Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Modeling of Programs and its Verification for Programmable Logic Controllers

Cleber A. Sarmento, José R. Silva, Paulo E. Miyagi, Diolino J. Santos Filho

University of São Paulo - Escola Politécnica, São Paulo, SP, Brazil
(Tel.: +55 11 3091 6025;
e-mails: cleber.sarmento@poli.usp.br; reinaldo, pemiyagi@usp.br; diolino.santos@poli.usp.br)

Abstract: Programmable logic controller is still the main device used for control of productive systems, which can be approached as discrete event dynamic systems. For programming these controllers, five languages were standardized by IEC 61131, and the LD (ladder diagram) language is distinguished among the others, i.e., it has been widely applied in productive systems, even with studies that confirm the restrictions and problems regarding the use of this language, such as the difficulties for errors identification in developed control programs. Therefore, this work presents a proposal for the modeling of extended finite state machines from control programs written in LD. These models are verified through a computational tool, aiming the identification of possible errors in the control program.

1. INTRODUCTION

Discrete event dynamic systems (DEDSs) are systems that evolve according to the occurrence of events in instants of time, which are usually non-deterministic. The variables that represent their states are modified discretely through the occurrence of events considered instantaneous. In this sense, productive systems (PSs) can be classified as DEDSs, and many manufacturing processes have used programmable logic controllers (PLCs) as the controlling devices, due to their reliability, hardiness and easy programming.

The IEC 61131 indicates the following languages for PLCs programming: IL (instruction list), ST (structured text), FBD (function block diagram), SFC (sequential function chart) and LD (ladder diagram). From all of the languages, the LD has still been widely used to develop control programs for PSs (Lucas; Tilbury, 2005).

According to (Uzam; Jones, 1998), when programmers develop a control program written in LD, they generally use heuristic programming methods, which are susceptible to errors. In general, these errors are hard to be identified because they are associated with the difficulty in analyzing the parallelisms, concurrences and sequences in control programs written in LD (Suesut et al., 2004). Then, several approaches have been conducted so as to eliminate these errors. Among these it is possible to find the ones that follow a procedure in which a controlling system model is initially created in a Petri net and then, the model is converted into a control program in LD (Lee et al., 2004). Another approach uses the inverse process, which is a more complex task than the aforementioned one, as can be seen in (Peng; Zhou, 2004).

There are approaches based on manual tests execution, where the control program under analysis is executed by a PLC and then results are analyzed according to the occurrence of events simulated by the programmer (Šusta, 2003). However, these tests may not present reliable results since they might not consider all the possible combinations of values of the input addresses of the control program.

So this work presents a proposal for modeling of extended finite state machines (EFSMs), which are derived from control programs written in LD, considering the PLC operational cycle. These EFSMs models can be verified using a computational tool, making the identification of errors feasible in the models, and consequently, in the control program.

This work is organized as follows: in the next section, definitions for control program elements and verification are presented. In section 3, some important concepts are revised. Section 4 describes the modeling procedure contribution for deriving EFSMs models from control programs written in LD. Final considerations are drawn in Section 5.

2. DEFINITIONS

The different types of data in a PLC can be represented, for example, by Boolean or integer variables, and these data can be associated with some control program elements whose proposed definition is the following:

Definition 1: control program elements refer to all the graphic elements of the LD language, which can be programmed neatly through the rungs (programming lines in LD).

These elements, according to the values of the variables associated with the input addresses of the PLC and its scan procedure, update the variables that are associated with the output addresses of the PLC. In this work, we consider the following five types of LD control program elements: contacts type NO (normally opened) and NC (normally closed), simple coils, set/reset coils, and timers type TON (timer on-

delay).

The following definition has its focus on the PSs controlled by PLCs and is mainly based on (IEC 61508-4, 1998; Zoubek, 2004; Wang; Tan, 2006).

Definition 2: verification is a process in which exhaustive tests are conducted on finite state models developed from existing control programs executed by a PLC aiming at identifying whether a specific state of the control program can be reached or not.

The specific state mentioned is formed by the combination of a set of variable addresses (and their respective values) that belongs to the control program in study. Such variables are related to inputs, outputs, flags, and timers. In case an unwanted state is reached through the verification of the model of the control program, it indicates the presence of error(s) in the model and, consequently, in the control program written in LD. When a sought state is not reached after the verification of the model it also indicates the presence of error(s).

An error in a control program written in LD can occur as a result of a mistake made by the programmer during the PLC addresses (variables) association with the control program elements, or even as a result of heuristic methods used by the programmer, which may result in the occurrence of problems related to the interlocking in the control program execution, and that may imply in an undesired dynamic behavior of the controlled PS. Another factor is related to the unexpected values associated with the PLC input addresses (input variables) which can be different from the expected ones in a certain moment due, for example, the occurrence of some fault in the detection and/or command devices connected with the PLC.

3. FUNDAMENTAL CONCEPTS

The verification of models generated from control programs written in LD is an issue studied by various researchers, for instance (Moon, 1994; Rossi; Schnoebelen, 2000; De Smet; Rossi, 2002; Zoubek et al., 2003), and this work benefits from the contributions of all these researchers. Before introducing the modeling procedure, a brief review of some fundamental concepts is presented.

3.1 PLC operational cycle

According to (De Smet; Rossi, 2002) the PLCs present an operational cycle as shown in Fig. 1.

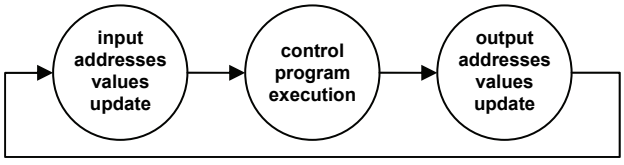

Fig. 1. Operational cycle of a PLC. The circles indicate the stages of the operational cycle and the arrows indicate the sequence of execution of these stages.

In Fig. 1 the scan time is the time measurement of the PLC operational cycle execution, which depends on the time value for execution of each one of the 3 stages of this operational cycle (Erickson, 1996).

As the amount of local input and output addresses of a PLC is not modified when it is running, the time value for execution of the stages "input addresses values update" and "output addresses values update" can be considered constant. In the stage "control program execution" the time value (that composes the scan time) results from the sum of the execution time of each control program element allocated neatly through the rungs of the control program (Chmiel et al., 2002). The control program generally includes conditional and/or unconditional jumps among rungs, and control program elements with a high execution time and whose execution is performed in non-deterministic time intervals. Then, the variation of the scan time value is basically dependent on the "control program execution" stage, and based on this consideration, the highest value associated with the scan time, which is the critical scan time, is the considered in the modeling procedure presented (see section 4.1 – step 7).

3.2 LD language

The programming language of PLC named LD was created to provide an easy assimilation language for professionals familiar with the electric circuits diagrams based on electromagnetic relays (Suesut et al., 2004). Rules for the construction and execution of a control program in LD language can be found in (Chmiel et al., 2002).

An important aspect of the LD language is the possibility for the rungs of the control program to be represented by Boolean expressions (Moon, 1994; Chmiel et al, 2002; Zoubek, 2004), and this aspect is fundamental to allow the development of the EFSMs models presented in this work.

3.3 Extended finite state machines (EFSMs)

The EFSMs can be considered an extension of the Mealy machines. In both machines (Mealy and EFSMs) restrictions and/or assignments are found in the transitions between their states. These restrictions and/or assignments that can be added to the transitions in an EFSM model are composed by variables in integer or Boolean format.

The mathematical definition of an EFSM adopted in this work is the one proposed by (Hong et al., 2002), which is valid for deterministic cases. Then, an EFSM is a mathematical model described by a tuple of the form (S, S_0, E, V, T), where:

- S is a finite set of states;
- $S_0 \subseteq S$ is the set of initial states;
- E is a finite set of events (reception and emission);
- V is a finite set of data variables partitioned into three disjoint subsets V_I, V_L, V_O, where V_I refers to the input variables, V_L refers to the local variables and V_O refers to the output variables;
- T is a finite set of transitions.

An example of an EFSM (simple coffee vending machine) presented by (Hong et al., 2002) is shown in Fig. 2.

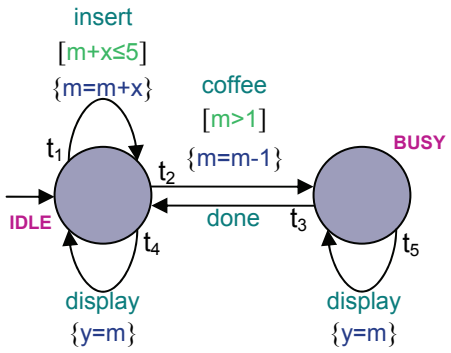

Fig. 2. An example of EFSM.

In Fig. 2 the following items are identified: states: IDLE and BUSY; initial state: IDLE; events: insert, coffee, done and display; transitions: t_1, t_2, t_3, t_4 and t_5; data variables of integer subrange [0..5]: x, m and y; restrictions to the transitions execution: [m+x≤5] (t_1) and [m>1] (t_2); value assignment to the variables with the execution of the transitions: {m=m+x} (t_1), {m=m–1} (t_2) and {y=m} (t_4 and t_5). The meaning of the data variables x, m and y is: x represents the credits inserted by the user in the vending machine; m represents the accumulated credits and; y shows the credits in the machine.

An EFSM presents a formalism that is associated with the concept of symbolic states (Hessel, 2007). This concept is important for the stipulation of temporal logic formulas, since these formulas refer to states and/or variables that belong to the model that will be verified. Therefore, a symbolic state is described by a tuple of the form (s, σ), where s represents the active state (in the process of simulation or verification) of the EFSM and σ represents the actual data value of all the variables of the EFSM.

3.4 Model verification

(Clarke et al., 2000) stated that the model checking is an automatic verification process of finite state concurrent systems, where specifications are stipulated using temporal logic formulas which are checked through an exhaustive scan process in the state space resulting from the models under analysis. In this work, the model verification can be approached as being a model checking, but the term verification will be used. The verification implies in identifying through the analysis of this state space whether a symbolic state (or symbolic states) fulfills (fulfill) the established temporal logic formulas or not.

In the work of (Hong et al., 2002) the use of temporal logic formulas that belong to the computation tree logic (CTL) is presented so as to verify models developed in EFSM. Here, we use a simplified version of the temporal logic CTL, which is compatible with the UPPAAL tool (Larsen et al., 1997), that is a tool for modeling and verification of timed automata extended with data variables, but that can be also applied to non-timed models, as seen, for example, in (Robinson-Mallett et al., 2005),. Among the formulas of this simplified version of the temporal logic CTL, our focus is on those that allow the verification of the reachability and safety properties.

The verification of the reachability property is determined by the formula $EF\ p$, where EF (*exist finally*) is the temporal operator that sets the verification tool to identify if there is at least one symbolic state (in the state space resulting from the verified model) that fulfills the propositional expression p. Such expression refers to states and/or values of data variables of the model that is being verified. The verification of the safety property is determined by the formula $AG\ p$, where AG (*all globally*) is the temporal operator that sets the verification tool to identify if all symbolic states (in the state space resulting from the verified model) always (or never: $AG\ \neg p$) fulfill the propositional expression p.

4. MODELING AND VERIFICATION

To describe the approach proposed we use symbols for addresses and control program elements found in the PLCs Allen-Bradley SLC 500 (Allen-Bradley, 2006). Symbols and control program elements found in other PLCs are non-restrictive for this work, being necessary that they are properly adequate to the EFSMs models proposed here.

Before presenting the modeling procedure, some examples of mapping a control program written in LD to EFSMs (using the UPPAAL tool) are illustrated. Fig. 3 presents an example of a rung from a control program written in LD and executed by a PLC Allen-Bradley SLC 500. In this example, when the two NO contacts in the mentioned rung have a true logic state, the simple coil will assume a true logic state.

Fig. 3. Example of a rung in LD.

In Fig. 3 the addresses associated with the two NO contacts are I:1.0/0 and I:1.0/1, and the address associated with the simple coil is O:2.0/0. A possible way to represent this rung as a Boolean expression can be: O:2.0/0 = I:1.0/0 & I:1.0/1. Such expression can be described as a value assignment expression that can be executed in a transition between EFSM states. In the format of the UPPAAL tool this expression can be written as: O2_0_0 = I1_0_0 && I1_0_1 (O2_0_0, I1_0_0 and I1_0_1 ∈ V_L), as could be seen in the transition between the states "execute_program" and "update_outputs" (Fig. 4).

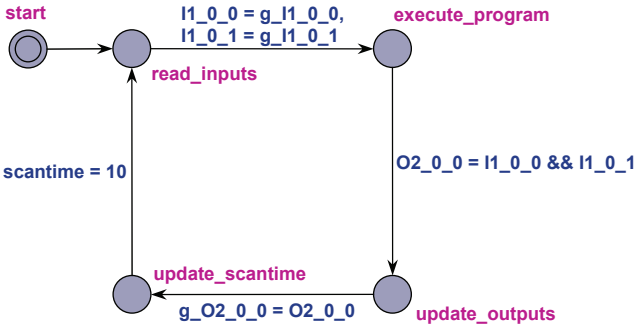

Fig. 4. Example of an EFSM main model.

The model in Fig. 4 is composed basically by 5 states: "start", "read_inputs", "execute_program", "update_outputs" and "update_scantime". In this EFSM is implicit the modeling of the three stages of the PLC operational cycle: between the states "read_inputs" and "execute_ program" the input addresses values are updated; between the states "execute_program" and "update_outputs" the execution of the mapped control program is done; finally, between the states "update_outputs" and "update_scantime" the values of the output addresses are updated. This example of EFSM presented in Fig. 4 is the so-called main model.

To perform the mapping of a control program written in LD the main model is not enough. Because of this fact other EFSMs that communicate with the main model are created. These EFSMs are called auxiliary models and they are used to model the behavior of the PLC input addresses and TON timer elements. The main model and the auxiliary models constitute a settling of EFSMs called global model.

The auxiliary models of the inputs (Zoubek, 2004) are necessary because the input addresses values can be different from the expected ones in a certain moment, as described in Section 2. Then, the auxiliary models of the inputs allow the verification tool to generate all the possible combinations of the input addresses values, so that the state space resulting from the global model verification can be constructed and analyzed. Two examples of auxiliary models of the inputs are shown in Fig. 5, which refer to the input addresses I:1.0/0 and I:1.0/1, that are modeled, respectively, by the variables g_I1_0_0 and g_I1_0_1 ($g\_I1\_0\_0, g\_I1\_0\_1 \in V_I$), which are also used in the main model presented in Fig. 4.

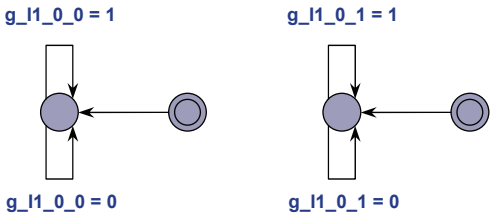

Fig. 5. Auxiliary models of the inputs (addresses I:1.0/0 and I:1.0/1).

Another auxiliary model proposed is the TON timer element. In Fig. 6, is shown an example of a rung where a TON timer element (address T4:0) is programmed, and whose preset value is 180s.

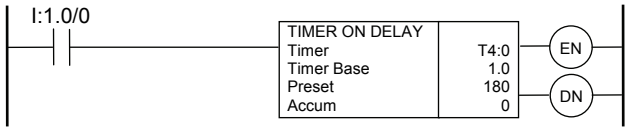

Fig. 6. Example of a rung that contains a TON timer element (address T4:0).

The proposed model in EFSM of the previous TON timer element (as shown in Fig. 6) is illustrated in Fig. 7.

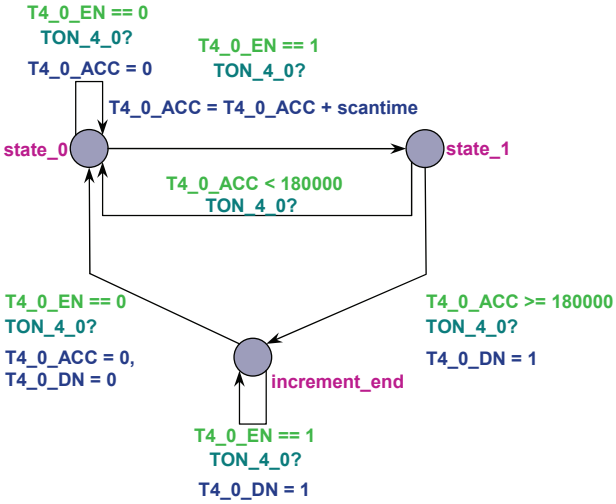

Fig. 7. Auxiliary model of the TON timer (address T4:0).

In the auxiliary model presented in Fig. 7 can be observed that all the transitions are conditioned to a fulfillment of expressions involving data variables and the reception of an event obtained through the synchronism channel TON_4_0? (reception). It can be also observed that the time value accumulated by the TON timer element T4:0, determined by the variable T4_0_ACC, is incremented by the variable scantime, whose value is updated in the main model through the execution of the transition between the states "update_scantime" and "read_inputs". This can be visualized in Fig. 8, which represents the main model of the rung showed in Fig. 6.

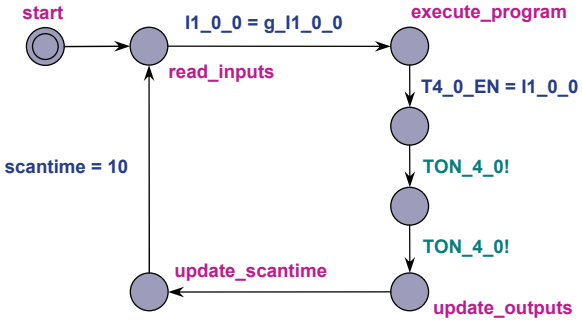

Fig. 8. Main model of the rung illustrated in Fig. 6.

In Fig. 8 is shown that in order to activate the auxiliary model of the TON timer element of the Fig. 7 it is necessary to set the variable T4_0_EN (T4_0_EN = 1) and execute two events of emission through two consecutive synchronism channels assigned by TON_4_0!.

4.1 Modeling procedure

The modeling procedure for deriving EFSMs from control program written in LD, which aims at identifying errors, can be synthesized through the following steps:

1. Initial parameterization of the main model;
2. Parameterization of the auxiliary models of the input addresses;
3. Parameterization of the auxiliary models of the TON timer elements;

4. Declaration of all the variables associated with the addresses of the control program;
5. Mapping of the rungs in the transition between the states "execute_program" and "update_outputs";
6. Insertion of two states between the states "execute_program" and "update_outputs" for each TON timer element existent in the control program. Each one of the two resultant transitions must be associated with the same channel of synchronism (emission) in order to establish the communication between the main model and the model of the TON timer element used;
7. Establishment of the critical scan time value.

4.2 Application example

Two examples of verifications are described to illustrate the application of the proposal. These verifications were both conducted in a global model, whose (original) main model is illustrated in Fig. 9. This main model is the result from the modeling procedure described earlier applied to a control program executed by a PLC Allen-Bradley SLC 500 that controls a gas burning equipment (Zoubek, 2004). The use of committed locations in the main model was necessary to avoid the out of memory error that was reported by the the UPPAAL tool, when such locations were not used.

In the first verification, the following temporal logic formula is used: *A[]* not (Main.read_input and Main.B3_0_0==1 and Main.B3_0_8==1). It is related to the verification of a safety property of the global model, and the verification tool checks for all symbolic states of the state space resulting from the global model under analysis if does not exist an occurrence of a symbolic state where the propositional expression Main.read_inputs and Main.B3_0_0==1 and Main.B3_0_8==1 holds (*A[]* has the same meaning given by temporal operator *AG*, and "Main" refers to the name given to the main model (Fig. 9) whose state and variables of interest are "read_inputs" (state), B3_0_0 and B3_0_8 (variables)).

In the current case, this verification is not satisfied, meaning that it is possible to obtain a symbolic state where Main.read_inputs and Main.B3_0_0==1 and Main.B3_0_8==1 hold, what is not intended, indicating the presence of error(s) in the modeled control program. Analyzing the history path of symbolic states generated by the UPPAAL tool, this path showed how the undesired symbolic state is reached. Then, association errors between addresses and control program elements are identified in the original control program. After correcting these errors in the main model of the control program, a new verification is performed, and the result shows that the propositional expression Main.read_inputs and Main.B3_0_0==1 and Main.B3_0_8==1 does not hold, confirming that the model is correct, that is, corresponding to its specification.

The second verification performed uses the following temporal logic formula: *E*<> (Main.read_inputs and Main.B3_0_0==1 and Main.B9_0_12==0). It is related to the verification of a reachability property, where *E*<> has the same meaning given by temporal operator *EF*. This second verification is also not satisfied and it indicates that the propositional expression stipulated does not hold. The values for the addresses B3:0/1==1 and B9:0/12==0 are not expected to hold simultaneously, according to the specification of the control program operation, and the achieved result indicates that the control program does not contain any error that results in B3:0/1==1 and B9:0/12==0 (simultaneously).

The corrections performed in the main model (Fig. 9) after the first verification were also applied to the original control program executed by a PLC Allen-Bradley SLC 500, and the tests confirmed the effectiveness of the proposal presented.

5. FINAL CONSIDERATIONS

This work introduces a modeling procedure for deriving EFSMs from control programs written in LD. These EFSMs models can be verified using a computational tool, making the identification of errors feasible in the models, and consequently, in the control program. Tests have been conducted with other existing and running control programs being executed by PLCs Allen-Bradley SLC 500, before and after the correction of errors (identified in the global model of these control programs), and the results confirmed the proposed procedure effectiveness.

In Zoubek (2004), the UPPAAL tool presented problems to finish the verification of the timed automata models generated from the control program of the gas burning equipment. Then, it was necessary to eliminate some rungs of the related control program to create smaller timed automata models in order to prevent the state explosion problem. In our proposal this elimination of rungs was not necessary.

There are some characteristics that were not considered in the proposed approach and that can be found in existing PLCs, for instance, any interruption during the execution of a control program so as to execute another procedure (this characteristic is being considered for the improvement of this work). Another improvement that is being considered is to adopt the value of the time execution of each rung of the modeled control program, in such a way that this value would be, after each rung execution, added to the value of the scantime variable, not being necessary to use critical values of scan time in the main models anymore.

This work was partially supported by Brazilian governmental agencies that support research: CNPq, CAPES and FAPESP.

REFERENCES

Allen-Bradley (2006). *SLC 500 Instruction Set – Reference Manual*.

Chmiel, M.; Hrynkiewicz, E.; Muszyński, M. (2002). The way of ladder diagram analysis for small compact programmable controller. *6th KORUS Russian-Korean Intern. Symp. on Sci. and Tech*., pp. 169-173.

Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; Veith, H. (2000). Progress on the state explosion problem in model

checking. *Lecture Notes in Computer Science*, Vol. 2000, Springer-Verlag, Berlin.

De Smet, O.; Rossi, O. (2002). Verification of a controller for a flexible manufacturing line written in ladder diagram via model checking. *ACC2002 American Control Conference*, pp. 51-56.

Erickson, K.T. (1996). Programmable logic controllers – the workhorse of factory automation keeps things on track. *IEEE Potentials (Publications)*. 15(1): 14-17.

Hessel, A. (2007). *Model-based test case generation for real-time systems*. PhD Dissertation, Department of Computer Systems, Uppsala University, Sweden.

Hong, H.S.; Lee, I.; Sokolsky, O.; Ural, H. (2002). A temporal logic based theory of test coverage and generation. *8th TACAS Intern. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science*, Vol. 2280, pp. 327-341.

IEC 61508-4. (1998). Functional safety of electrical/ electronic/programmable electronic safety-related systems – Part 4: Definitions and abbreviations.

Larsen, K.G.; Pettersson, P.; Yi, W. (1997). Uppaal in a Nutshell. *Intern. J. of Software Tools for Technology Transfer*. 1(1-2): 134-152.

Lee, G.B.; Zandong, H.; Lee, J.S. (2004). Automatic generation of ladder diagram with control Petri net. *J. of Intelligent Manufacturing*, 15: 245-252.

Lucas, M.R.; Tilbury, D.M. (2005) Methods of measuring the size and complexity of PLC programs in different logic control design methodologies. *Intern. J. of Advanced Manufacturing Technology*. 26(5-6): 436-447.

Moon, I. (1994). Modeling programmable logic controllers for logic verification. *IEEE Control Systems Magazine*, 14(2): 53-59.

Peng, S.S.; Zhou, M.C. (2004). Ladder diagram and Petri-net-based discrete-event control design methods. *IEEE Trans. on Systems, Man and Cybernetics - Part C: Applications and Reviews*, 34(4): 523-531.

Robinson-Mallett, C.; Liggesmeyer, P.; Mücke, T.; Goltz, U. (2005). Generating optimal distinguishing sequences with a model checker. *ACM SIGSOFT Software Engineering Notes. Session: Advances in Model-Based Testing*, 30(4).

Rossi, O.; Schnoebelen, P. (2000). Formal modeling of timed function blocks for the automatic verification of ladder diagram programs. *4th ADPM Intern. Conf. Automation of Mixed Processes: Hybrid Dynamic Systems*, Dortmund.

Suesut, T.; Inban, P.; Nilas, P.; Rerngreun, P.; Gulphanich, S. (2004). Interpretation Petri net model to IEC 1131-3: LD for programmable logic controller. *IEEE Conf. on Robotics, Automation and Mechatronics*. Singapore, Vol. 2, pp. 1107-1111.

Šusta, R. (2003). *Verification of PLC programs*. Doctoral Dissertation, Department of Cybernetics of the Faculty of Electrotechical Engineering. Prague, Czech Republic.

Uzam, M.; Jones, A.H. (1998). Discrete event control system design using automation Petri nets and their ladder diagram implementation. *Intern. J. of Advanced Manufacturing Systems*. 14(10): 716-728.

Wang, L., Tan, K.C. (2006). *Modern Industrial Automation Software Design – Principles and Real-World Examples*. John Wiley & Sons Inc., Hoboken, New Jersey.

Zoubek, B.; Roussel, J-M., Kwiatkowska, M. (2003). Towards automatic verification of ladder logic programs. *IMACS-IEEE CESA'03 Computational Engineering in Systems Applications*, paper S2-I-04-0169, 6 p.

Zoubek, B. (2004). *Automatic verification of temporal and timed properties of control programs*. PhD Thesis, School of Computer Science, University of Birmingham, UK.

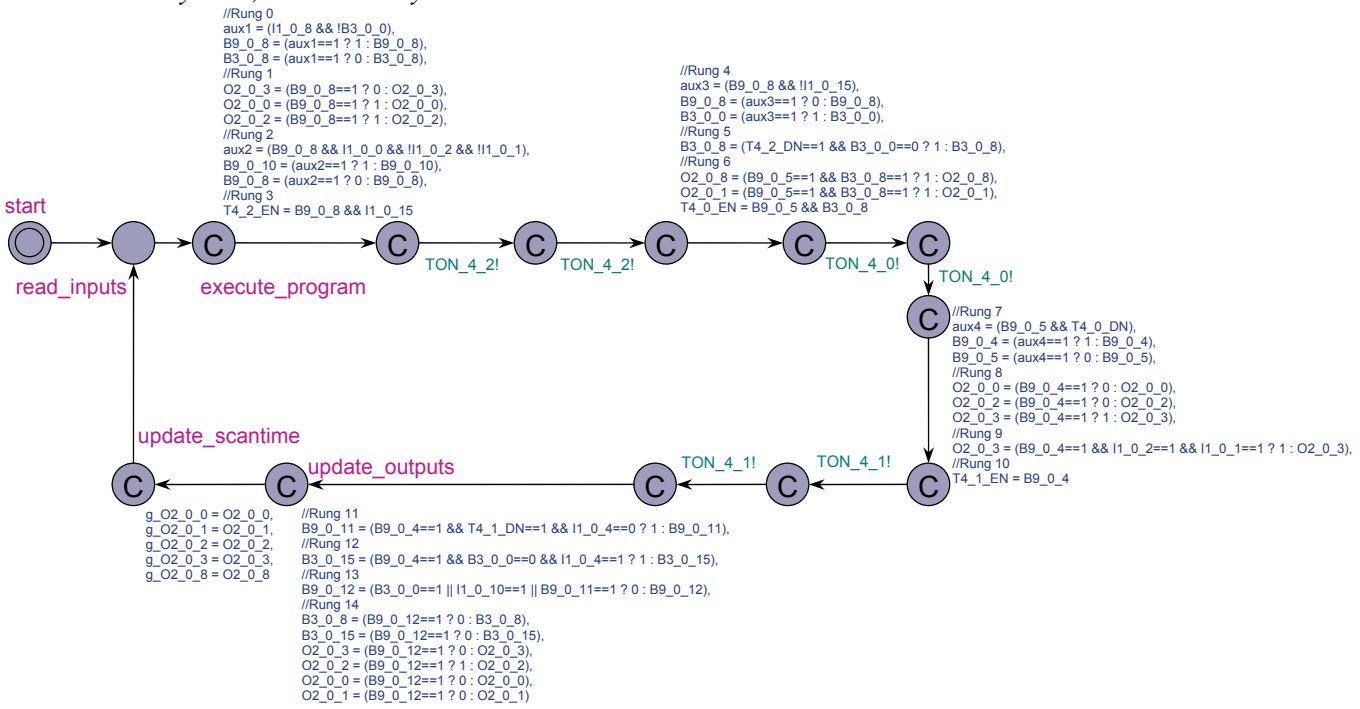

Fig. 9. Main model of the gas burning equipment.