets S_q^{on} and S_q^{off} respectively. Then the functions f_q^1 and f_q^0 will have the following forms:

$$f_q^1 = \bigvee_{s_i \in S_q^{on}} (z_i \overline{\bigvee_{s_r \in S^{iq}} z_r}) \quad f_q^0 = \bigvee_{s_i \in S_q^{off}} (z_i \overline{\bigvee_{s_r \in S^{iq}} z_r}) \,.$$

Here S^{iq} depends on s_i and y_q: $S^{iq} = \{s_r / (y_q \in C(s_r)) \,\&\, (s_r \in \nu_j) \,\&\, (t_j \in T_i^{from})$, where $C(s_r)$ denotes a set of variables controlled by the partial state s_r [Cheremisinova, 1988].

Minimization of the code length

The shortcoming of 1-hot state encoding is that the number of introduced inner variables is overmuch. But it seems to imply a fast program having rare rungs (or equations). At the same time it should be taken into account that the number of encoding variables can be considerably decreased (sometimes to 0) at the expense of using the values of some existing variables of an automaton as encoding ones. They can separate some incompatible partial states.

The set X of input variables of the control device can be divided into two subsets X_1 and X_2: those arriving from an outside environment (from human operator, for example) and those from the object under the control. The variables from X_1 can be considered as the inputs of the control system as a whole (with the controlled object) and, generally speaking, their behavior is unknown. The variables from X_2 are the internal variables for the control system. Their values depend on the object under the control. How the values of these variables will change, one can predict if the behaviour of the object under the engineering system control is known. PRALU-language allows to describe together the behavior algorithms of the object and the control device [Cheremisinova, 1989]. In such a description of the whole control system, variables from X_2 can be considered as inertial ones. Besides variables from X_2 output variables can be added to the set U of inertial variables. Just inertial variables are allowed to use as encoding for partial state assignment. That is permissible if their values are known in any automaton state controlling these variables. The conjunction of the values, which these variables have, when the automaton is in a partial state, can be used as a part of its code. For these purposes the conjunction terms k_n^{in} and k_n^{out} of a parallel automaton transitions $\mu_n : -k_n^{in} \to \nu_n / k_n^{out}$ are expanded up to the conjunctions k_n^{in*} and $k_n^{\nu*}$ of the generated sequent $k_n^{in*} \wedge k_n^{\mu} \vdash k_n^{\nu*}$. The method of calculating a set of all possible states on the set U includes defining [Cheremisinova, 1988*]:

1) concurrency relation on the set of partial states;

2) sets (non-overlapping for parallel partial states) of variables from U controlled by partial states;

3) set of eventual values of the variables from U that are controlled by every given partial state.

The first problem is defined [Cheremisinova, 2004] on a dynamic model of a parallel automaton (or a control algorithm) – its "skeleton" called as α-net [Zakrevskij, 1999]. When the concurrency relation has been found, for any partial state it can be said what partial states it is parallel to. A variable is called as "controlled" by a partial state s_k if its value can be changed when automaton finds itself in s_k and it may be changed in no partial state parallel to s_k. The subset of all such variables for a partial state s_k includes all variables under consideration except those changing their values in partial states parallel to s_k. For the considered above parallel automaton partial states s_2, s_4, s_5, s_7 and s_8 are parallel to s_3 and s_6; s_4 and s_7 – to s_5 and s_8. s_1 control all output variables {a, b, c, d}; s_3 and s_6 – the only d; s_4 and s_7 – {a, b}; s_5 and s_8 – the only c.

Total variable states are considered on a set U of variables the automaton is inertial in relation to it. Each possible global state $\boldsymbol{s_k}^*$ defines a set S_k^* of partial states and appropriate total variable state $\boldsymbol{u}_k^*$ on the set of variables controlled by partial states from S_k^*. This state $\boldsymbol{u}_k^*$ consist of non-orthogonal (by definition) variable states $\boldsymbol{u}_i$ corresponding to partial states $s_i \in S_k^*$. In [Cheremisinova, 1988] a concept of initial and final variable states of transitions of parallel automaton was introduced. An initial variable state $\boldsymbol{u}_n^b$ of a transition $t_n = \mu_n : -k_n^{in} \rightarrow \nu_n / k_n^{out}$ is a vector of values that might have the variables controlled by partial states $s_i \in \mu_n$ when that transition takes place. The final variable state $\boldsymbol{u}_n^e$ of the transition differs from appropriate initial one in values of variables from k_n^{out}.

An initial variable state $\boldsymbol{u}_n^b$ is a concatenation of component-vectors $\boldsymbol{u}_{nj}^b$ of values, variables controlled by partial states $s_j \in \mu_n$ might have. $\boldsymbol{u}_n^b$ implies each of $\boldsymbol{u}_{nj}^b$ (they are compatible). Each vector $\boldsymbol{u}_n^b$ component $\boldsymbol{u}_{nj}^b$ is defined by final variable states of transitions t_i: $\mu_i \rightarrow \nu_i$ to the partial state $s_{nj} \in \mu_i$ and $s_{nj} \in \nu_i$. The set of initial and final variable states of all automaton transitions gives the entire domain of the control system definition. If the variable state $\boldsymbol{u}_j$ implicating component-vectors $\boldsymbol{u}_{nj}^b$ of all initial variable states of all transitions t_n: $\mu_n \rightarrow \nu_n$, $s_j \in \mu_n$ is found, it may be considered as a part of the code of s_j on the set of variables controlled by it.

Let us consider the example of the reciprocating motion from [Allen-Breadly, 1976] (Figure 2). The object is to start out at the right hand of the table, activating the limit switch r. When the "start" pushbutton (the appropriate variable s) is pressed, right to left motion should occur until the limit switch l is tripped. Then left to right motion occurs until the limit switch r is tripped. This cycle should continue until the limit switch r is tripped again after the "stop" (the appropriate variable e) pushbutton is pressed. At this point motion will stop until "start" is pressed again. The output signals L and R of the control

device are introduced to cause right to left and left to right motions correspondingly, as shown in Figure 2.

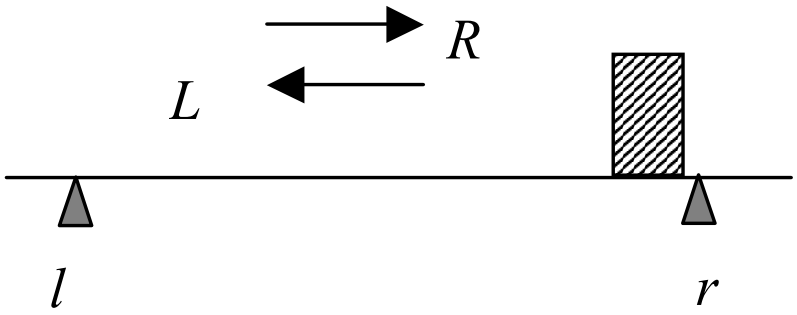

Figure 2. Example of reciprocating motion.

The following PRALU-algorithm describes the behavior of the control needed to provide such a reciprocating motion (the internal variable *g* is introduced to indicate the last of the pressings "start" or "stop"):

Motion(*s*, *e*, *l*, *r* / *L*, *R*)

1: $\rightarrow$ 2.3

2: $- g \rightarrow L - l \rightarrow \overline{L} \rightarrow R - r \rightarrow \overline{R} \rightarrow 2$

3: $- s \rightarrow g - e \rightarrow \overline{g} \rightarrow 3$

The following PRALU-algorithm describes the behavior of the whole control system:

Motion(*s*, *e*, *l*, *r* / *L*, *R*)

$\overline{l}\,\overline{r}\;\overline{g}\;\overline{L}\;\overline{R}$

1: $\rightarrow$ 2.3

2: $- g\;\overline{R}|_4 \rightarrow L \rightarrow \mathbf{r} \rightarrow \mathbf{l} - l|_5 \rightarrow \overline{L} - \overline{L}|_6 \rightarrow R \rightarrow \mathbf{l} \rightarrow \mathbf{r} - r|_2 \rightarrow \overline{R} \rightarrow 2$

3: $- s|_7 \rightarrow g - e|_3 \rightarrow \overline{g} \rightarrow 3$

Here $X = \{s, e, l, r\}$, $Y = \{L, R\}$, $Z = \{g\}$ are the sets of input, output and internal variables. $\overline{l}\,\overline{r}\;\overline{g}\;\overline{L}\;\overline{R}$ specifies the values of the variables, the automaton is inertial to, immediately before the motion starting. The part of PRALU-algorithm describing the behavior of the object of the control is set off in bold face. The waiting operations "$-\ \overline{L}$" and "$-\ \overline{R}$" (not changing the control algorithm) have been introduced to do the control algorithm as LD program realizable without doubling inner variables. The variables *l* and *r* define the behavior of the object of control and they are internal for a control system as a whole, so their

exhaustive behavior can be defined as well as for internal and output variables of the control device: *g*, *L* and *R*. Thus the variables from $U = \{I, r, g, L, R\}$ may be used as encoding ones for automaton partial states.

The first column of Table 2 shows the obtained parallel automaton of Moore type realizing the behavior of the control system. The sets of initial and final variable states of transitions of the automaton on the set *U* are given having in mind that the partial states from $\{s_2, s_4 - s_6\}$ and $\{s_3, s_7\}$ control the variables from $U_2 = \{I, r, L, R\}$ and $U_3 = \{g\}$ (the values of uncontrolled variables are marked as "+").

Table 2. Parallel automaton of Moore type

The automaton transitions	Initial states *I r g L R*	Final states *I r g L R*
t_1: s_2: $- g - \overline{R} \to s_4 \to L$	0 1 0 0 0	0 1 + 1 0
t_2: $\to \overline{r}$	**0 1 + 1 0**	**0 0 + 1 0**
t_3: $\to I$	**0 0 + 1 0**	**1 0 + 1 0**
t_4: s_4: $- I \to s_5 \to \overline{L}$	1 0 + 1 0	1 0 + 0 0
t_5: s_5: $- \overline{L} \to s_6 \to R$	1 0 + 0 0	1 0 + 0 1
t_6: $\to \overline{I}$	**1 0 + 0 1**	**0 0 + 0 1**
t_7: $\to r$	**0 0 + 0 1**	**0 1 + 0 1**
t_8: s_6: $- r \to s_2 \to \overline{R}$	0 1 + 0 1	0 1 + 0 0
t_9: s_3: $- s \to s_7 \to g$	+ + 0 + +	+ + 1 + +
t_{10}: s_7: $- e \to s_3 \to \overline{g}$	+ + 1 + +	+ + 0 + +

The partial states s_2, s_4, s_5 and s_6 (as well as s_3 and s_7) of the control automaton are pair-wise incompatible and their codes should be orthogonal. But we see that their initial states are parewise orthogonal with respect to variables from $\{I, r, L, R\}$ ($\{g\}$). So we need no more additional variables to encode the automaton partial states. So, the following functional sequent automaton realizing realizes the parallel automaton (shown in the first column of Table 2):

$$\overline{I}\, r\, g\ \overline{L}\ \overline{R} \mid\!- L$$

$$I\ \overline{r} L\ \overline{R} \mid\!- \overline{L}$$

$$I\ \overline{r}\ \overline{L}\ \overline{R} \mid\!- R$$

$$\overline{I} r\ \overline{L} R \mid\!- \overline{R}$$

$$s\ \overline{g} \mid\!- g$$

$$e\, g \mid\!- \overline{g}$$

As follows from the parallel automaton description (Table 2) there exist no unstable partial states. So implementing the sequent automaton with LD program there is no need to duplicate the variables *L* and *R* that are used as inner ones. Then taking into account the sequential mode of LD execution we can simplify the sequent automaton by discarding some inner variables from sequents:

$$\overline{I}\, g\ \overline{R} \mid\!- L$$

$$I \mid\!- \overline{L}$$

$$\overline{r}\ \overline{L} \mid\!- R$$

$$r \mid\!- \overline{R}$$

$$s \mid\!- g$$

$$e \mid\!- \overline{g}$$

As one could see from Table 1 any Boolean function can be realized by means of two LD instructions: a simple relay coil or latching relay (latch-unlatch). In the first case two sequents for the same variable (of a functional sequent automaton) is realized as a single rung of LD program, in the second case – as two rungs.

Taking into account that the sequent automaton is inertial one over inner and output variables we can realize any pair of sequents defining w_i and $\overline{w_i}$ by means of one of two LD instructions. That is Set-Reset (latching relay) instruction) or more preferable Out instruction (its relay coil analog). So, any pair of sequents $w_i = f_i^1$ and $w_i = f_i^0$ could be implemented as:

$$w_i = f_i^1 \vee w_i\ \overline{f_i^0} \text{ or } w_i = (f_i^1 \vee w_i)\ \overline{f_i^0}$$

The appropriate LD program is shown in Figure 1.

Conclusion

This paper presents the process of mapping control algorithm onto PLC program as a sequence of transformations of mathematical models of PRALU-algorithm. The task of minimization of PLC program (in the form of LD diagram) length is achieved as a result of solution of some optimization problems concerning transformations of PRALU-algorithm models.

Acknowledgement

The paper is published with partial support by the project ITHEA XXI of the ITHEA ISS (www.ithea.org) and the ADUIS (www.aduis.com.ua).

Bibliography

[Karatkevich, 2007] A.Karatkevich. Dynamic Analysis of Petri Net-Based Discrete Systems. In: Lecture Notes in Control and Information Sciences. Springer -Verlag, vol. 356, 2007, 166 p.

[Karatkevich, 2015] A.Karatkevich. Petri Nets in Design of Control Algorithms. In Design of Reconfigurable Logic Controllers. Springer -Verlag, Vol. 45 of the series Studies in Systems, Decision and Control, 2015, pp 1-14.

[Zakrevskij, 1989] A.D.Zakrevskij. To the theory of parallel algorithms for logical control. In Izvestiya AN SSSR. Tekhnicheskaya Kibernetika, 1989, No. 5, pp. 179–191 (in Russian).

[Zakrevskij, 1999] A.D.Zakrevskij. Parallel algorithms for logical control. Minsk: Institute of Engineering Cybernetics of NAS of Belarus, 1999. (in Russian).

[Hack, 1972] N.Hack. Anaysis of production schemata by Petri nets. Project Project MAC TR-94, Cambridge, 1972.

[Zakrevskij, 1984] A.D.Zakrevskij. Parallel automaton. In Doklady AN BSSR, 1984, vol. 28, No. 8, pp. 717 – 719 (in Russian).

[Bolton, 2015] W.Bolton. Programmable Logic Controllers, Elsevier, 2015, 6th Edition, 424 p.

[PLC, 1993] International Electrotechnical Commission. Programmable Controllers. Part 3. Programming Languages, IEC Publication, 1993, pp. 1131-1133.

[Allen-Bredly, 1976] Allen-Bredly Company. PLC Programming and operations manual, 1976, Bulletin 177.

[Cheremisinova, 2002] L.D.Cheremisinova. Implementation of parallel algorithms of logical control. Minsk: Institute of technical Cybernetics of NAS of Belarus, 2002, 246 p.

[Zakrevskij, 2000] A.Zakrevskij, B.Steinbach. Sequent automaton - a model for logical control. In Proceedings of the Intern. Workshop "Discrete optimization methods in scheduling and computer-aided design", Minsk: Republic of Belarus, Sept. 5–6, 2000.

[Cheremisinova, 2004] L.D.Cheremisinova. State Assignment of an Asynchronous Parallel Automaton. In Journal of Computer and Systems Sciences International, 2004, vol. 43, No. 5, p. 743–750.

[Chevalier, 1980] G.Chevalier. Le Grafcet: les automatismes par le diagramme fonctionnel et la technologie modulaire. Paris: Dunod, 1980.

[Cheremisinova, 1988] L.D.Cheremisinova, V.K.Vacilenock, E.V.Sheludko, L.V.Krasilnikova. The system of logical design of control devices on the base of programmable controllers. Minsk: Institute of Engineering Cybernetics of the of Belarus Academy of Sciences, 1988, 100 p. (in Russian).

[Cheremisinova, 1989] L.D.Cheremisinova. The methods of automaton realization of concurrent control algorithms expressed on PRALU language. Minsk: Institute of Engineering Cybernetics of the of Belarus Academy of Sciences, 1989, 58 p. (in Russian)

[Cheremisinova, 1988*] L.D.Cheremisinova. Minimization of finite-response sequential automata that implement parallel logical control algorithms. In Automatic Control and Computer Sciences, 1988, vol. 22, No. 4, pp. 69 – 74.

[Cheremisinova, 2004] L.D.Cheremisinova. Concurrency relation and automaton realization of logical control algorithms. In Proceedings of the Fifth Intern. Conf. on Computer-Aided Design of Discrete Devices, CAD DD'2004, v.1, Minsk: Rep. of Belarus, Nov.16–17, 2004, pp. 112 – 120.

Authors' Information

Liudmila Cheremisinova *– The United Institute of Informatics Problems of National Academy of Sciences of Belarus, principal recearcher, Surganov str., 6, Minsk, 220012, Belarus; e-mail: cld@newman.bas-net.by*

Major Fields of Scientific Research: *Discrete mathematics, Logic design automation*