J Inno Sci Eng,2023, 7(1):13-28
https://doi.org/10.38088/jise.1120186

Research Article

A New Method for Verification and Evaluation of PLC Software

Muhammed A. Öz[1*], **Özgür T. Kaymakci** [2]

[1] *Yıldız Technical University, Department of Control and Automation Engineering, İstanbul, Turkey*
[2] *Canakkale Onsekiz Mart University, Department of Electrics and Electronics Engineering, İstanbul, Turkey*

Cite this paper as: Muhammed A. Oz and Ozgur T. K. (2023). A New method for verification and evaluation of PLC software. 7(1): 13-28.

*Corresponding author: Ozgur T. Kaykamci

E-mail: okaymakci@comu.edu.tr

Received Date:24/05/2022
Accepted Date:09/09/2022

Abstract

Varying market demands and changes in production standards require production systems to be effortlessly modifiable and quickly operational. On the other hand, designing, developing, and testing the control system of a new production system prove costly and time-consuming. Therefore, most engineers write code intuitively and apply basic and insufficient tests. Moreover, most of the code developed for industrial control systems is still written manually using the ladder programming language. At the same time, almost all code development platforms support users with only manual test interfaces. This causes the testing process to be very long and laborious. In addition, not all possible input and output combinations of the code can be tested most of the time. This is a serious handicap, especially for safety-related systems. This study aims to develop a reusable and quickly implementable method that will accurately translate RTC program and the behavior of RTC in a modular Petri net model. Through this translated model, the system and safety requirements written in the Computation Tree Logic can be verified. An advantage of this method is that it does not require a plant model which makes it reusable for new plants and provides a quick verification method for code written intuitively. A case study is given to demonstrate the correctness of our method.

Keywords: Industrial automation, Programmable logic controller (PLC), Code verification, Safety.

1. Introduction

Automation systems are an indispensable part of the industry because of their plentiful benefits. Even though these systems save labor, energy, and materials improve quality and accuracy, they have a big disadvantage related to their high initial costs. It takes a fairly long amount of time and resources to design and develop a new automation system. Designing an automation system can be divided into two main tasks. The first task is to choose and assemble the right components for the system. This task is relatively simple when compared with the second task which is to develop a control algorithm and realize it on a real-time controller.

Most developers in the industry follow the V-model when a real-time control system is to be developed [1]. The model includes the steps to take in the process of development, which are requirements, analysis, implementation, testing, verification, and validation. To apply this V-model, the system must be modeled as a discrete event system, after the requirements of the system are collected. A control strategy must be chosen and its result must be converted into real time controller code. Ljungkrantz et al. proposed a method to develop specifications for safety components in PLC programs. [2]. Viera et al. used automata theory to model flexible manufacturing systems consisting of several subsystems and presented a method that allows designers to systematically convert supervisory control theory results into a programmable logic controller code [3]. Hu et al. modeled and analyzed automatic manufacturing systems with synchronous operations using petri nets [4]. Different modeling needs result in a variety of modeling techniques, for scholastic systems a modified petri net called colored stochastic petri nets [5], and for time-critical systems called timed arc Petri nets [6] can be used.

Röshe et al. and Ovatman et al. presented review papers on model based testing approaches [7, 8]. Methods on verification of RTC programs can be classified into two general categories. One of the categories would include the methods where the model to be verified is a combination of the plant model and the model of the RTC program. Bauer et al. presented a method to convert timed sequential functional charts to discrete event timed automata. They also analyzed the converted model UPPAAL [9]. Mertke and Frey worked on Signal Interpreted Petri Nets and presented a new graphical design approach. They implemented the results on a benchmark problem. [10]. Alenlejung et al. introduced the discrete event modeling language Sensor Graphs, which is intended for modeling physical systems from the perspective of a PLC programmer and for usage within formal verification, process observation, and fault detection [11]. Nellen et al. presented two CEGAR-based methodologies for the reachability analysis of SFC-controlled chemical plants [12].

A different approach of including system model is through using simulations. Carlsson et al. used OPC interface in order to connect PLC and a simulation tool. They defined four major problems related to OPC and introduced two possible solutions [13]. Rankin and Jiang developed a platform that provides a flexible simulated testing environment which enables synchronized coupling between the real and simulated world [14]. Park et. al. introduced a visual verification platform based on discrete event systems specifications approach. Here the models can be in a hierarchical, modular manner [15]. Patil et al. presented how integrated circuit (IC) verification method can be used effectively for assuring functional correctness and response time analysis of PLC program [16]. Koo et al. presented a framework of virtual plant models for the verification of PLC logic through modeling and simulation [17]. Although including system model in the verification process increases correctness, it also increases costs and start up time. Besides, this method requires a specialist to model the system and methods are not reusable.

Another way to verify a RTC program is without using a plant model. Wang et al. has proposed a systemic method for the construction of verification model. PLC system architecture and PLC features has been modeled as components and connectors [18]. Zhang et. al. has proposed a method that generates timed event sequences and implemented this strategy into a program named VETPLC [19]. Adiego et al. propose a general methodology to perform automated model checking of complex properties expressed in temporal logics on PLC programs and is based on an intermediate model [20]. Xiao et al. introduces the compositional verification framework for PLC programs [21]. Ulewicz et al. proposed a novel method for regression verification of PLC code, which allows one to prove that two variants of a plant's software behave identically in specified situations, despite being implemented differently [22]. He et al. proposed a model-based verification of PLC programs using Simulink design [23]. Adiego et al. used PLCverif that was produced by CERN. Here they tried to reveal the bugs in the PLC program that was generated from a functional safety perspective [24]. While the methods above eliminate the need for a system model, they lack the critical components of verification such as readability, modularity and reusability. In addition, the use of intermediate models are also problematic as they require translating the system a few times and decrease the accuracy of the model.

Varying market demands and changes in the production standards require production systems to be effortlessly modifiable and quickly operational. While physical parts of the production system can be built efficiently, designing, developing and testing the control system prove costly and time consuming. Therefore, most engineers write a code out of intuition and apply basic and insufficient tests. The aim of this study is to develop a reusable and quickly implementable method to verify the safety and performance requirements of the respective system. This method will improve modification and development times, financial costs and safety.

Main contribution of this study is to provide a method which accurately translates RTC program and the behavior of RTC in a modular Petri net model. Through this translated model, the system and safety requirements can be verified. An advantage of this method is that it does not require a plant model which makes it reusable for new plants as long as requirements are updated. Petri net models are verified and viewed through TAPAAL, which is a tool for editing, simulating, analyzing and verifying TAPN. The verification process is done on the basis of certain safety requirements, which are written in Computation Tree Logic (CTL), which models time as a tree-like structure, formulation [25], [26]. A general schematic of the proposed method is given in Figure 1.

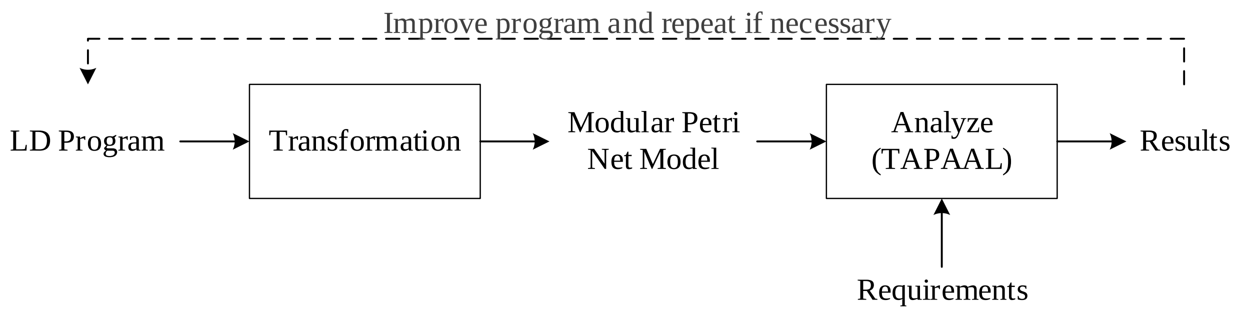


Figure 1. General schematic of the proposed method.

2. Modeling Real Time Controllers

Programmable logic controllers (PLCs) are among the most used RTCs in the industry. PLCs can be defined as cyclic data processors meaning they repeat a predetermined control algorithm indefinitely. The algorithm that is executed cyclically is called the program. To unify the syntax and semantics of programming languages for PLCs, the International Electrotechnical Committee (IEC) published IEC 61131-3. One of the two graphical languages recommended by this standard, the ladder diagram (LD) will be covered in this paper.

2.1. LD program

LD is a programming language that is represented by a graphical diagram based on the circuit diagrams of relay logic hardware. It consists of a main program body and subprograms which perform a specific task and can be called from the main program body. Consider an LD program P having m rungs and n subprograms $P_1, \dots, P_n$ each having m_i, $i = 1, \dots, n$ rungs.

$$P = \{(j, diagram_j) | j = 1, \dots, m\} \cup \bigcup_{i \le n} P_i$$

Program P can be defined as given in equation where

$$P_i = \{(j_i, diagram_{j_i}) | j_i = 1, \dots, m_i\}, for\ all\ i = 1, \dots, n$$

Where $diagram_j(diagram_{j_i})$ designates the diagram at rung $j(j_i)$ of $P(P_i)$.

2.2. Model of an LD Program

To describe the potential behavior of the program a Petri Net can be used. Let $PN = (\mathrm{P}, \mathrm{T}, \mathrm{F}, \mathrm{W}, M_0)$ be a 5-tuple where, P is a finite set of places, T is a finite set of transitions, the places P and transitions T are disjoint $(P \cap T = \emptyset)$, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, $W : F \to (\mathbb{N} \setminus \{0\})$ is the arc weight mapping, and $M0 : P \to \mathbb{N}$ is the initial marking representing the initial distribution of tokens. PN model of an LD program can be constructed as follows:
Set of places P. $P = P_p + P_b + P_c$

P_p is a set of places expressing the variables of the program defined by the programmer. Although LD supports many data types in this paper only Booleans are discussed as they are most commonly used in the industry. Assuming the main program P_r contains n_{pr} variables and each subprogram P_{ri} contains n_{pri} variables, then $P_p = P_{pr} x P_{pr_2} x \dots x P_{pr_n}$ where $P_{pr} = Pr_{var_1} x Pr_{var_2} x \dots x Pr_{var_{n_{pr}}}$ and $Pr_{var_j} = \{\text{true}, \text{false}\}$ for $j = 1, \dots, n_{pr}$.

P_b is a set of states expressing the behavior of the plc. In this approach, variables that are not defined in the program by the user but are still needed to express some functions such as rising edge trigger. This function requires the previous value of the variable. Thus, a place must be created to store value of the variable. Assuming the main program P_b contains n_b functions as describes above and each subprogram P_{b_i} contains n_{b_i} functions, then $P_b = P_b x P_{b_1} x \dots x P_{b_{n_i}}$ where $P_b = Pb_{var_1} x Pb_{var_2} x \dots x Pb_{var_j}$ and P_b defines the variables needed for expressing all functions in the appropriate subprogram and Pb_{var_j} defines the variables needed for the respective function in that subprogram.

P_c is a set of places defining program counters. The program counter of each of the program modules together to form the set SPC. Thus, $SPC = \{1, \dots, maxpc\} \times \{1, \dots, maxpc1\} \times \dots \times \{1, \dots, maxpcn\}$. Adding the $P_{c_{Reading}}$ a place to define the reading cycle of PLC and $P_{c_{Writing}}$ a place to define the writing cycle of PLC to this set, a PLC behavior can be accurately modeled.

Flow relation, $F \subseteq (P \times T) \cup (T \times P)$ describes how the tokens of the places change after the firing of each transition. The condition described by the LD diagram is modeled using the firing rule of the transitions and the actual firing of the transition models the action taken as a result of the respective conditional statement. Since this study only deals with logical values of the software, places in the PN model can only have two values, as a result the upper limit of tokens for our Petri net places is one. Here zero tokens express the value false whereas one token expresses the value true. Furthermore, the weights of the arcs must always be set to one considering that places can only contain one token. The Petri net will be self-looping because the input places lose their tokens when transition fires and their tokens must be returned. A Petri net under these terms is called an ordinary, finite-capacity net with strict firing rule.

This study proposes modeling the RTC and its ladder program in three steps. First, rungs of the LD program are modeled one by one as independent model components. Next, a behavioral model of the RTC is generated considering the requirements of the LD program. Finally, model components of LD program and the behavioral model are assembled.

2.3. Model of an LD Program

The logic in a ladder diagram typically flows from left to right. The diagram which resembles a ladder can be divided into sections called rungs. RTC executes these rungs one by one from top to bottom. Each rung typically consists of a combination of input instructions and these instructions lead to a single output instruction. However, rungs may also contain function block instructions. A rung can be tough as a condition and an action taken depending on the condition. In this example a single rung of a LD program is given in figure A and its corresponding PN model is given Figure 1. The condition on this rung is that P_0 must be true and P_1 must be false. The action taken depending on this condition is setting the output variable P_2 to true. In the case of P_2 being true, meaning it already has a token, after the firing of transitions its token will increase to more than one and this is not acceptable. Therefore, two transitions are needed to take the respective action when the condition is satisfied. Transition T_3 is fired when the previous value of P_2 is false and transition T_2 is fired when previous value of P_2 is true. Notice that in order to complete the execution in the PN model the token of $rung_n$ must be transferred to the place $rung_{n+1}$. Consequently, transitions T_0 and T_1 are used to transfer the token from $rung_n$ to $rung_{n+1}$ when the condition is not satisfied. The components with the dotted line are not necessary to model the code but to model the behavior of the RTC. The places $rung_n$ and $rung_{n+1}$ are members of the program counter set. The place $rung_n$ is used to simulate the RTCs behavior of running the code line by line with order. After $rung_n$ loses its token $rung_{n+1}$ recieves it and executes the PN model of the next rung.

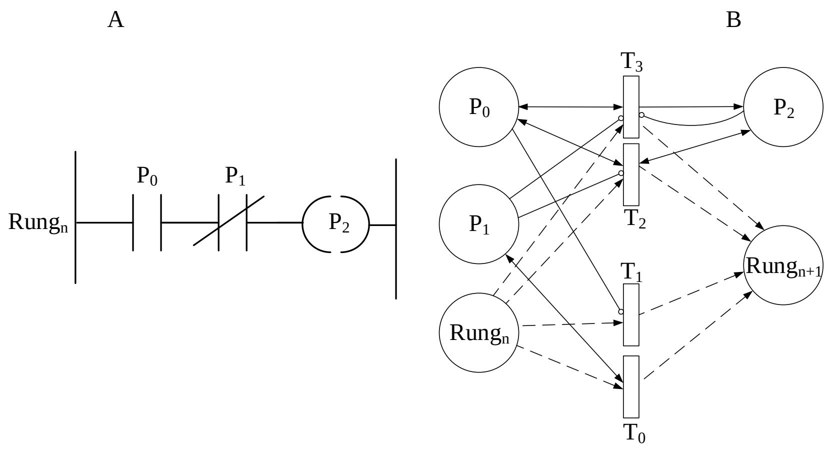


Figure 2. An example of a RTC instruction and its Petri net model correspondence.

An example of a RTC instruction and its corresponding Petri net model is given in Figure 2-A and Figure 2-B. The instruction calculates the conjunction of variable P0 and the complement of variable P1, and writes the result on the output variable P2. In the Petri net correspondence, in addition to the instruction rung indicators are used to run Petri net model components in order just as a RTC. Furthermore, the component must complete the instruction operation in one transition. At the end of the operation next rung indicator must get a token and the input variables must keep their initial values. Petri net correspondences of commonly used instructions are given in the Appendix.

2.4. Model of PLC Scan Cycle

When the scan cycle starts, PLC checks each input card to determine its logical state and saves this information in a data table to be used in the next cycle step. This speeds up the process while avoiding cases where an input changes from the start to the end of the program. The PLC executes programs one instruction at a time using only the memory copy of the inputs. When all instructions are completed, the outputs are updated using the temporary values in memory. The PLC updates the status of the outputs based on which inputs were on during the first step and the results of executing a program during the second step. The PLC now restarts the process by scanning inputs.

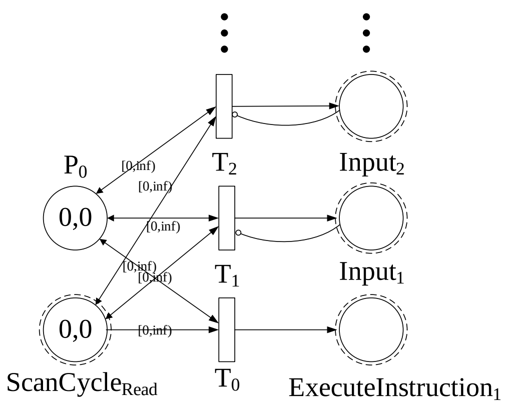


Figure 3. Petri net model of RTC's reading cycle.

Proposed method simulates the same behavior. Each scan cycle stage is indicated by a place and by connecting these places to respective transitions, PLC scan cycle stages and one by one execution of the program instructions can be achieved in the model. Similar to the PLC, the model first determines the logical states of the input places. Software that is being tested should function safely under all conditions. Therefore, the model should allow all input combinations even when they are not possible in the physical system. This is achieved using the PN model component given in Figure 3. This component takes advantage of the firing rules of PNs by enabling many transitions until T_0 transition is fired and all input combinations are possible. The place $ScanCycle_{Read}$ indicates the scan cycle step: read input values. When $ScanCycle_{Read}$ loses its token, the input combination is decided and will stay same until the next cycle. After this cycle stage $ExecuteInstruction_1$, which indicates that first instruction should be executed, receives a token and instructions are executed in order. Each scan cycle step and each instruction is a component of the Petri network and interactions are provided through shared places, and these shared places are shown with two circles where the outer one is dotted. This improves readability, reduces design complexity of the network and pave the way to automatic modeling.

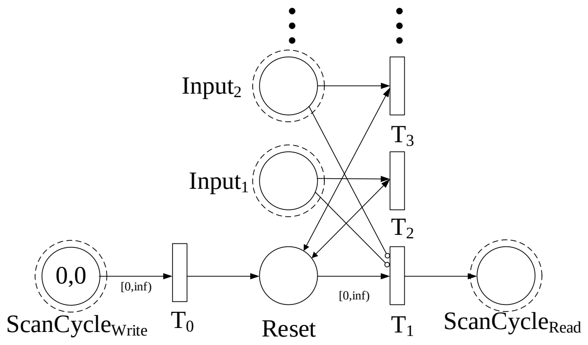


Figure 4. Petri net model of RTC's output cycle.

Last scan cycle is the output cycle and its representation is given in Figure 4. $ScanCycle_{Write}$ is the indicator of this cycle. This model component has two main tasks; the first task is to store the previous input logic states especially for PLC instruction such as rising and falling edge trigger, and the second task is to clear all inputs before going into the reading cycle step using sink transitions, transitions without any output place. This component is also important because it represents when outputs are updated therefore must be kept in mind in the verification process. The values of outputs should only be checked in the output cycle. Model representations of all scan cycle steps are shown in the Figure 5.

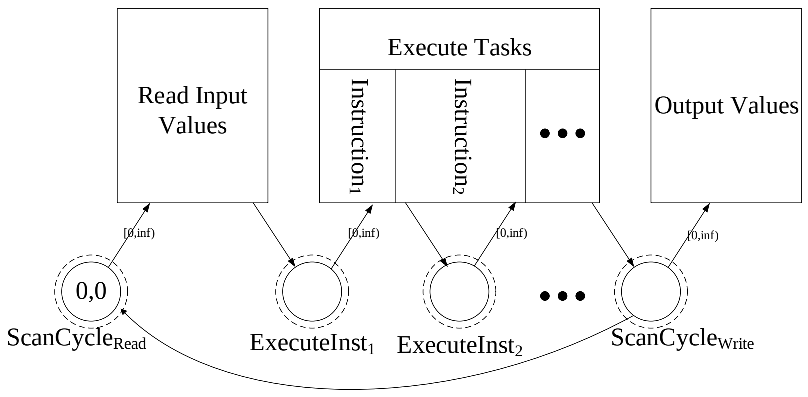


Figure 5. Model representations of all scan cycle steps.

3. Case Study

A program that controls two robot arms which are a sub-system of a production line is used for the case study. A representation of the system is given in Figure 6. There are two conveyors that carry materials inside the subsystem and only one conveyor moves materials out of the system with the guidance of presence sensors. These conveyors are controlled by a different higher level program of the production system. The control program is written for two robot arms which pick up materials when ready from the assigned conveyor and places them on conveyor 3.

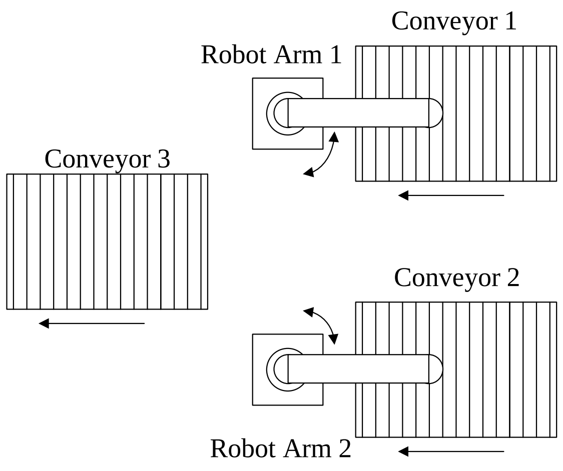
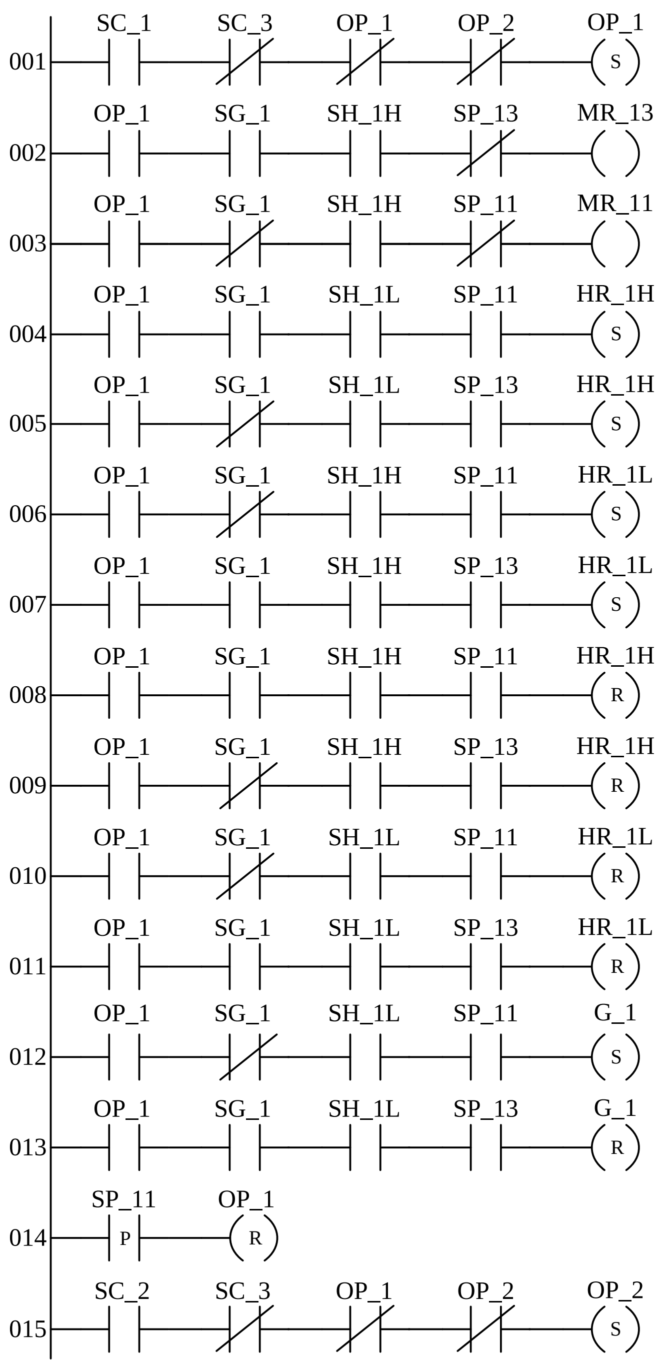


Figure 6. Schematics of the production system.

While the system is idle, which means it is waiting for a detection of a material on either conveyor, the robot arms are in a position above their assigned conveyors with the robot’s gripper being at a certain height. When a material is detected on either conveyor, the assigned robot arm lowers itself in order to pick the material with its gripper then goes back to the original height and starts moving in circular motion towards conveyor 3. Once the robot arm lowers its gripper again until it can place the material safely on conveyor 3 and moves back to its original height, it moves towards its assigned conveyor. The process of moving a material is finished once it is in a position above its assigned conveyor.


001 SC_1 SC_3 OP_1 OP_2 OP_1 S
002 OP_1 SG_1 SH_1H SP_13 MR_13
003 OP_1 SG_1 SH_1H SP_11 MR_11
004 OP_1 SG_1 SH_1L SP_11 HR_1H S
005 OP_1 SG_1 SH_1L SP_13 HR_1H S
006 OP_1 SG_1 SH_1H SP_11 HR_1L S
007 OP_1 SG_1 SH_1H SP_13 HR_1L S
008 OP_1 SG_1 SH_1H SP_11 HR_1H R
009 OP_1 SG_1 SH_1H SP_13 HR_1H R
010 OP_1 SG_1 SH_1L SP_11 HR_1L R
011 OP_1 SG_1 SH_1L SP_13 HR_1L R
012 OP_1 SG_1 SH_1L SP_11 G_1 S
013 OP_1 SG_1 SH_1L SP_13 G_1 R
014 SP_11 P OP_1 R
015 SC_2 SC_3 OP_1 OP_2 OP_2 S

If a material is detected on conveyor 1 and no material is detected on conveyor 3 and no other process is working then start the process of transferring material from conveyor 1 to 3 (Process 1).

If Process 1 is active and arm it at moving height then if an object is in the gripper and robot is not at conveyor 3 then move to conveyor 3. If there is no object in gripper and robot is not at conveyor 1 then move to conveyor 1.

If Process 1 is active and arm is at gripping height then start lifting the arm to moving height if an object is in the gripper and robot is at conveyor 1 or if there is no object in gripper and robot is at conveyor 3.

If Process 1 is active and arm is at moving height then start lowering the arm to gripping height if an object is in the gripper and robot is at conveyor 3 or if there is no object in gripper and robot is at conveyor 1.

If Process 1 is active and arm is at moving height then stop lifting the arm to gripping height if an object is in the gripper and robot is at conveyor 1 or if there is no object in gripper and robot is at conveyor 3.

If Process 1 is active and arm is at gripping height then stop lowering the arm to gripping height if an object is in the gripper and robot is at conveyor 3 or if there is no object in gripper and robot is at conveyor 1.

If Process 1 is active and arm is at gripping height then grab the object if an object is not on the gripper and robot is at conveyor 1

If Process 1 is active and arm is at gripping height then release the object if an object is on the gripper and robot is at conveyor 3.

When the arm gets back to conveyor 1 stop process 1.

If there a material is detected on conveyor 2 and no material is detected on conveyor 3 and no other process is working then Start the process of transferring material from conveyor 2 to 3 (Process 2).

Figure 7. Program for the production system and comments.

PLC code segment that is given in Figure 7 is an intuitively written program for the production line mentioned above. This code segment except the last rung runs one of the robot arms. Rest of the program is just a copy of this segment where the names of the inputs and outputs are changed accordingly to work with the other robot arm hence, space is preserved, and readability is improved for the rest of the program which is not included. The process does not allow two robots operate at the same time. Thus, just adding the last rung is enough to verify that under no circumstance robots work simultaneously. All in all, the whole program can be verified by just verifying the program segment since the robots operate the same. This program was tested on a simulative environment and no mistake or unwanted scenarios were detected. Table 1 presents the lists of the inputs and the outputs used in the PLC code that will be verified using the proposed method in this paper.

Table 1. Verification result before and after correction.

Sensor Inputs	
SC_x	Material presence detection on conveyor x *(x={1,2,3})*
SP_x1	Robot arm *x* is in position to pick materials from conveyor *x (x={1,2})*
SP_x3	Robot arm *x* is in position to place materials on conveyor 3 *(x={1,2})*
SH_xL	Robot arm *x* is at the appropriate height to pick and place materials *(x={1,2})*
SH_xH	Robot arm *x* is at the appropriate height to move materials *(x={1,2})*
SG_x	Robot arm *x* is holding a material *(x={1,2})*
Control Outputs	
MR_x1	Move the robot arm x towards the assigned conveyor *x (x={1,2})*
MR_x3	Move the robot arm x towards conveyor *3 (x={1,2})*
G_x	Controls the pick and place process of robot arm *x (x={1,2})*
MH_xL	Decreases the height of the robot x gripper *(x={1,2})*
MH_xH	Decreases the height of the robot x gripper *(x={1,2})*
Intermediate Variables	
OP_x	Starts the process of transferring material from conveyor *x* to conveyor 3 *(x={1,2})*

3.1. Transformation

Using the proposed approach, this program is transformed into a modular TAPN model so that each rung of the code is a counterpart to a component of the model. This property simplifies modeling and increases the readability of the model which is useful when searching for errors in the code. TAPAAL, a software tool for modeling and verifying TAPN models, is used in this case study.

All transformation steps are standardized and ordered, which make automating the transformation process possible. First, reading and writing cycles of the RTC are modeled using the variables of the program. Next, rungs of the program are transformed beginning with the places associated with the controller's operation cycle, such as program counter, being implemented into the model component. Variables used in the rung are added as places into the model and with the help of a transition the update of the output is simulated. Even though a transition is enough for this process other combinations are implemented to ensure that the token is transferred between program counter places. Only a few components of the resulting model is given here to preserve space. Transformed TAPN model component of rung 6 is given in Figure 8 part A and transformed TAPN model component of rung 14 combined is given in Figure 8 part B.

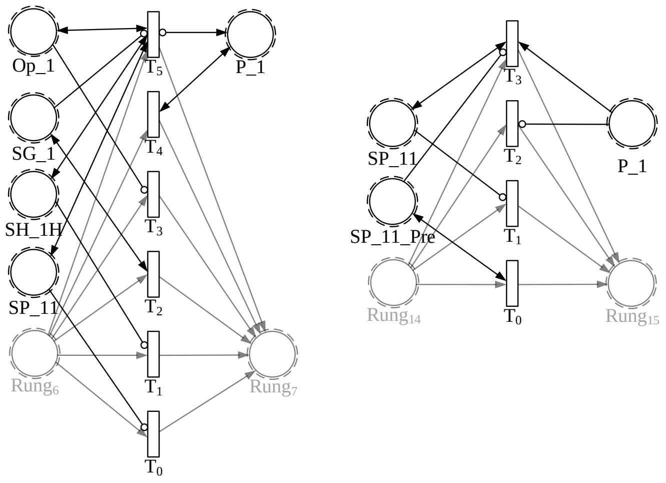


Figure 8. The TAPN model of some rungs.

In the component TAPN model of rung 6, transition named T_5 fires when the right combination of input values that set P_1 variable are present. T_4 transition is there to keep P_1 place from receiving more than one token, which is an unwanted situation. Any other combination of input values will fire one of the other transitions to transfer the token to the next program rung. TAPN Model component of the Rung 14 is very similar.

3.2. Verification and Correction

To ensure the reliable operation of this system, some safety features must be maintained. These safety requirements are added to the model with the help of Tapaal editor using Computer Tree Logic (CTL) formulation. CTL logic is a branching-time logic. CTL logic formulas are evaluated over all possible paths of a Kripke structure. To verify the formulized requirements Tapaal translates the TAPN model into Network of Timed Automata (NTA) and use Uppaal verification on the produced NTA. All queries are checked via Tapaal Discrete Verification method based on the Breadth First search order in state space. As the coverability tree is too large, it is not given in the study. Since the model simulates a PLCs working behavior, outputs are formulized with scan cycle write place added. Safety requirements of the system are given below with the corrections if the requirement is not satisfied.

SR1: The two robot arms must not operate concurrently. Since robots have overlapping operation routes, to avoid a possible and highly likely to occur collision, concurrent working must not be allowed.

$$AG\neg(ScanCycle_{write} \geq 1 \wedge OP\_1 \geq 1 \wedge OP\_2 \geq 1)$$

SR2: The robot arms cannot move horizontally while moving up. Difference between the heights of the conveyors can cause robot extremities to collide with the conveyor.

$$AG\neg(ScanCycle_{write} \geq 1 \wedge MH\_1L \geq 1 \wedge (MP\_10 \geq 1 \vee MP\_11 \geq 1))$$

SR3: The robot arms cannot move horizontally while moving down. Just like safety requirement 2, difference between the heights of the conveyors can cause robot extremities to collide with the conveyor.

$$AG\neg(ScanCycle_{write} \geq 1 \wedge MH\_1H \geq 1 \wedge (MP\_10 \geq 1 \vee MP\_11 \geq 1))$$

SR4: The robot gripper must not try to place its load while moving to conveyor 3. This can cause defective products, which means high cost depending on the value of the material being carried.

$$AG\neg(ScanCycle_{write} \geq 1 \wedge MP_{13} \geq 1 \wedge SG\_1 = 0)$$

SR5: The robot gripper must not try to place its load while moving to conveyor 3. This can cause defective products, which means high cost depending on the value of the material being carried.

$$AG\neg(ScanCycle_{write} \geq 1 \wedge MP_{13} \geq 1 \wedge SG\_1 = 0)$$

SR6: Robot must not move down to place its load when a product is present on conveyor 3. Same as safety requirement 4, this event can cause defective products, even worst it can cause damage to robots or the conveyor.

$$AG\neg(ScanCycle_{write} \geq 1 \wedge MH\_1L \geq 1 \wedge SP\_11 \geq 1 \vee SC\_3 \geq 1)$$

SR7: The robot should not carry a load while all conveyors are occupied. This event is referred to as a deadlock in the system where all allowed actions are blocked.

$$AG\neg(ScanCycle_{write} \geq 1 \wedge SC\_3 \geq 1 \wedge SC\_1 \geq 1 \wedge OP\_1 \geq 1)$$

Table 2. Verification Result Before and After Correction.

S. R. No	*Name of Safety Requirement*	*Verification Before Correction*			*Verification After Correction*		
		Mb	*Second(s)*	*Result*	*Mb*	*Second(s)*	*Result*
1	*Concurrent Working*	*43*	*0.021*	✓	*25*	*0.53*	✓
2	*Unauthorized Movement Up*	*43*	*0.085*	✗	*24*	*0.534*	✓
3	*Unauthorized Movement Down*	*23*	*0.223*	✗	*25*	*0.535*	✓
4	*Misplace*	*43*	*0.813*	✓	*24*	*0.53*	✓
5	*Lock*	*0*	*0.114*	✗	*26*	*0.517*	✓
6	*Collision*	*0*	*0.049*	✗	*0*	*0.546*	✓
7	*Deadlock*	*0*	*0.054*	✗	*0*	*0.051*	✗
System Info		*Intel Core I7-2630QM Cpu @2,00 Ghz x64 6 Gb Ram Windows 10 64-Bit*					
Verification tool		*TAPAAL 3.9.1*					

The power of this method comes from its straightforward application and ability to find errors that can be very rare or even caused by faulty sensors. Within the case study, intuitively written program, which was tested before with no errors, failed to satisfy almost all requirements except safety requirement 1 and 4. After verification, the results can be used to correct the program. Failure to satisfy requirement 2 and 3 means that unnecessary variables are used in rung 8 through 11, which can prevent robot from stopping under rare circumstances. Requirement 5 indicates that rung 8 through 11 and 13 fails at resetting when OP_1 is false due to the fact that before operation resets anyone of the set functions can be activated with no way to reset. Requirement 6 points out an obvious error where rungs, which run, place operation does not include information from the sensor of conveyor 3. Corrected versions of these rungs are given in Figure 9.

Since conveyors are under control of another control system, the possibility of a deadlock cannot be evaded by changing the program at hand. But since conveyors only carry materials in one direction the deadlock will not be permanent.

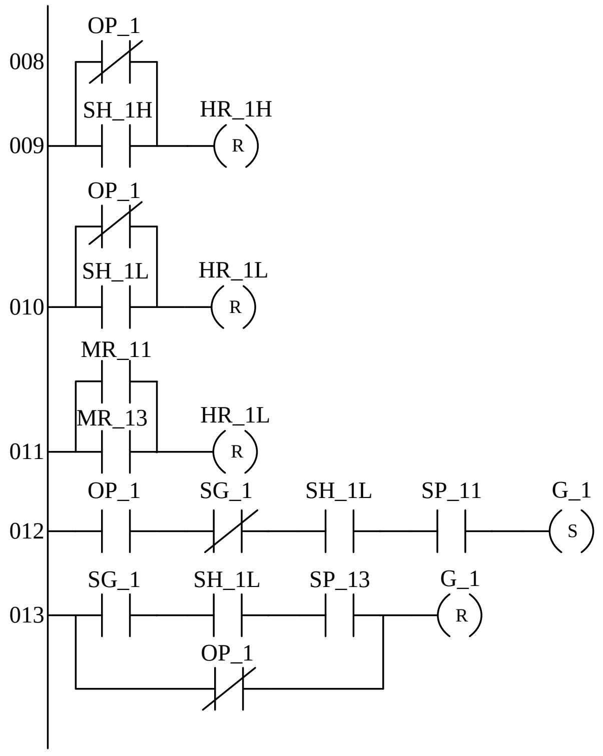


Figure 9. Corrected versions of faulty rungs.

4. Conclusion

Today, most of the code still developed for industrial control systems is written manually, usually using the ladder programming language. Although most of the code developed for industrial automation systems is based on basic logic relations, when the number of inputs and outputs and their possible combinations are considered, the entire code cannot be tested. Generally, the process is progressed with test scenarios based on white box testing and black box testing strategies. For this reason, code development platforms are content with developing interfaces where relevant test scenarios can be run according to these strategies. On the other hand, showing that the safety-related code meets the relevant safety conditions and proving that these conditions are met in all possible combinations will be a very valuable output for the safety of the process.orship) are not allowed. After receipt of the corrected proofs, the article in PDF format will be published online.

References

[1] Zhou, M.; Wan, H.; Wang, R.; Song, X.; Su, C.; Gu, M.; Sun, J. (2013). Formal component-based modeling and synthesis for PLC systems, *Computers in Industry*, Vol. 64, No. 8, 1022–1034. doi:10.1016/j.compind.2013.07.003

[2] Ljungkrantz, O.; Akesson, K.; Yuan, C.; Fabian, M. (2012). Towards Industrial Formal Specification of Programmable Safety Systems, *IEEE Transactions on Control Systems Technology*, Vol. 20, No. 6, 1567–1574Presented at the IEEE Transactions on Control Systems Technology. doi:10.1109/TCST.2011.2169262

[3] Vieira, A. D.; Santos, E. A. P.; de Queiroz, M. H.; Leal, A. B.; de Paula Neto, A. D.; Cury, J. E. R. (2017). A Method for PLC Implementation of Supervisory Control of Discrete Event Systems, *IEEE Transactions on Control Systems Technology*, Vol. 25, No. 1, 175–191Presented at the IEEE Transactions on Control Systems Technology.

doi:10.1109/TCST.2016.2544702

[4] Chen, C.; Hu, H. (2015). Maximally permissive distributed control of automated manufacturing systems with assembly operations using Petri nets, *2015 IEEE International Conference on Automation Science and Engineering (CASE)*Presented at the 2015 IEEE International Conference on Automation Science and Engineering (CASE), , 532–538. doi:10.1109/CoASE.2015.7294134

[5] List, G. F.; Mashayekhi, M. (2016). A Modular Colored Stochastic Petri Net for Modeling and Analysis of Signalized Intersections, *IEEE Transactions on Intelligent Transportation Systems*, Vol. 17, No. 3, 701–713Presented at the IEEE Transactions on Intelligent Transportation Systems. doi:10.1109/TITS.2015.2483324

[6] Wang, X.; Mahulea, C.; Silva, M. (2013). Fault Diagnosis Graph of time Petri nets, *2013 European Control Conference (ECC)*Presented at the 2013 European Control Conference (ECC), , 2459–2464. doi:10.23919/ECC.2013.6669417

[7] Rösch, S.; Ulewicz, S.; Provost, J.; Vogel-Heuser, B. (2015). Review of Model-Based Testing Approaches in Production Automation and Adjacent Domains—Current Challenges and Research Gaps, *Journal of Software Engineering and Applications*, Vol. 08, No. 09, 499–519. doi:10.4236/jsea.2015.89048

[8] Ovatman, T.; Aral, A.; Polat, D.; Ünver, A. O. (2016). An overview of model checking practices on verification of PLC software, *Software & Systems Modeling*, Vol. 15, No. 4, 937–960. doi:10.1007/s10270-014-0448-7

[9] Bauer, N.; Engell, S.; Huuck, R.; Lohmann, S.; Lukoschus, B.; Remelhe, M.; Stursberg, O. (2004). Verification of PLC Programs Given as Sequential Function Charts, H. Ehrig; W. Damm; J. Desel; M. Große-Rhode; W. Reif; E. Schnieder; E. Westkämper (Eds.), *Integration of Software Specification Techniques for Applications in Engineering: Priority Program SoftSpez of the German Research Foundation (DFG), Final Report*, Springer, Berlin, Heidelberg, 517–540. doi:10.1007/978-3-540-27863-4_28

[10] Mertke, T.; Frey, G. (2001). Formal verification of PLC programs generated from signal interpreted Petri nets, *2001 IEEE International Conference on Systems, Man and Cybernetics. e-Systems and e-Man for Cybernetics in Cyberspace (Cat.No.01CH37236)* (Vol. 4)Presented at the 2001 IEEE International Conference on Systems, Man and Cybernetics. e-Systems and e-Man for Cybernetics in Cyberspace (Cat.No.01CH37236), , 2700–2705 vol.4. doi:10.1109/ICSMC.2001.972974

[11] Alenljung, T.; Lennartson, B.; Hosseini, M. N. (2012). Sensor Graphs for Discrete Event Modeling Applied to Formal Verification of PLCs, *IEEE Transactions on Control Systems Technology*, Vol. 20, No. 6, 1506–1521Presented at the IEEE Transactions on Control Systems Technology. doi:10.1109/TCST.2011.2168607

[12] Nellen, J.; Driessen, K.; Neuhäußer, M.; Ábrahám, E.; Wolters, B. (2016). Two CEGAR-based approaches for the safety verification of PLC-controlled plants, *Information Systems Frontiers*, Vol. 18, No. 5, 927–952. doi:10.1007/s10796-016-9671-9

[13] Carlsson, H.; Svensson, B.; Danielsson, F.; Lennartson, B. (2012). Methods for Reliable Simulation-Based PLC Code Verification, *IEEE Transactions on Industrial Informatics*, Vol. 8, No. 2, 267–278Presented at the IEEE Transactions on Industrial Informatics. doi:10.1109/TII.2011.2182653

[14] Rankin, D. J.; Jiang, J. (2011). A Hardware-in-the-Loop Simulation Platform for the Verification and Validation of Safety Control Systems, *IEEE Transactions on Nuclear Science*, Vol. 58, No. 2, 468–478Presented at the IEEE Transactions on Nuclear Science. doi:10.1109/TNS.2010.2103325

[15] Park, S. C.; Park, C. M.; Wang, G.-N.; Kwak, J.; Yeo, S. (2008). PLCStudio: Simulation based PLC code

verification, *2008 Winter Simulation Conference*Presented at the 2008 Winter Simulation Conference, , 222–228. doi:10.1109/WSC.2008.4736071

[16] Patil, M. M.; Subbaraman, S.; Joshi, S. (2011). Exploring Integrated Circuit Verification Methodology for Verification and Validation of PLC Systems, *2011 International Symposium on Electronic System Design*Presented at the 2011 International Symposium on Electronic System Design, , 88–93. doi:10.1109/ISED.2011.47

[17] Koo, L.-J.; Park, C. M.; Lee, C. H.; Park, S.; Wang, G.-N. (2011). Simulation framework for the verification of PLC programs in automobile industries, *International Journal of Production Research*, Vol. 49, No. 16, 4925–4943. doi:10.1080/00207543.2010.492404

[18] Wang, R.; Guan, Y.; Luo, L.; Song, X.; Zhang, J. (2013). Formal Modelling of PLC Systems by BIP Components, *2013 IEEE 37th Annual Computer Software and Applications Conference*Presented at the 2013 IEEE 37th Annual Computer Software and Applications Conference, , 512–518. doi:10.1109/COMPSAC.2013.85

[19] Zhang, M.; Chen, C.-Y.; Kao, B.-C.; Qamsane, Y.; Shao, Y.; Lin, Y.; Shi, E.; Mohan, S.; Barton, K.; Moyne, J.; Mao, Z. M. (2019). Towards Automated Safety Vetting of PLC Code in Real-World Plants, *2019 IEEE Symposium on Security and Privacy (SP)*Presented at the 2019 IEEE Symposium on Security and Privacy (SP), , 522–538. doi:10.1109/SP.2019.00034

[20] Fernández Adiego, B.; Darvas, D.; Viñuela, E. B.; Tournier, J.-C.; Bliudze, S.; Blech, J. O.; González Suárez, V. M. (2015). Applying Model Checking to Industrial-Sized PLC Programs, *IEEE Transactions on Industrial Informatics*, Vol. 11, No. 6, 1400–1410Presented at the IEEE Transactions on Industrial Informatics. doi:10.1109/TII.2015.2489184

[21] Xiao, L.; Li, M.; Gu, M.; Sun, J. (2014). A hierarchy framework on compositional verification for PLC software, *2014 IEEE 5th International Conference on Software Engineering and Service Science*Presented at the 2014 IEEE 5th International Conference on Software Engineering and Service Science, , 204–207. doi:10.1109/ICSESS.2014.6933545

[22] Ulewicz, S.; Vogel-Heuser, B.; Ulbrich, M.; Weigl, A.; Beckert, B. (2015). Proving equivalence between control software variants for Programmable Logic Controllers, *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*Presented at the 2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA), , 1–5. doi:10.1109/ETFA.2015.7301603

[23] He, N.; Oke, V.; Allen, G. (2016). Model-based verification of PLC programs using Simulink design, *2016 IEEE International Conference on Electro Information Technology (EIT)*Presented at the 2016 IEEE International Conference on Electro Information Technology (EIT), , 0211–0216. doi:10.1109/EIT.2016.7535242

[24] Adiego, B. F.; Lopez-Miguel, I. D.; Tournier, J.-C.; Blanco, E.; Ladzinski, T.; Havart, F. (2022). Applying Model Checking to Highly-Configurable Safety Critical Software: The SPS-PPS PLC Program, *Proceedings of the 18th International Conference on Accelerator and Large Experimental Physics Control Systems*, Vol. ICALEPCS2021, 5 pages, 0.178 MB. doi:10.18429/JACoW-ICALEPCS2021-WEPV042

[25] Ljungkrantz, O.; Akesson, K.; Fabian, M.; Yuan, C. (2010). Formal Specification and Verification of Industrial Control Logic Components, *IEEE Transactions on Automation Science and Engineering*, Vol. 7, No. 3, 538–548Presented at the IEEE Transactions on Automation Science and Engineering. doi:10.1109/TASE.2009.2031095

[26] Bel Mokadem, H.; Bérard, B.; Gourcuff, V.; De Smet, O.; Roussel, J.-M. (2010). Verification of a Timed Multitask System With Uppaal, *IEEE Transactions on Automation Science and Engineering*, Vol. 7, No. 4, 921–932Presented at the IEEE Transactions on Automation Science and Engineering. doi:10.1109/TASE.2010.2050199