Modeling, Specification and Construction of PLC-Programs

E. V. Kuzmin and V. A. Sokolov
Demidov Yaroslavl State University, Sovetskaya str., 14, Yaroslavl, 150000 Russia
e-mail: kuzmin@uniyar.ac.ru, sokolov@uniyar.ac.ru
Received January 10, 2013

Abstract—A new approach to constructing reliable discrete PLC-programs with timers—programming based on specification and verification—is proposed. Timers are modelled in a discrete way. For the specification of the program behavior we use the linear-time temporal logic LTL. Programming is carried out in the ST-language according to an LTL-specification. A new approach to programming PLC is shown by an example. The proposed programming approach provides an ability of a correctness analysis of PLC-programs using the model checking method. The programming requires fulfillment of the following two conditions: (1) the value of each variable should be changed not more than once per one full PLC-program implementation (per one full working cycle of PLC); (2) the value of each variable should only be changed in one place of a PLC-program. Under the proposed approach the change of the value of each program variable is described by a pair of LTL-formulas. The first LTL-formula describes situations that increase the value of the corresponding variable, the second LTL-formula specifies conditions leading to a decrease of the variable value. The LTL-formulas (used for specification of the corresponding variable behavior) are constructive in the sense that they construct the PLC-program, which satisfies temporal properties expressed by these formulas. Thus, the programming of PLC is reduced to the construction of LTL-specification of the behavior of each program variable.

DOI: 10.3103/S0146411614070244

INTRODUCTION

Application of programmable logic controllers (PLCs) for systems controlling complex industrial processes makes exacting correctness demands to PLC-programs. Any software error is considered to be inadmissible. However, the existing development tools for programming PLC, for example, widely known CoDeSys (Controller Development System) [7], provide only usual debugging facilities through testing programs (not guaranteeing total absence of errors) by means of visualization of PLC-control objects. At the same time certain theoretical knowledge and experience of applying the existing developments in the field of formal methods of modeling and analysis of software systems are accumulated. The programming of logical controllers is a practical area, in which existing developments could have successful application. Successful application is understood as implementation of formal methods in the process of program design at the level of a well-functioning technology which is clear to all specialists involved in this process—engineers, programmers and testers. Being as usual of a small size and having a finite state space, PLC-programs are exceptionally convenient objects for the formal (including automatic) analysis of correctness.

Programmable Logic Controllers (PLCs) are a special type of a computer widely used in today’s industry (in automation systems) [5, 6]. The PLC is a reprogrammable computer, based on sensors and actors, which is controlled by a user program. They are highly configurable and, thus, are applied to various industrial sectors. PLCs are a classical example of reactive systems. The PLC repeats the execution of a user program periodically. There are three main phases for program execution (working cycle): (1) reading from inputs (sensors), (2) program execution, (3) writing to outputs (actors).

Programming languages for logic controllers are defined by the IEC 61131-3 standard. This standard includes the description of five programming languages: SFC, IL, ST, LD and FBD. The IL (Instruction List) language is a typical Assembler with the accumulator and transitions on labels. The ST (Structured Text) language is a high-level language. It syntactically corresponds to a slightly adapted Pascal. The LD (Ladder Diagram) language is a graphic language implementing structures of electric circuits. The FBD (Function Block Diagram) is very similar to the principal scheme of an electronic device on chips. The SFC (Sequential Function Chart) is a high-level graphical tool. SF Charts consist of steps and transitions

1 The article is published in the original.
which divide tasks into simple stages with a formally defined logic of system functioning. A transition firing is defined by a condition. Every step is associated with specific actions. Descriptions of the actions are performed in any language of IEC 61131-3.

The languages are simple, but sufficiently powerful tools for implementing tasks of PLC. This “simplicity” of the languages provides a possibility of application of all existing methods of program correctness analysis - testing, theorem proving [1] and model checking [2]—for verification of PLC-programs. Theorem proving is more applicable to “continuous” stability and regulation tasks of the engineering control theory, since an implementation of these tasks on PLC is associated with the programming of an appropriate system of formulas. Model checking is the most suitable for “discrete” tasks of a logical control, requiring PLCs with binary inputs and outputs. This provides a finite space of possible states of PLC-programs. The most convenient for programming, specification and verification of PLC-programs are ST, LD and SFC languages, since they do not cause difficulties for neither developers nor engineers and can be easily translated into languages of software tools of automatic verification.

Earlier in the article [3], a review of methods and approaches to programming “discrete” PLC problems was carried out on the example of constructing PLC-program for controlling a code lock. Convenience of program correctness analysis using the model checking regarding automatic verification tool Cadence SMV [8] was estimated for these approaches. Potential vulnerabilities of PLC programs and the difficulties of the program correctness analysis, arising in traditional approaches to programming, were identified.

In this article, a new approach to constructing discrete PLC-programs with timers (as the essential element of most PLC-programs) is proposed—programming by specification and verification. This approach to programming PLC provides a possibility of PLC-program correctness analysis by the model checking method. For the specification of the program behavior, we use the linear-time temporal logic LTL. Programming is carried out in ST-language according to an LTL-specification. Discrete modeling of a timer is carried out. A new approach to PLC-programming is shown by an example.

The further aim is to build software tools for modeling, specification, construction and verification of PLC-programs.

1. MODEL CHECKING

Model checking is a process of checking whether a given model meets a given specification. A Kripke structure is used to represent the program behavior. Typically, the specification contains safety requirements. A temporal logic formula express the program property.

A Kripke Structure on a set of atomic propositions P is a transition system \( \mathcal{S} = (S, s_0, \rightarrow, L) \), with a non-empty set of states \( S \), an initial state \( s_0 \in S \), a transition relation \( \rightarrow \subseteq S \times S \) which is defined for all \( s \in S \), and a function \( L : S \times 2^P \) labeling every state by a subset of atomic propositions. The Kripke structure represents the program behaviour.

A Path of the Kripke structure from the state \( s_0 \) is an infinite consequence of states \( \pi = s_0 s_1 s_2 \ldots \) where \( \forall i \geq 0 s_i \rightarrow s_{i+1} \). For a path \( \pi = s_0 s_1 s_2 \ldots s_i s_{i+1} s_{i+2} \ldots \) we have \( \pi' = s_i s_{i+1} s_{i+2} \ldots \) and \( \pi(i) = s_i \).

The linear-time temporal logic language is considered as a specification language. PLC is a classic reactive control system, which once run, must always have a correct infinite behavior. LTL formulas allow to represent this behavior.

The syntax of the LTL formula is given by the following grammar:

\[
\varphi, \psi ::= \text{true} | p_0 | p_1 | \ldots | p_n | \ldots | \neg \varphi | \psi \land \varphi | X \varphi | \psi U \varphi | F \varphi | G \varphi, \text{where } p_i \in P.
\]

An LTL formula represents a property of a path in the Kripke structure, which descend from an emphasized current state. The temporal operators \( X, F, G \) and \( U \) are interpreted as follows: \( X \varphi \) means that \( \varphi \) must hold at the next state, \( \psi U \varphi \) means that \( \varphi \) holds at the current or a future state, and \( \psi \) must hold up until this point, \( F \varphi \) means that \( \varphi \) must hold at some future state, \( G \varphi \) means that \( \varphi \) must hold at the current state and all future states. Operators \( F \) and \( G \) are derived and introduced for the convenience of specification: \( F \varphi = \text{true} U \varphi \), \( G \varphi = F \neg \varphi \). In addition, classical logical operators \( \lor \) and \( \Rightarrow \) will be used further: \( \varphi_1 \lor \varphi_2 = \neg (\neg \varphi_1 \land \neg \varphi_2) \), \( \varphi_1 \Rightarrow \varphi_2 = \neg \varphi_1 \lor \varphi_2 \).

Formally, the satisfiability relation \( \models \) of LTL-formula \( \varphi \) for some path \( \pi \) of a Kripke structure \( S \) on \( P \) is defined by induction:

\[
\pi \models \text{true}; \ \pi \not\models \text{false};
\]

\[
\pi \models p \text{ for } p \in P \iff p \in L(\pi(0));
\]