Softw Syst Model (2016) 15:937–960
DOI 10.1007/s10270-014-0448-7

THEME SECTION PAPER

# An overview of model checking practices on verification of PLC software

**Tolga Ovatman · Atakan Aral · Davut Polat · Ali Osman Ünver**

Received: 15 November 2013 / Revised: 23 July 2014 / Accepted: 9 December 2014 / Published online: 25 December 2014

**Abstract** Programmable logic controllers (PLCs) are heavily used in industrial control systems, because of their high capacity of simultaneous input/output processing capabilities. Characteristically, PLC systems are used in mission critical systems, and PLC software needs to conform real-time constraints in order to work properly. Since PLC programming requires mastering low-level instructions or assembly like languages, an important step in PLC software production is modelling using a formal approach like Petri nets or automata. Afterward, PLC software is produced semiautomatically from the model and refined iteratively. Model checking, on the other hand, is a well-known software verification approach, where typically a set of timed properties are verified by exploring the transition system produced from the software model at hand. Naturally, model checking is applied in a variety of ways to verify the correctness of PLC-based software. In this paper, we provide a broad view about the difficulties that are encountered during the model checking process applied at the verification phase of PLC software production. We classify the approaches from two different perspectives: first, the model checking approach/tool used in the verification process, and second, the software model/source code and its transformation to model checker's specification language. In a nutshell, we have mainly examined SPIN, SMV, and UPPAAL-based model checking activities and model construction using Instruction Lists (and alike), Function Block Diagrams, and Petri nets/automata-based model construction activities. As a result of our studies, we provide a comparison among the studies in the literature regarding various aspects like their application areas, performance considerations, and model checking processes. Our survey can be used to provide guidance for the scholars and practitioners planning to integrate model checking to PLC-based software verification activities.

**Keywords** Model checking · Programmable logic controllers · Program verification

Communicated by Prof. Einar Broch Johnsen and Luigia Petre.

This study is supported by The Scientific and Technological Research Council of Turkey within the project numbered 113E272.

T. Ovatman (✉) · A. Aral · D. Polat · A. O. Ünver
Department of Computer Engineering, Istanbul Technical University, Istanbul, Turkey
e-mail: ovatman@itu.edu.tr

A. Aral
e-mail: aralat@itu.edu.tr

D. Polat
e-mail: dpolat@itu.edu.tr

A. O. Ünver
e-mail: unverao@itu.edu.tr

## 1 Introduction

Programmable logic controllers (PLCs) can be seen as special kind of computers, which are capable of processing a high number of I/O operations conforming real-time constraints. A typical PLC's processing cycle is arranged in three distinct sections where the input data are read into memory, data in the memory are processed, and the output data are written. The essence in widespread usage of PLCs is the limited duration of this cycle where input processing and production of outputs should be produced in hard deadlines. This situation makes PLCs a key artifact in real-time automation and control processes like railway interlocking systems [30, 89], nuclear power plants [57, 59, 115], and manufacturing conveyors [19, 63].

Being a digital computer, a PLC needs to be programmed in order to serve different purposes in different areas of usage. Tremendous increase in the utilization of PLCs in the last decades has also risen the number of PLC manufacturers and hence the variety in programming/modelling aspects. During the development of PLC programming, the industry has gone a series of evolutions in program specification and the programming language to be used, forming an industry standard called IEC 61131 [61]. A part of this standard defines common programming elements like variables and data types for a total of four programming languages in an either graphical or textual format. Most of the tools and practices in today's PLC programming activities are based on these programming languages, namely Instruction Lists (IL), Structured Texts (STs), Function Block Diagrams (FBD), and Ladder Diagrams (LD).

With respect to conventional computer programming techniques, PLC programming is performed in low-level programming languages where bitwise operations and Boolean variables are frequently used. This situation makes understanding and debugging of PLC programs inherently hard, increasing the importance of testing and verification of PLC programs. Even more importantly, PLCs are mostly used in mission critical real-time systems where the flaws in the complete correctness of the system software generally lead to produce hazardous effects. These two aspects combined make it very important to verify the correctness of PLC software in a rigorous manner.

The intense need for the verification of PLC software has given formal methods a key aspect in this area for two main reasons. Firstly, because of the mathematical rigor behind the formal methods, it is able to provide proofs for the correctness of software/model under consideration. Secondly, it is possible to apply formal methods on different levels of abstractions of the target system, which makes it possible to work on early stages of the software design. This property of formal methods is useful since it is known that the most serious (and hard to fix) software defects are known to arise at the early design stage of software production. The two prominent methods in formal verification are known to be theorem proving and model checking. These methods are applied in a large context in PLC software verification; however, we will be focusing on model checking practices in this paper's context.

Model checking is a widely used formal method where the system to be verified is represented by a suitable model and the desired property to be verified is checked by systematically exploring all the possible states that the modelled system may go through in a brute-force manner. By considering all possible scenarios, the verified property can be guaranteed depending on the correctness of the system model. For instance, a common model checking practice is to build the model using a state transition system and to specify the temporal properties of the system using linear temporal logic (LTL) [93]. This way, the automated model checking process can be applied by performing an exhaustive state space search over the multiplication of the transition system and the temporal specification to be checked. The transition system used in model checking is constructed by parallel composition of software/hardware component models.

Model checking is specifically useful when it comes to the verification of PLC systems, because it can be easier to model low-level PLC software as state transition systems compared to conventional computer programming. Moreover, since the PLC programs need to be transferred frequently among different PLC hardwares, modelling takes an important place in PLC programming in abstracting away unnecessary details. Even more, standard graphical languages like FBDs or Sequential Function Charts (SFCs) [61] and widely accepted modelling languages like Petri nets [92] have the potential to be translated to transitions systems more easily.

Following the progress in PLC programming and model checking, a large number of integration studies have been carried out in the related fields during the last decades. Most of these studies were about the translation challenges between formalisms used in PLC programming and a specific model checking tool. In this paper, we present a broad overview of these studies and summarize the challenges faced in the course. We present a twofold classification in the paper: the main classification groups the studies according to the programming/modelling methodology used in expressing PLC program and the target model checker. A secondary classification for each programming/modelling methodology is also presented where a number of important aspects (e.g., application context, applied system size, performance, and automation level) about each study in the class are compared.

For our main classification, we have firstly used the five main programming languages in the IEC 61131-3 standard, which are Ladder Diagrams, ILs, ST, FBDs, and SFCs. In addition to those languages, we have also included the studies that use other formalisms, mainly Petri nets as well as PLC-Automata [24], condition/event systems [104], and a few others that are explained in Sect. 3.4. In our main classification, we have also included model checking languages that were used, mainly SMV [80] (NuSMV, CadanceSMV), Timed Automata [2] (UPPAAL, Kronos), Promela/SPIN [53], and a few others. As a result of this classification, we also aim to reveal, if exists, the relationships between certain PLC programming languages and model checking methods.

For our secondary classification, we have enlisted some important properties of the studies for each group of PLC programming languages. These properties include application areas, system size, performance evaluation (if any), level of automation, and specification representation. By examining the commonalities among these properties, we can comment about some certain characteristics of PLC programming languages from the perspective of model checking.

As a result of our classification study, we present important common challenges that are present in the examined studies and discuss future studies that can be fruitful in the research area such as the use of timer on-delays and classical state space explosion problem. Our findings also provide a general overview for the practitioners who wish to apply model checking on a PLC-based system. The state-of-the-art application area for model checking PLC programs is transforming programs represented by Functional Block Diagrams (FBDs) to either SMV or UPPAAL models depending on the necessity to consider time explicitly in the model. Our overview gives insight about the type of model checkers used for specific types of PLC programming languages, the size of the system for the model checking to be applied, and the obstacles that may be present during the process.

The rest of the paper is organized as follows: Sect. 2 provides an overview of the merits and weaknesses of PLC programming compared to conventional programming, and Sect. 3 introduces PLC programming techniques included in the paper. Section 4 begins with the explanation of research methodology and an overview of related surveys in the industrial automation and PLC programming and later present the main classification of the studies that will be covered in the paper. Overview of studies for each group of PLC programming language is presented in Sects. 5–8 together with the discussion of secondary classification results. We also review recent studies that practically apply PLC model checking on industrial-sized systems in Sect. 9. In Sect. 10, we present a discussion about the challenges present in the examined papers and clarify some open problems and future challenges. Finally, we conclude the paper in Sect. 11.

## 2 PLC programming

Historically, PLC programming has grown from the roots of ladder diagrams (see Fig. 4) almost directly modelling the early usage of relays in control systems [61]. The basic usage principles of LDs can be used to understand neatly how PLC programming works in general. In LDs, a series of on/off switches (relays) are used in conjunction and disjunction in order to connect PLC inputs to PLC outputs representing the control logic as a propositional logic formula. In PLC programming, inputs and outputs are predefined in PLC hardware making the programming simply a correct selection of inputs/outputs and application of control logic. Roughly comparing to conventional computer programming where a new input/output variable is defined as the program code evolves, PLC programs usually start with a full range of I/O as refinements are continuously applied during the development process.

Another major difference in PLC programming is the execution logic of the PLC program after it has been developed. As mentioned earlier, PLC programs follow a "read input"–"execute logic"–"update outputs" approach, which results in the re-execution of the PLC program in each execution cycle. In terms of execution, PLC programs are prepared under an inherent parallel execution approach, because of their basis in electrical control circuits. For instance in LDs, each relay lane (which is also called a rung) is executed in parallel. This execution approach and the large number of I/O makes PLC programs undesirably large and complicated making them very hard to debug and maintain.

There are approaches like SFCs (see Sect. 3.3.1) that can be used to abstract away subsections of LDs as blocks to provide a perspective to the overall PLC program. However, too much abstraction can be undesired for PLC programs since it can make the PLC program even harder to maintain during system failures. The main usage of PLCs and PLC programs is manufacturing, conveyor systems, and critical systems like power plants making long downtimes unacceptable. This situation brings up the preference of large program size and under-abstraction rather than longer debugging and maintenance durations.

All of the characteristics of PLC programming discussed above makes it an appropriate application area for model checking, because of the following three reasons such as: (i) The program logic of PLC programs can be easily transformed to propositional logic and state transition systems; (ii) PLC programs inherently run parallel; and (iii) PLC programs are mostly used in real-time systems making verification a more important issue. There also exists more recent techniques than LDs and SFCs in PLC programming, we enlist and examine them in the following section.

## 3 PLC program models

Among model checking practices in the area of PLC software, our main interest is the program code/model that was used as the main artifact when performing the translation to the modelling language of the model checker. During this translation process, PLC program models are also used as a basis in large number of studies instead of PLC program code. In these studies, PLC programs are generated either manually or automatically; however, translation to the model checking language is done using the models. For this reason, we also include a few modelling languages to our classification in addition to the standard PLC programming languages.

In the industry standard IEC 61131-3, two main groups of programming languages are included, namely textual programming languages and graphical programming languages. Our classification is mainly based on this definition; however, there also exist higher level of graphical modelling languages that provide more abstract models for the organization of PLC programs. One of those languages is SFCs which structure the internal organization of a control program, treating blocks of PLC program as components. SFCs are also included in IEC 61131-3 standard providing a step of higher-level modelling to programming languages defined in the standard. Having a strong formal basis and allowing concurrent execution modelling, Petri nets are another modelling language frequently used in the modelling of PLC programs.

Together with the mentioned PLC programming standards, we also treat SFCs and Petri nets as a separate class of studies in model checking PLC programs since there exists a large number of studies using these modelling languages as basis. In this section, we continue by giving brief overviews of these languages since we will be frequently mentioning properties of them through the remainder of the paper. In addition to these six categories, there are also some other studies that we aggregate into a distinct group including non-conventional ways of PLC program specification. We explain those studies in more detail in Sect. 3.4.

## 3.1 Text-based programming languages

The two text-based models introduced in the industry standard are ILs and STs. Both of these programming languages resemble conventional low-level programming languages and take their roots from the very early days of PLC technology.

ILs are the primary means of PLC programming similar to an assembly language in syntax. Instructions in the IL are imperative operations that may have parameters and use registers to store values. IL programs also use the very basic components of the PLC hardware during its operations. IL is very frequently used in translation to model checking languages since almost any other programming language used in PLC programming can be converted to IL programs. An example IL program can be seen in Fig. 1.

Structured Lists, on the other hand, take its roots from Pascal programming language allowing conditional and iteration statements included in the programs. Those kind of functionality can be realized by using jump statements in IL.

```
          LD      Speed
          GT      1000
          JMPCN   VOLTS_OK
          LD      Volts
VOLTS_OK  LD      1
          ST      %Q75
```

**Fig. 1** An example PLC program snippet written in Instruction List [74]

```
IF Speed > 1000 Then
  Volts := Volts - 10;
END_IF;
%Q75 := 1;
```

**Fig. 2** An example PLC program snippet written in Structured Text (equivalent of Fig. 1)

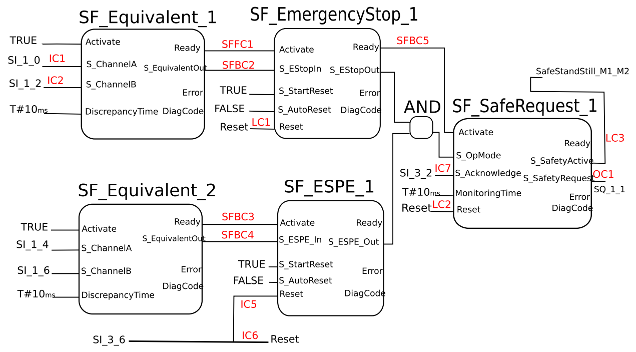


**Fig. 3** Function block network of a PLC software [103]

ST also defines a more convenient syntax for defining functions and function blocks forming a higher-level language when compared to IL. An example ST program can be seen in Fig. 2.

## 3.2 Graphical programming languages

### *3.2.1 Function Block Diagrams*

Function blocks are defined as the equivalent of integrated circuits for the PLC programs. They gather the functions supplied by the PLC to perform a specific functionality. These functions can be elementary blocks performing basic functions like move and compare or composite blocks that were constructed by connecting a set of functions. Having well-defined input and output, function blocks can be used like black boxes by the PLC programmers.

FBDs are the graphical structures that contain information about how the function blocks inside PLC program are related and how the information is going to flow among them. An example program with FBDs can be seen in Fig. 3. By their nature, FBDs mimic different levels of abstractions by encapsulating the elementary functions and interconnected function blocks. This makes FBDs very popular in model checking FBD programs, because reduction in system complexity by applying abstractions is one of the key practices in reducing the state space during model checking process.

### *3.2.2 Ladder Diagrams*

LDs, originally used for designing relay racks, have evolved into a programming language for PLC controllers in time. Also expressed as ladder logic or relay ladder logic, ladder

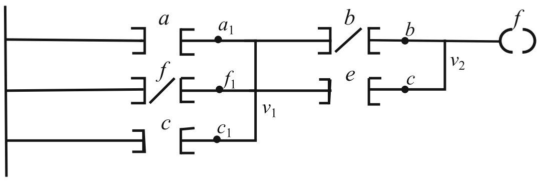


**Fig. 4** An example Ladder Diagram snippet [56]

diagrams actually are composed of a series of rules, called rungs, which can be executed sequentially during a PLC's cycle. The concept of rung can be seen as the basic building blocks of the ladder diagrams, so most of the literature on translating LDs to model checking models is centered around translating rungs.

There may be elements in each rung which are executed from left to right in a sequential way. This way, the output of each element in a rung becomes an input to another element. There may exist two important entities in each rung of a LD called coils and contacts. Coils are always to the rightmost side and act as a boolean variable output. On the other hand, contacts represent boolean input variables that may be either open or closed (negated). Connecting the elements in a rung in a serial way forms a logical conjunction while connecting in parallel forms a logical disjunction. Moreover, function blocks can be included in rungs for some PLC vendors programming tools as well.

For instance, the LD in Fig. 4 contains only one coil labeled as $f$ at the upper right side of the figure. The elements labeled $a$ to $e$ are contacts, and each horizontal line containing contacts on them is rungs; there exists three rungs in the example, which contains contact $a$–$b$, $c$–$e$, and $d$, respectively. LDs can be interpreted as propositional logic formulae easily, which makes this kind of interpretation a frequently studied topic in model checking PLC software. An example program along with LDs can be seen in Fig. 4. This piece of LD corresponds to the propositional logic formula $((a \vee \neg f \vee c) \wedge (\neg b \vee e)) \leftrightarrow f$

## 3.3 Modelling languages

### *3.3.1 Sequential Function Charts*

SFCs are defined as elements structuring the internal organization of PLC programs and function blocks. Most of the time, each block in an SFC contains a LD pointing to a lower-level abstraction in the PLC program. Not only SFCs are used to provide a broader view of the program with their structure similar to flowchart diagrams but also they can introduce parallelization by being able to represent multiple program flows within a single diagram. Moreover, SFCs were inspired by Petri nets and an older Grafcet standard, so that it would be more appropriate to categorize SFC-based studies separately

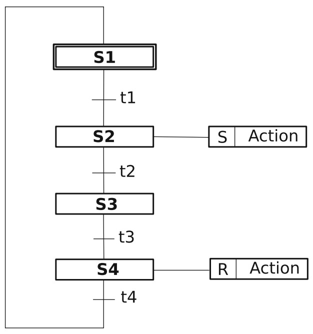


**Fig. 5** Steps and functions in a Sequential Function Chart method [39]

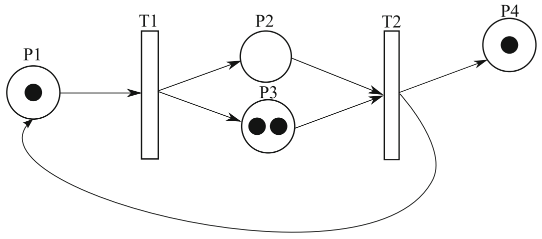


**Fig. 6** An example Petri net model

from programming language studies, but rather as modelling studies with Petri nets.

SFCs bring structure to the elements inside a PLC program by defining steps, linked with action blocks and transitions. Program flow is actually composed of a series of special "step transitions" whereby in each transition, the emerging step of the transition is deactivated and the next step is activated. An example to those steps and transitions of a SFC can be seen in Fig. 5. Steps can be linked with an action block, which performs a control action when a step is activated. Each step in this execution can be modelled as one of the standard programming languages defined above or as an SFC model recursively. SFC program explicitly represents the execution order of program component units, which can be arranged in an alternating and/or parallel way.

### *3.3.2 Petri nets*

One of the most commonly used formal modelling approaches in describing PLC programs is using Petri nets [92]. A Petri net consists of places, transitions, and arcs where each place may hold a number of tokens. Tokens are used to bring concurrency in Petri net execution, where a number of tokens can be transiting among places in a Petri net. Transmission of a token from one place to another is constrained by the transitions and arcs connecting places to each other. Two or more places can be connected to each other over one transaction and multiple arcs. In order to connect two places, there should be an arc from one of the places to a transition and another arc from that transition to the other place. An example program modelled with a Petri net can be seen in Fig. 6.

The tokens (indicated with black dots) inside places (indicated with “P” labels) are the basic elements used to model parallel executions by floating over the transitions (indicated with “T” labels) between the places. For the initial configuration, an arbitrary number of tokens can be present in the Petri net. Tokens can perform place transitions only if there exists a transition between two places. There may be multiple tokens inside a place at a time. During Petri net run, tokens perform transitions between the places at each step following the transition rules, which drive the parallel behavior of tokens. For example, if there exists a transition from a single place to a multiple number of places, the token is duplicated for each destination place. Conversely, if a transition connects many (assume *n*) places to a single destination place, there should be at least one token in each source place that is going to be merged with others in the destination place after the transition occurs.

Petri nets are frequently used in PLC program modelling and model checking purposes since they can be converted to PLC programs relatively easier than most of the other formal modelling approaches. Moreover, Petri nets are also frequently used in model checking purposes having a strong tool and analyzer support in the field. During modelling PLC programs, many Petri net variants like Signal Interpreted Petri nets and Colored Petri nets [58] are used.

### 3.4 Other approaches

Studies on model checking PLC programs are not limited to the standard and conventional techniques described above but also a large number of studies exist using different kinds of programming languages and modelling approaches. Below, we give a brief description for each of the concepts used during our analysis in Sect. 8.

- *PLC-Automata* A special extension of automata having formal temporal semantics defined with duration calculus. PLC-Automata [24] can be transformed directly to PLC executable code.
- *Timed Automata* Timed Automata [2] is originally a formal modelling methodology that is frequently used in model checking purposes. Instead of transforming the PLC software model to a model checking formalism, directly modelling the system using Timed Automata is preferred in a few studies.
- *Condition Event Systems* Condition/event system [104] is a discrete state formalism developed for modelling discrete event systems. It inherits functionality from Petri nets and can be directly model checked. Temporal variants of this approach are also used in some studies.
- *Unified Modelling Languages* Unified Modelling Language [97], originally developed to model object-oriented software intensive systems, is later extended to state chart models. This modelling approach is frequently used in modelling PLC software, and it has been used for model checking purpose as well in a couple of studies.
- *MATLAB State charts* Different adoptions of state charts [48] are present today, UML state charts and MATLAB state charts being widely used two adoptions. MATLAB state charts are used in conjunction with Simulink Design Verifier for PLC program verification.

In addition to those approaches directly applied in relation to PLC program verification purposes, there also exist a number of studies where researchers use their own PLC modelling approach or their own model checker tools in order to contribute to the challenges faced by the practitioners of the approaches listed above. Some of them combined different techniques listed above to gain advantage from strengths of each approach. There also exist some studies [22,23,50] where the approach is not directly related to PLC model checking, but the process or the outcome can be used in such purposes; therefore, we chose to mention them as well at the end of Sect. 8.

## 4 Research methodology and classification of practices

### 4.1 Previous surveys

Before examining in detail the model checking studies performed on PLC program verification, we would like to explain the research methodology we have followed for the research and review process of the papers we have included in the survey. We have started the research process with a few studies that perform a similar survey in the past about verification of PLC programs or alike.

In the study by Frey and Litz, verification and validation activities on PLC programming is discussed over a generic control design process model proposed by the authors [37]. In their study, they analyzed the integration process of formal methods in PLC programming and discussed various practices in different stages of this iteration process. Later, they classified the verification approaches, formalisms used, and methods applied during the integration process. Model checking was one of the methods in this classification among theorem proving, reachability analysis, and simulation.

Later, the dissertation by Huuck [54] contains a survey of model checking studies applied on PLC programs. His study focuses on developing formal approaches on PLC programs specified in IL and SFC and proposes a model checking approach based on translating SFCs to CadanceSMV models. In his study, Huuck also provided a discussion of model checking approaches for PLC programming. In his discussion, it can be seen that most of the studies in the area are performed over IL models and a few studies exist on other programming approaches at the time.

Finally, the study by Younis and Frey [116] provides a classification scheme for the works done in formalization of existing PLC programs. They classified the studies based on four criteria: sources used for the formalization, level of formalization, aim of formalization, and the formal model used to describe the PLC program. Although their discussion mostly include model checker formalisms as the targets of PLC program translation, they also mention a few approaches on static analysis and reverse engineering as well. Our study can be seen as updating and expanding their study.

The latest survey presented above is dated back to 2003; during the last decade, practices on model checking has evolved in a noticeable manner especially in the area of FBD translation. Earlier studies focus on verifying textual PLC programs or LDs where only boolean variables are used. Beginning from 2000s FBDs started to take over due to their more modular structure, ability to handle complicated programs more easily and availability of different types of variables.

Additionally, model checking tools are being continuously improved as well, most of the model checking tools have improved their efficiency and published new releases of their software. Moreover, the computing capabilities of hardware is also continuously increasing so it became possible to model check many complicated systems, which were not suitable for model checking before. An obvious example is the increase in studies aiming toward verification of real-time properties. Three quarters of the number of Timed Automata-based verification studies included in our surveys are performed after 2003.

Handling larger programs is another improvement that can be seen in latest studies. Even though the system sizes were being measured in terms of variables instead of function blocks in earlier days, latest studies report an improvement of a hundred times in larger sized systems. Comparisons in latter parts of our study show that FBD-based verification studies are now able to handle thousands of variables where the numbers were less than a hundred for the studies performed using LDs.

## 4.2 Research methodology

During our research process, we have used the surveys reviewed above as basis together with some very early studies published on the subject like the paper by Halbwachs et al. [45] and Moon et al. [84]. We have built our initial paper base by including all the papers reviewed by the surveys above and the early studies mentioned. We have applied a number of iterations by following the steps below until we were sure that the paper base is not expanding anymore.

1. Widely known electronic library resources (ACM Digital Library, Elsevier Science Direct, IEEE Explore, Springer Link and Wiley Online Library) are searched for the related papers cited by the papers in our paper base.
2. Widely known academic indexing sites (Citeseer, DBLP, Google Scholar, Microsoft Academic Search) are searched for the papers that cite the papers in our paper base.
3. Full range of academic studies of the authors that are present in our paper base is skimmed.
4. Mostly used keywords in our paper base (PLC, Model Checking, LTL, FBD, etc.) are searched in electronic library resources.
5. International Federation of Automatic Control's events and publications is directly searched.

After each iteration of the steps above, we applied a preliminary review and included the appropriate papers to our paper base. We keep track of both included and excluded papers to our paper base in order to rapidly eliminate any sort of duplicate reviews. At the end of our iterations, we have used the following criteria to be included in the detailed review process

– Studies that use a present model checking tool in verification of PLC software like SMV and UPPAAL.
– Studies that the authors have developed their own model checkers in the verification of PLC software
– Studies that apply model checking on the PLC software developed by the programming languages included in the IEC 61131 standard.
– Studies that use modelling languages (Petri nets, UML, etc.) in representing PLC software.

Following criteria are used to exclude papers from the paper base:

– Studies that use formal methods other than model checking like theorem proving.
– Studies that focus on test case generation, state reduction, and specification representation even though we mention and cite them whenever needed.

At the end of our review process, we have reviewed 78 papers where 54 of the papers were included in the proceedings of related symposia, conferences, and workshops; 16 of the papers were published in journals and the rest of the papers were technical reports, MSc/PhD thesis, and books. Interestingly, all of the journal articles included in our survey has been published in a separate journal. A more coarse-grained clustering can be done by the publishers of the journals where IEEE journals has the lead by five different journals. Even though the conference proceeding papers are more concentrated than journals, there are still 34 distinct conferences for the papers covered in our study. IEEE conferences cover a total of 31 papers published in the area; among those, IEEE Conference on Emerging Technologies and Factory Automation proceedings contains 6 of the covered papers, followed by IEEE International Conference on Systems, Man, and Cybernetics contains 5 of the covered papers. Another notable clustering is in IFAC conferences, where 4 different conference proceedings contain 5 papers covered in this survey.

## 4.3 Classification of practices

In this section, we present our main classification discussion on the studies that we are going to examine in more detail. Before presenting our main classification, we would like to mention the main model checking approaches and tools used in the studies that will be discussed. Briefly, three main set of tools used in a wide range of studies, which are

– SMV-based tools, which include NuSMV [17] and CadenceSMV [80]. Symbolic model checking techniques and binary decision diagrams [79] are applied in SMV-based tools. Those tools can verify properties written in both LTL and Computation Tree Logic (CTL).
– Timed Automata-based tools, which is mostly from UPPAAL family [69] or Kronos [117] in a few studies. Timed Automata is an extension of automata with a set of real-valued clocks. These clocks are actually positive integers that increase monotonically and in a synchronous way during automata run. Timed Automata-based tools are used to perform model checking on real-time system models and allow specifications in Timed CTL.
– SPIN model checker. SPIN is one of the major model checkers where the program models are expressed in Promela language and converted by SPIN to programs in C language to verify properties written in LTL.

Apart from those model checkers, there are also studies performed using model checker Tina [9] or authors' own model checker implementations.

Our main classification approach is examining the studies by the PLC programming or modelling language being used as the source for translation to the model checkers modelling formalism. As explained in the former section, we have used a similar taxonomy presented in the IEC 61131 standard with the only exception of widely used Petri nets examined in addition to SFC-based models. A diagrammatic representation of the taxonomy we use in our classification is presented in Fig. 7.

Second criterion in our main classification is the target model checking formalism and specific model checking tool

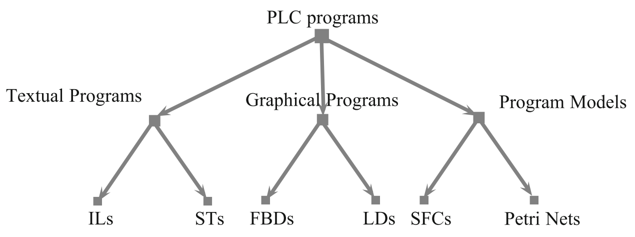


**Fig. 7** Main classification of PLC programs used in classifications

used. In Tables 1 and 2, we present a matrix of all the studies we have examined, grouped in rows of the matrix according to their PLC programming/modelling approaches and grouped in columns of the matrix according to the kind of model checking formalism they use.

In Table 1, examined studies are classified according to the programming language and the model checking tool used in the study. It can be seen that the mainstream model checking tool used in the studies is SMV followed by UPPAAL. The main difference between these tools is the real-time model checking capability offered by UPPAAL where real-time clocks can be included in the model. On the other hand, SMV allows model checking timed properties implicitly. In SMV, temporal properties can be expressed (using temporal logics) by referring to an implicit "current time," and the properties are specified relying on the ordering of events with respect to the current time. The studies that contain model checking real-time properties using UPPAAL include heavier discussion of abstraction and state space reduction compared to SMV-based studies.

In two of the Timed Automata studies, Kronos is used (indicated with "K" superscript) rather than UPPAAL. However, both of these studies are rather outdated, which can be interpreted as UPPAAL dominating the timed model checking studies in PLC model checking. For SMV, a more balanced distribution of choices is present between NuSMV and Cadence SMV both in terms of numbers and recentness. There are also earlier studies using earlier versions of SMV model checker based on binary decision diagrams.

Apart from these two mainstream model checkers, there are also studies that use Promela/SPIN, Tina, and other model checking tools, mostly authors' own implementations. In Table 1, studies that use SPIN are indicated with an "S" superscript and studies that use Tina are indicated with "T" superscript. Most of the SPIN-based studies are performed around year 2000 where the two Tina-based studies are relatively more recent compared to SPIN-based studies.

An interesting comment on Table 1 can be the excessive use of FBDs in recent studies. All of the studies that use FBDs are performed after 2007, and a great portion of these studies are performed after 2010. On the other hand, sequential function charts were mostly used between 2000 and 2005, they seem to be not preferred for more recent model check-

**Table 1** A general classification of studies performed in model checking PLC programs

| | Timed Auto. | SMV | | | SPIN$^{S}$, Tina$^{T}$ and Others |
|---|---|---|---|---|---|
| | UPPAAL and Kronos$^{K}$ | SMV | NuSMV | CadenceSMV | |
| Textual Programs | [113, 118] | | [41, 90] | [16] | [77]$^{S}$, [5, 10, 100] |
| Function Block Diagrams | [20, 29, 30, 103] | | [87, 89] | [57, 59, 115] | [5] |
| Ladder Diagrams | [83, 99, 119] | [94, 101, 106] | | [96] | [8]$^{T}$, [33] $^{T}$, [5, 84] |
| Sequential Function Charts | [75] $^{K}$, [7] | [12, 19, 39] | | [6, 7, 55] | [14] $^{S}$, [77] $^{S}$, [5] |
| Petri nets | | [81] | [38, 42–44] | [40, 63, 112] | [36] |

**Table 2** Classification of studies performed using nonstandard tools or languages

| | Timed Automata | SMV | Others |
|---|---|---|---|
| PLC-Automata | [26, 86] | | |
| Timed Automata | [67, 109, 114] | | |
| Condition/event systems | | [95] | [47, 66, 108] |
| State machines | [98] | [64] | |
| CFCs | [110] | | |
| Simulink and data/state flowcharts | | | [60, 78] |
| Other | [107] (REMES) | [45] (LUSTRE), [105] (tMPSG) | [111]$^{S}$ (VKRC) |

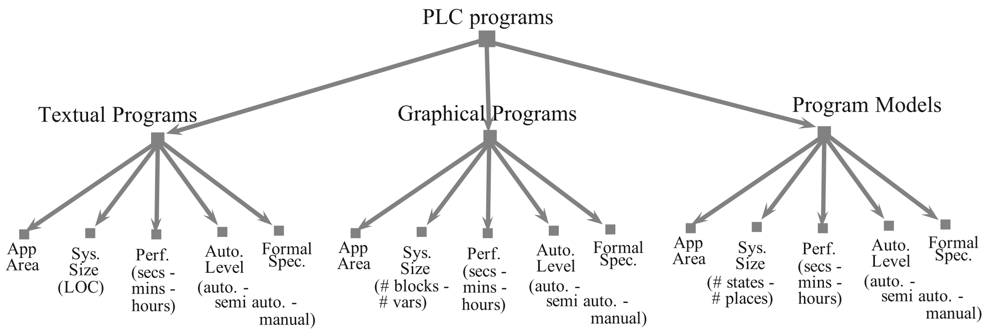

PLC programs
Textual Programs
Graphical Programs
Program Models
App Area
Sys. Size (LOC)
Perf. (secs - mins - hours)
Auto. Level (auto. - semi auto. - manual)
Formal Spec.
App Area
Sys. Size (# blocks - # vars)
Perf. (secs - mins - hours)
Auto. Level (auto. - semi auto. - manual)
Formal Spec.
App Area
Sys. Size (# states - # places)
Perf. (secs - mins - hours)
Auto. Level (auto. - semi auto. - manual)
Formal Spec.

**Fig. 8** Properties used in the classification of PLC program verification studies

ing studies. Another interesting point is the lack of Timed Automata studies using Petri nets. The capability of modelling timed properties using Petri nets explicitly can be the reason for the lack of such translations, the authors choose to either use Petri nets or Timed Automata when real-time modelling is needed.

In Table 2, we present a similar classification for the studies that do not use IEC 61131-3 standard programming languages or Petri nets. A considerable amount of studies exist that use a modelling language derived from finite state automata such as PLC-Automata and Timed Automata. Other approaches include data and state flowcharts and event systems where more recent studies are focused. Simulink is used in a couple of studies to perform verification using the built-in verifier. Timed automata is used more than SMV when nonstandard languages are considered, because most of the time used modelling language is automata or state transition-based making it easier to transform into Timed Automata models.

Both these tables are presented to give a brief overview of the relationship between the model checkers and the programming languages used in model checking PLC systems. In the following sections, we elaborate our overview by discussing the mentioned approaches above and making comparisons when possible.

In our classifications, we use five main properties of performed studies explained below to make a clear distinction among the individual studies. A graphical representation of the classified properties is also presented in Fig. 8.

1. *Application area* The usage area of the PLC system where the model checking is performed upon is identified with

this class. It can be interesting to check whether any relationships exist between certain usage areas and specific tools/techniques like the classic example of railway crossings and Timed Automata.
2. *System Size* The unit used in system size can change according to the PLC programming language in the study. For instance, for textual programs, the system size can be defined as the lines of program while for FBDs, it can be the number of function blocks in the system model. Some studies prefer to provide the size of the state space during the model checking process; however, this number can be subjective, because of the modelling tool and state space reductions used.
3. *Performance* The time spend during model checking process. We prefer to use a broad unit of measure for performance since the performance can change according to the system size and hardware used in model checking. We simply use "seconds," "minutes," or "hours" to indicate an approximate duration.
4. *Automation Level* We separate the studies that perform fully automatic process in converting the PLC program models to model checker's models. We present three classes of automation "automatic," "manual," and "semiautomatic" where in semiautomatic conversions additional interventions are performed over automatic conversions either to perform abstraction of small modifications.
5. *Formal Specification* Specification of the properties to be verified using a temporal specification language (LTL, CTL, TCTL, etc.) are also performed in an automatic way in some of the examined studies. We make a distinction by providing "automatic," "manual," and "semiautomatic" classes.

We have also included an additional comparison about the kind of properties that has been verified in the studies. Although the properties that were checked can be intuitively guessed from the type of model checking tool used in each study, some studies do not fully utilize the capabilities of the tools they use. A typical example is to use Timed Automata in modelling the system and not to include any timed properties in the specifications that are checked. Our comparison tables contain the following three types of data.

1. Real-time properties: This column is used to indicate whether any timed properties have been verified in the study.
2. Correctness properties: The property to be verified can either be an invariant that is used to verify regular correctness properties (indicated with capital I) or the property can be a safety property ensuring that an unwanted situation never happens (indicated with capital S) or it can be a liveness property ensuring the continuous execution of the system (indicated with capital L).
3. Specification logic: This column contains the temporal logic used in specification.

It should be noted that the properties above are indicated in the classification based on explicit examples in the paper instead of author claims.

## 5 Model checking textual PLC programs

Textual PLC programs are the means of PLC programming practices where we exhibit the earliest studies in formalization for model checking. This is quite natural, because text-based programs were being used earlier than FBDs and parsing them is more straightforward than LDs. One of the most prominent challenges in model checking textual PLC programs is reflecting timer on-delay instruction (TON) type of timers used in PLC programs for the purpose of ensuring the real-time properties. Basically, TON instructions are used to present a delay mechanism for their input signals. A TON's *true* input is reflected to its output only if the input signal is stable for a constant amount of time specified by the PLC programmer.

Also being one of the earlier studies in the area Mader and Wupper [76] study on transforming IL programs to Timed Automata models. They discussed the problem of TON timers and proposed two solutions: the first solution is using IL to program TON blocks and the counterpart is using automata to model TONs in the program. They indicated that it is more preferable to adopt the second approach since it is more modular and simpler. However, in the future work by Mader et al. [77], they have chosen to use Promela models instead of Timed Automata when performing model checking on an industrial case study. Interestingly, this study is also one of the few studies where SPIN is used in PLC program verification rather than SMV or UPPAAL.

Almost during the same years, Willems studies a similar TON problem and came up with a similar solution where he used Timed Automata to model TONs in the system [113]. Moreover, he has dealt with Zenoness issues that may arise and proposed solutions for such problems. Willems was able to reduce the state space size between 5-fold and 30-fold in his studies using Caesar/Aldebaran Development Package for performing state reduction on the produced models.

One of the latest studies that deal with the TON problem by using Timed Automata as well is by Zhou et al. [118] where the authors claim they have expanded Willem's work. In their study, Zhou et al. used four different modules namely "Coordinator" to model PLC synchronization, "Program" for PLC program, "Environment" for I/O, and "Interruption" to model time-based interruptions. Even though the first three

**Table 3** A classification of studies performed with textual PLC programs

| | App. area | Syst. size | Performance | Auto. level | Formal spec. |
|---|---|---|---|---|---|
| [113] | N/A | 18 lines | N/A | Automatic | N/A |
| [16] | Tool changing | 89 lines | N/A | Automatic | Manual |
| [77] | Batch plant | N/A | Minutes | Manual | Manual |
| [41] | N/A | 4000 vars | Seconds | Automatic | N/A |
| [90] | Counter | N/A | Minutes–hours | Semiauto. | Manual |
| [100] | Counter | N/A | Seconds | Automatic | Manual |
| [118] | N/A | N/A | N/A | Automatic | Manual |

modules are conventionally present in most of the studies in this area, interruption module is specific to this study.

Timed Automata is not the only formalism used in IL model checking. There are two studies that utilize SMV models in order to model check IL programs as well. The earlier study by Canet et al. [16] deals with single smaller modules and does not consider the timers during their studies. On the other hand, Pavlovic et al. [90] include interesting discussions in their study where they provide a meta description of the IL language to their translation process to be able to adapt the possible modifications in IL standard in the future. Their discussion also references a method by Peleska and Haxthausen [91] to check the behavioral equivalence of their formal models with the original PLC program. They also use SMV as the model checking formalism.

Another important commonality in textual program transformation studies is the dominant usage of IL over ST. Most of the papers below use mainly IL and some of them use ST programs secondarily in the transformation process. Gourcuff et al. use ST and perform dependency analysis between the variables in the program before transforming the program to SMV models [41]. They have also compared their results with DeSmet et al.'s study [101] where the results show significant improvement.

Lastly, we would like to mention Schlich et al.'s study where IL programs are model checked directly without using any conventional formalism [100]. They have used "concrete" and "abstract" simulators to generate state spaces from IL programs where concrete simulators generate state(s) for each PLC cycle. On the other hand, abstract simulators aggregate suitable states to reduce the state space. Their results show that the same example in Pavlovic et al.'s study [90] can be checked in 6 s where in Pavlovic's work process was taking 8 h. They also compare their work with another study by Huuck [55] and show significant improvements as well.

Examining Tables 3 and 4,[1] we see that the studies that explicitly state the size of the system (in terms of number of lines) do not consider the performance of the model checker. Conversely, the system size is not mentioned for the studies that mention the system performance. Nevertheless, it can be inferred from the results that model checking is performed in acceptable times for most of the studies. Model checking a 90-line program may not seem large enough for realistic systems; however, undersized experiments are unavoidably common for the model checking case. An automatic translation between PLC program and model checker's input language is performed most of the time due to the ease of parsing textual programs. On the other hand, none of the studies mention automatic generation of specifications to be checked compared to a few studies present for other programming models. Even though the reason behind this situation can be the low-level nature of textual representations that does not contain any abstractions, automatic specification extraction area seems to be an open area for this field of study.

Among all the studies discussed above, Willems et al.'s study and Zhou et al.'s study are the only ones that used timed models in model checking process distinguishing them from other studies. These studies can handle simpler programs compared to the more recent studies by Pavlovic et. al's and Schlih et al.'s studies being able to handle much larger state spaces. Lastly, we would like to mention Mader et al.'s study including very detailed examples on a practical case study.

**Table 4** Properties checked when model checking textual PLC programs

| | Real time | Correctness | | | Spec. logic |
|---|---|---|---|---|---|
| | | Invariance | Safety | Liveness | |
| [16] | No | Yes | Yes | Yes | LTL |
| [77] | No | Yes | No | No | LTL |
| [100] | No | Yes | No | No | CTL |
| [118] | Yes | Yes | No | No | CTL |

## 6 Model checking graphical PLC programs

### 6.1 FBD programs

The most recent studies on model checking PLC programs are performed on FBD programs; almost all of the studies

[1] Papers that do not include explicit information were omitted.

**Table 5** A classification of studies performed with FBDs

| | App. area | Syst. size | Performance | Auto. level | Formal spec. |
|---|---|---|---|---|---|
| [59] | Nuclear plant | 16 blocks 7 vars | N/A | Semiauto. | N/A |
| [20] | Hydrogen gen. Unit | 19 blocks 12 vars | N/A | Automatic | N/A |
| [115] | Nuclear plant | 1,500 blocks 1,000 vars | N/A | Automatic | Manual |
| [89] | Railway interlock | 100 vars | Minutes | Automatic | Manual |
| [103] | Safety app. | 6 blocks | N/A | Automatic | N/A |
| [29,30] | Train control | 30 blocks | $<1$ s | Automatic | Automatic |
| [57] | Nuclear plant | 20,000 blocks 9,000 vars | N/A | Automatic | N/A |
| [87] | N/A | N/A | N/A | Manual | Manual |

examined below belong to the last 5 years, which point to FBDs being the most recently used means of PLC program verification in this context. When it comes to model checking, formalisms studies almost split in half in using either SMV or UPPAAL models.

The work of Jee et al., Jeon et al., and Yoo et al. all focuses on model checking a PLC program of a nuclear power plant control system by using Verilog models and CadenceSMV for the model checking process. Again, common to all studies, a rule-based engine is used in performing translations. Jeon et al.'s and Jee et al.'s study specifically focuses on producing more understandable counterexamples [57,59] since the output of their tool produces tables of values for all the variables in the system. To provide a consistent translations from FBDs to model checker's language, both studies required assumptions on FBD programs like predefined execution orders and type safety. On the other hand, Yoo et al. [115] use VIS verification technique [13] to check the conformance of behaviors between their FBDs and Verilog models.

Pavlovic and Ehrich [89] use their own intermediate format, which they call *tFBD* after they transform their FBD program into a text-based representation they call *textFBD*. By defining operational semantics of *textFBD* format isomorphic to FBD semantics, they assure the equivalency between models in their translation process. They claim that their tFBD format, which is based on compacting *textFBD* to propositional logic formulae, dramatically reduces the state space during the model checking process realized with SMV. The compaction process combines chains of assignments (which is frequently present in FBDs) into single assignments reducing the amount of variables (especially temporary variables) used in *tFBD*'s logic formulae. They apply their method in the area of railway automation to a small and a more general case study and report that the state space is reduced in a dramatic way from $10^{65}$ states to $10^{14}$ states for one of their examples.

Pakonen et al. also work on translating FBD programs to SMV models for verification purposes; their work is focused on generating an Eclipse-based tool, which does not perform automatic transformations but provide vendor independence [87].

**Table 6** Properties checked when model checking FBD-based PLC programs

| | Real time | Correctness | | | Spec. logic |
|---|---|---|---|---|---|
| | | Invariance | Safety | Liveness | |
| [89] | No | No | Yes | No | CTL |
| [29,30] | No | Yes | Yes | No | CTL |
| [57] | No | Yes | Yes | No | N/A |

One of the studies which use UPPAAL and Timed Automata for model checking is the work by Soliman et al. where a rule-based transformation engine is used to transform safety function blocks, connections, inputs, and triggers in an FBD separately into Timed Automata models [103]. Unfortunately, their paper does not contain a detailed evaluation on the effectiveness of their work.

Two studies we have included in this comparison focus on generating test suites using the model checking tool UPPAAL to be used in PLC testing. Even though authors do not directly apply model checking, their approach can influence researchers and practitioners working in model checking PLC programs. The first study in this particular area is performed by Enoiu et al. [29,30] where they were able to generate 40–50 state test suites in less than a second from 30 FBD PLC programs. Another study by Silva et al. also focuses on test generation from standard FBD programs [20] by generating a synchronization automaton to represent PLC cycles and a behavior automaton for each FBD.

Tables 5 and 6[2] summarizes the important aspects about the model checking studies on FBD programs. Translation from FBD to model checker language is done automatically for most of the cases. Verification of systems up to the level of thousand blocks was possible even though the time required for verifications is not explicitly included in most of the studies.

[2] Papers that do not include explicit information were omitted.

### 6.2 LD programs

A wide range of studies exist for model checking LD programs spanning over SMV and UPPAAL models as well as Tina model checker. LD program-based studies span over time as well, there exists very early studies that utilize LDs as well as recent studies. For the case of LD model checking, the most recent studies generally use UPPAAL where earlier studies chose to use SMV variants to perform model checking.

Very early studies by Turk et al. and Probst et al. both use NuSMV as model checker and also they both use relay logic ladders, the early versions of LDs. Turk et al. [106] discussed the main challenge as transforming to the implicit time domain present in SMV where Probst et al. [94] handle the issue by modelling the hardware and nondeterministic human behavior separately to produce more realistic inputs during model checking process.

A more recent study by Smet and Rossi [101] uses a Python-based parser to transform each rung in LDs to a separate SMV model while the main challenge is to conserve the connections between the rungs among SMV models. Even though the details of the transformation process are not provided, a considerable amount of discussion is provided on specifying temporal properties of the system. By using their system, they were able to model check three different systems having between 27K and 16M states during model checking. The model checking process is performed between 1 s and 4 and a half hours, respectively.

Rossi et al. [96] have used CadenceSMV as a tool and translated text representation of LDs using a 6 ruled transformation engine. Although the paper does not contain a clear evaluation of the approach, they focus on TON semantics and how they can be handled in a large context.

Zoubek and Schnoebelen [119] also focused on TONs in their paper, but they chose Timed Automata and UPPAAL for model checking. Conventionally, they model the user input, the program behavior, and the PLC cycle separately, but their transformation works on a total of seven different instructions. An important contribution of their work is to use program slicing in reducing the state space of the produced system. They also provide additional manual abstractions to further shrink the state space. As a result of their studies, they were able to reduce the model checking duration of their case study to a few minutes, which normally took unmanageable amount of time.

UPPAAL is also used by Sarmento et al. [99] in their studies, but the models do not include explicit time properties. They use a finite state intermediate model, which contains integer and Boolean variable annotated transitions. They provide a seven-step modelling procedure for their methodology; however, their discussions do not include any aspect about automation of their process. Recently, in Mokadem et al.'s study [83] on multitasking, PLCs are model checked using Mader–Wupper model [76] with further manual modifications as an intermediate model to effectively handle TON timers and reduce state space.

Specific to LD model checking, two studies have used Tina as model checker. In Bender et al.'s study [8], a model-driven approach is applied by using ATL [62] transformations over LDs to produce timed Petri nets, which then can be automatically transformed to Tina models. A rule-based translation is used in LD translation, and race conditions are handled by checking whether the stabilizing inputs yield to stable outputs. The stability of the output variables is checked by using two timed Petri net places for each variable's true and false state, respectively. Absence of race conditions is checked by observing stabilized outputs as a result of stabilized inputs. Authors claim they were able to reduce the state space of a six actuator seven sensor system from 7 million states to 40 states using Tina. A later study with Tina is carried out by Farines et al. [33] where a model-driven engineering approach is used in the transformation of LDs to an intermediate form of FIACRE platform (a timed transition system in particular). Their work is very similar to Bender et al.'s work except the intermediate format they use.

Lastly, it is worth to mention James et al.'s study [56] where LDs are utilized to produce LTL formulas to be used in model checking. In most of the studies examined in this paper, such specifications are produced manually, which makes this study more valuable.

Model checking LD programs are applicable on systems having less than 100 variables as the Table 7 depicts. For most of the studies, the process was completed in an order of minutes while automatic translation is performed in around half of the studies. In the study by Bender et al. [8], performance was discussed over the size of the Petri net model, which is used as an intermediate format so it was considered likewise in the comparison table as well. In Table 8, we can see that most of the studies include explicit examples of the properties (especially safety properties) that were checked. Most of the time, CTL is used as a specification language, a few studies verify timed properties and use TCTL.

## 7 Model checking PLC program models

### 7.1 SFC models

Active research on using SFC models for model checking purposes mostly fall between 2000 and 2005. Even though SMV models have been the primary focus in SFC translation studies, there also exists work using Timed Automata in the process.

An early work by L'Her et al. [75] uses Kronos tool for model checking process by inferring temporal properties of

**Table 7** A classification of studies performed with LDs

| | App. area | Syst. size | Performance | Auto. level | Formal spec. |
|---|---|---|---|---|---|
| [94] | Screw conveyor | 74–93 vars | Minutes | Semiauto. | Manual |
| [106] | Chemical plant | 24–93 vars | Minutes | Manual | Manual |
| [96] | N/A | N/A | N/A | Automatic | Manual |
| [101] | Machining line | 30 vars | Seconds–hours | Automatic | Manual |
| [119] | Pumping line | 39 vars | Minutes | Semiauto. | Manual |
| [8] | N/A | 23–31 places | N/A | Automatic | Automatic |
| [99] | Gas burning equipment | N/A | N/A | Manual | Manual |
| [83] | Pinion identifier | N/A | Seconds | Manual | Manual |
| [33] | Pneumatic | N/A | Seconds | Automatic | Manual |

**Table 8** Properties checked when model checking LD-based PLC programs

| | Real time | Correctness | | | Spec. logic |
|---|---|---|---|---|---|
| | | Invariance | Safety | Liveness | |
| [94] | No | Yes | Yes | Yes | CTL |
| [106] | No | Yes | Yes | Yes | CTL |
| [96] | No | Yes | Yes | Yes | CTL |
| [101] | No | Yes | Yes | Yes | CTL |
| [119] | Yes | Yes | Yes | No | TCTL |
| [8] | No | Yes | Yes | No | LTL |
| [99] | No | Yes | Yes | No | CTL |
| [83] | Yes | Yes | Yes | Yes | TCTL |
| [33] | No | No | Yes | No | CTL |
| [56] | No | Yes | Yes | No | Lustre |

SFC diagrams. In their paper, authors examine the corresponding elements in Timed Automata models for sets of activities that can be present in SFC diagrams. They apply their approach on an 8-state SFC, but the resulting model could not be checked by Kronos since it includes 250K transitions, way too much for Kronos to check. They were able to reduce the state space to around 100 transitions by adding constraints on the translation process and by eliminating unnecessary variables in SFC states.

Couffin and Lesage [19] have performed translations to SMV models by expressing the behavior in SFC steps using propositional logic formulae. These formulae are later used in building state transition conditions of the automata in SMV models. The largest state space checked by the authors contains $10^6$ states, taking 4 s to be model checked. Citing this work, a paper by Smet et al. [102] also mentions SFC-based model checking and summarizes the research group's many PLC model checking studies in their paper.

The study by Fujino et al. [39] mainly performs simulations on Petri net models translated from SFC diagrams. They also claim they were able to easily convert Petri net models into SMV models and perform model checking, but did not include a detailed evaluation of this process. Many different techniques are used in combination by Brinksma et al. [14], where they perform translation from SFC and IL initially to SPIN models and perform state reduction by selecting states belong to cost optimal schedules generated using UPPAAL CORA. Another study by Bornot et al. [12] focuses on the reachability properties of SFC steps and did not include output variables in their SMV models in this respect .

CadanceSMV was commonly used in more recent studies on SFC model verification. Bauer et al., in two different studies, have verified nuclear power plant control programs [6,7]. In the earlier study, they have used variables to represent actions while in the latter one they were able to use automata for this purpose. Their main challenge in the second study was eliminating malformed sequences in SFC sequences, which they perform by searching for predefined subgraphs in the graphs generated from SFC models. They were able to model check a model with 40 different automata in about 15 min.

Finally, Huuck et al. [55] try to identify unsafe and unreachable states in SFC models. They use model checking to search for some rules that violate safeness of the model, which can be done by reachability analysis in state space generated by model checking. They claim they obtain successful results even for large SFCs, but did not explicitly include how large the checked SFC were.

Table 9 compares the studies on SFC programs where model checking was feasible at around ten states. For two of the studies, the model checking was evaluated using the number of states generated during the model checking process indicated with transition system (TS) states. The level of automation is above average in SFC-based studies as well, where most of the studies perform at least a semiautomatic translation where abstractions are applied after the automated translation process. In Table 10, we can see that in model checking SFC-based programs, generally CTL is used. An interesting fact is none of the studies on model checking SFC

**Table 9** A classification of studies performed with SFCs

| | App. area | System size | Performance | Auto. level | Formal spec. |
|---|---|---|---|---|---|
| [75] | Production Cell | 7 SFC states | N/A | Semiauto. | Semiauto. |
| [12] | N/A | 8 SFC states | N/A | Automatic | Manual |
| [14] | Batch plant | 24 SFC states | seconds | Manual | N/A |
| [39] | Cooling and alarm | 4–8 SFC states | N/A | Semiauto. | Manual |
| [19] | Manufacturing | $10^6$ TS states | Seconds | Semiauto. | Manual |
| [6] | Chemical plant | 14 SFC states | Seconds | Automatic | Manual |
| [55] | N/A | N/A | Seconds | Automatic | Manual |
| [7] | Chemical plant | 14 SFC states | Minutes | Automatic | Manual |

**Table 10** Properties checked when model checking SFC-based PLC programs

| | Real time | Correctness | | | Spec. logic |
|---|---|---|---|---|---|
| | | Invariance | Safety | Liveness | |
| [75] | No | No | Yes | No | CTL |
| [14] | No | Yes | No | Yes | LTL |
| [12] | No | Yes | Yes | Yes | CTL |
| [102] | No | Yes | Yes | Yes | LTL |
| [39] | No | Yes | No | No | CTL |
| [19] | No | Yes | No | No | CTL |
| [6] | No | Yes | Yes | Yes | CTL |
| [55] | No | Yes | Yes | No | CTL |
| [7] | No | Yes | Yes | Yes | CTL |

programs verify real-time properties, although SFC models inherently involve parallelism.

## 7.2 Petri net models

There exists a huge number of studies on model checking Petri nets in the literature not only focusing PLC program models but also embedded systems, and many other areas where Petri nets are used. We will be focusing on the ones that explicitly mention the area of application as PLC program verification in our discussions.

Earlier studies also discuss how Petri nets can be used in PLC program modelling purposes. For instance, in Frey and Litz's work [36], the usage of signal interpreteds Petri nets (SIPN) in control systems and the usage of their verification tool Netmate on a dissolving tank example are demonstrated. In a later study of Frey on SIPNs, he utilizes hSIPN (hybrid SIPN) to be able to fold states of Petri nets and presents analysis on such models [35]. Frey and Wagner [38] apply model checking in a later short paper by presenting a multipurpose PLC programming toolbox. In SIPN toolbox, the capability of exporting to NuSMV models is also present.

In this area, Frey has also co-authored many other papers. In Mertke and Frey's work [81], an extensive study of modelling PLC programs and the behavior of the environment is performed. They have combined Petri net model of the PLC and the environment to perform a complete model checking practice. Moreover, they also transform the timed requirements to be checked from a German semiverbal presentation, which is not frequently performed for the studies in PLC model checking.

The work of Weng et al. contains formalization of Petri net places. Inputs and outputs to CadanceSMV models [112] are discussed on the control system of an air chamber. In their study, Klein et al. adopt hSIPN approach [35] and perform CadanceSMV transformation with the approach in Weng's study on the verification of a manufacturing system [63]. They also transform SIPN models to SFC models and remodel check to validate their approach.

Apart from Frey et al.'s work, an earlier study is performed by Heiner et al. where Petri nets are not used in directly modelling PLC programs but as an intermediate format to be generated from IL program [51]. However, their study only includes this translation process, model checking practice is left as a future work. On the contrary, a recent study by Gergely et al. [40] performs translation to SMV models manually, but discusses the model checking process and focuses on the specification of the properties with CTL.

In a series of studies by Grobelna et al., control interpreted Petri nets are transformed to a rule-based textual intermediate format called logical models and transformed to NuSMV models by a rule-based translation engine [42–44]. Lastly, Nemeth et al. translate FBDs to colored Petri nets to use them as intermediate formats in model checking a nuclear power plant's control system [85].

Interestingly, in none of the studies, we have discussed a sound performance evaluation is present for Petri net-based model checking as shown in Table 11. This situation drives us to assume that Petri nets with around 5–20 places can be model checked in applicable durations. A single study where a Petri net with around 50 states was model checked was performed by Klein et al. [63]; however, the authors state in

**Table 11** A classification of studies performed with Petri nets

| | App. area | Syst. size | Performance | Auto. level | Formal spec. |
|---|---|---|---|---|---|
| [36] | Dissolv. tank | 6 places | N/A | N/A | N/A |
| [112] | Air chamber | 5 places | N/A | N/A | Manual |
| [81] | Air chamber | 5 places | N/A | Automatic | Automatic |
| [63] | Manufact. | 55 places | N/A | Automatic | Manual |
| [38] | N/A | N/A | N/A | Automatic | Semiauto. |
| [40] | Mixing tank | 5 places | N/A | Manual | Manual |
| [42,44] | Fluid mixture | 9 places | N/A | Manual | Manual |
| [43] | Drink prod. | 20 places | N/A | Automatic | Manual |

**Table 12** Properties checked when model checking Petri net PLC program models

| | Real time | Correctness | | | Spec. logic |
|---|---|---|---|---|---|
| | | Invariance | Safety | Liveness | |
| [81] | No | Yes | Yes | Yes | CTL |
| [112] | No | Yes | Yes | No | CTL |
| [63] | No | Yes | Yes | No | LTL |
| [40] | No | Yes | Yes | Yes | CTL |
| [42,44] | No | Yes | Yes | No | CTL |
| [43] | No | Yes | Yes | Yes | CTL |

their work that additional abstractions can be necessary when transforming from Petri nets to PLC models. These additional abstractions are applied by compacting repeated structures inside the Petri net into a single place. Table 12[3] contains similar results to the SFC property verification results, none of the studies explicitly include verification of real-time properties and most of them use CTL as specification logic.

## 8 Other approaches that use model checking

In spite of the large number of studies using IEC 61131 standards, PLC model checking is not limited to translation from standard PLC programming languages and modelling languages to a limited set of model checking tools. There also exists a large number of studies using a variety of different formats/tools to make the approach more effective and easy to apply for the community. Additionally, exploring the state space of PLC programs is not limited to model checking, it can also be used to generate a wide range of test cases, inspiring a number of academic research in the area.

PLC-Automata are a specific type of automata, which can define machines that periodically polls inputs and operate on them [25]. Formal semantics of PLC-Automata has been defined in duration calculus, and such automata can be directly translated to PLC programs. Dierks et al. performed translation from PLC-Automata to Timed Automata models and validate their translation by verifying the same set of properties with duration calculus and translated Timed Automata [26]. They have used Kronos and UPPAAL for model checking and show that model checking is viable for tiny and small systems. Olderog et al. transformed constraint diagrams obtained from user specifications to PLC-Automata and use PLC-Automata as an intermediate format [86]. Model checking PLC-Automata is performed by translating PLC-Automata models into Timed Automata models using Moby/PLC, a tool developed by Dierks and Tapken [27].

There also exist some other studies that use directly modelling PLC and its environment using Timed Automata. For instance, Wang et al. [109] used UPPAAL to model check a controller that control the motions of a theater steeve that lights, screen, and curtains are adorned to. Witsch et al. [114] performed a similar study by modelling PLC-based ethernet controllers directly using Timed Automata. Another study is by Lahtinen [67] where he checked an arc protection control system modelled in Timed Automata. He presents a satisfying evaluation of the Timed Automata models and memory consumption/model checking time. Even though these studies' subjects are PLCs and they use model checking, they did not perform a full integration of formal methods to their verification process.

More than a few studies also exist where condition/event systems (C/E) are used in PLC program verification [104]. In Hanisch et al.'s study [47], a variant of C/Es (Timed Net C/E[4]) is used, and IL of PLCs is transformed to C/E models. They use their own model checking tool in their study. Another early study by Rausch and Krogh [95] also uses Timed Net C/E and transform them to SMV models using a rule-based engine. Kowalewski et al. [66] also used C/Es together with the HyTech tool [52] to perform reachability analysis. A more recent study by Vyatkin et al. [108] also uses net C/Es and presents a framework supporting conver-

[3] Papers that do not include explicit information were omitted.

[4] Timed Net C/E's actually use Petri nets in representation of internal dynamics. Nevertheless, we will be discussing them together with other condition/event system-based approaches.

sion from state charts and model checking using SESA tool. Pang and Vyatkin [88] perform conversion from function blocks to C/Es, but did not apply model checking in their studies.

With the widespread usage of object-oriented design and UML models, there also exist some studies that use UML state charts in model checking process. Sacha [98] defined finite state time machines and use it as an intermediate format in conversion between state machine diagrams and UPPAAL models. Klotz et al. [64] also use UML state chart models and transform them to NuSMV models in verification of a case filling machine. They were able to verify basic liveness and safety properties in around 80 s for a system model with three state charts.

Wardana et al. [110] used continuous function charts (CFC), a graphical programming language widely used in process industry, and model check a state space with three million states around 50 s using UPPAAL. Mazzolini et al. [78] used MATLAB state flowcharts and perform verification of a shoe manufacturing plant model with Simulink Design Verifier in 35 s where the model contained 28 states, 34 transitions, and 21 variables. Jimenez-Fraustro and Rutten [60] also use Simulink in verifying PLC programs modelled using a data-flow language SIGNAL [71]. Weissmann et al. chose to apply model checking approach on PLC programs of industrial robot systems programmed using a special purpose VKRC language. In their paper, they translate VKRC programs into Promela models and perform model checking using SPIN [111]. They focused on deadlocks in their studies and were able to perform successful model checking to systems with around a thousand variables belonging to 10 different processes in around 2 min. The study by Anjos et al. [3] can also be mentioned where LabView-UPPAAL conversion is performed in order to model check robot controller systems. Even though a practical PLC program conversion was not implemented in the study, the authors mentioned the ease of conversion from LabView models to PLC programs in the paper.

In a few other studies, nonconventional or special design modelling languages have been used to specify PLC programs. Thapa et al. [105] used timed version of their Message-Based Part Graph (MPSG) modelling language and transformed PLC models to SMV models. Vulgarakis and Causevic [107] use REMES, a modelling language for embedded systems and extend the language for interrupt support to be used in PLC model checking with UPPAAL.

Biallas et al.'s implementation [10], called Arcade.PLC, uses an internal representation that can be obtained by transforming text-based formats: SL and IL. Their system also supports Siemens SIMATIC S7's statement list format. They also implement their own model checker operating on their internal representation format and used abstract interpretation to implement the model checker. Their model checker is capable of modelling checking past time LTL and ∀CTL specifications.

In their work [5], Barbosa et al. support a wide range of IEC61131-3 standard PLC programming languages (ST, SFC, FBD, and LD) coded in PLCOpen[5] XML format by implementing adapters that transform program organization units to the B-method's [72] specification format. They used the ProB model checker [1] to verify safety and liveness properties of the door subsystem of trains in a railway project in about 10 min.

Since most of the widely used model checkers merely exist in very early days, earliest studies use their own tool for model checking like the study by Halbwachs et. al [45] using LESAR verification tool and Moon [84] using relay ladder logic and their own model checker implementation.

As a final group of studies, we shall mention automatic test case generation and conformance testing using model checking tools and models on PLC systems. Studies by de Assis Barbosa et al. [22] and de Vasconcelos Oliveira et al. [23] use binary logic diagrams, a standard defined by instrument society of America, which is then transformed into IEC 61131-3 standards and test case generation is performed. Both of the studies use UPPAAL TRON and focus on conformance of models and PLC programs. Different from these studies, Heimdahl et al. used NuSMV to model check a flight guidance system modelled using $RSML^{-e}$ (requirements specification model). During model checking process, Heimdahl et al. provided model checker with test criterion formulations as a verification conditions. That way NuSMV creates a trace that can be used in testing purposes [50].

All of the studies discussed above provide a different approach to solve the problems they face during model checking PLC programs using conventional approaches. Reimplementing the studies listed above can be challenging; however, their way of solving the shortcomings of standards in their domains can be very influential for the researchers and practitioners working on the area.

## 9 PLC model checking in industry

Proper adoption of formal methods is an old debate subject in software industry rooted back to 1990s [46]. In case of automation and control industry, the concerns for obtaining the necessary expertise does not vanish. In their paper on formal methods in PLC programming, Frey and Litz stated in 2000 "Although being a rather intuitive discipline for a long time, industrial PLC programming will be more and more supported by formal methods" [37]. In this section, we would like to mention some applications in the industry

[5] PLCOpen XML formats for IEC61131-3 standards: http://www.plcopen.org/pages/tc6_xml/downloads/tc6_xml_v201_technical_doc.

that adopt the approaches and/or tools in the papers we have examined in the former sections.

The earliest attempts on using model checking on aviation industry were Nasa's Java PathFinder, where autonomy flight software was transformed to Promela models and checked by SPIN model checker [49]. These earlier attempts were not able to gain wider usage in avionics industry until recently.

The article by Cofer et al. [18] grounds four barriers in using formal methods in industry and especially in avionics industry. Three of these barriers are the cost of using formal analysis, building consistent models with the problem, and the use of unfamiliar notations. Cofer claims these barriers can be overcome by using model-based development. In the paper, the fourth barrier which is performance requirements of tools is being improved as the Moore's law progresses.

In a study on model checking Airbus' ground spoiler (part of an aircraft wing flaps) controller function, Lustre specification language and Luster model checker were used by Bochot et al. [11]. The study contains a detailed discussion of the model checking the outcomes of the approach; the two key problems mentioned in those outcomes were the 48-h verification time using a decent computer and the difficulties in transforming informal specifications to formal ones.

In 2009, Steven P. Miller commented about the same situation in his paper and claimed there is a growing application area for model checking by the utilization of model-based development tools like MATLAB Simulink®[6] and Esterel Technologies Scade Suite™[7] especially in avionics and automotive industries.

A recent study by Miller et al. [82] introduces the utilization of Halbwachs et al.'s approach [45] in an adaptive airline control system. Rockwell Collins and University of Minnesota have collaborated to represent Simulink models and State flowcharts in Lustre formal specification language as an intermediate format. From Lustre format, they were able to transform the specifications into the input language of three different model checkers including SMV and two different theorem provers (see Fig. 9). In this study, they were able to check around 500 properties and corrected around 100 errors this way.

The use of model checking PLC programs is not only limited to avionics industry, it has also been applied in railway control systems. A very recent study by Ferrari et al. [34] works on General Electric Transportation System's Automatic Train Protection system by transforming Simulink programs to NuSMV models. There are also other studies [31,73] in the same domain in industrial-sized systems; however, their primary focus is not PLCs.

[6] The Mathworks, Simulink Product Description: http://www.mathworks.com/help/simulink/gs/product-description.html.

[7] Esterel Technologies, SCADE Suite Product Description: http://www.esterel-technologies.com/products/scade-suite/.

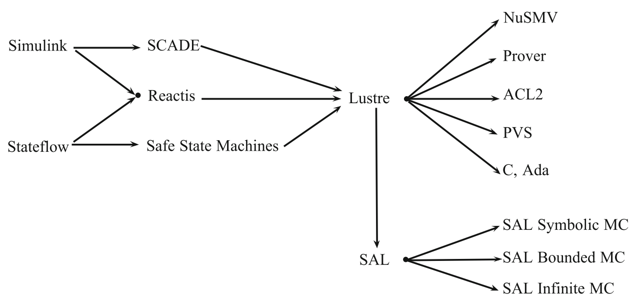


**Fig. 9** Model checking by model-based development as implemented in [82]

**Table 13** Number of studies related to different programming languages in 5-year periods

| PLC prog. | Textual | LD | SFC | FBD | Petri net | Other |
|---|---|---|---|---|---|---|
| –1999 | 1 | 3 | 1 | 0 | 1 | 5 |
| 2000–2004 | 4 | 3 | 10 | 0 | 3 | 3 |
| 2005–2009 | 4 | 3 | 0 | 3 | 2 | 8 |
| 2010– | 0 | 2 | 0 | 5 | 2 | 3 |

In addition to model checking, the application of formal methods in industrial systems is a very broad topic, which needs the inclusion of various different aspects of formal methods and also widely applied techniques like discrete event system approach [15,21]. Surveys that focus on industrial areas rather than PLC programs can also be found in the literature [32,68].

In this section, we have tried to summarize recent advances in industry that transform PLC programs and utilize model checking directly on them. It can be seen that in industry current trend in the application of model checking PLC programs is using model-based development tools on symbolic model checkers like NuSMV.

## 10 Open problems and research challenges

Examining the historical development of the PLC model checking studies, we can see in Table 13 that the FBD model checking area contains the majority of the papers published in the last 5 years. Model checking textual PLC programs seem to saturate during 2000s, although LDs being almost the same age as textual programs seem to have a constant pace of publications since they can be easily converted to propositional logic representations. The ability to use Petri nets at a more abstract level even before the PLC program is produced makes them an area that is being studied at a constant pace starting from 2000s. SFC-based PLC model checking has its golden era in the beginning of 2000s before being replaced by FBDs, which are being widely used in industry today. Nonstandard programming languages are also being constantly studied in order to overcome some difficulties like modelling PLC-specific programming constructs (e.g., TONs) or to bridge the gap between widely used modelling tools (e.g., UML and LabView).

After examining the model checking studies performed in the area of PLC software verification, we shall present some important common practices that were frequently applied in the studies. After discussion of common practices, we are going to present some open challenges and key points that should be considered during conducting research in the area.

### 10.1 Common challenges

The challenges that were faced during specific PLC programming areas are explained in the related sections, but the common challenges that are faced during the studies in the survey can be summarized as follows:

- *State Space Explosion* Especially textual programs possess a more granular structure, and further abstractions are needed to be applied during the automatic translation process or after the translation process manually. These abstractions aim to gather equivalent states and remove the unnecessary ones to shrink the state space by automatic conversion of the textual program. Many different techniques are applied in order to overcome state space explosion including applying further abstractions like compacting recurring place clusters inside a Petri net into a single place.
  As mentioned, these further abstractions can be done automatically by applying a second step of automatic process during the conversion. This is often done by directly (without any abstraction) converting source code to an intermediate format using a rule-based system. Successively, further abstractions are applied over the intermediate model and another conversion is performed to produce input to the target model checking tool.
  These abstractions are performed manually in some studies, but these situations negatively affected the automation of the process. Another point where manual intervention applied is over the source of PLC program. A subset of the source language instruction set is selected in almost all of the studies, and additionally, in most of the studies, only the boolean variables are selected to be included in the translation process. A more effort-demanding solution is applied by developing specialized model checkers for the specific case.
- *Model Consistency* The usual problem in model checking is to ensure that the built model to be checked is correct and the model represents the system consistently. For the PLC programming domain, the automatic conversion is easier compared to conventional programming, because conventionally PLC program development life cycle already starts with state-based representations like automata or Petri nets. PLC programs using mostly primitive data types and boolean variables make the process even easier. The hard part is to model the timing constructs and reflect the real-time nature of the PLC programs. The most common solution to this problem is to apply divide and conquer strategy and reuse manually converted and heavily tested timed components of PLC programs in automating the conversion process.
- *Specifying Properties to be Checked* A problem almost as hard as building a correct model is to correctly specify the properties to be checked using the model. Due to its low-level nature and development life cycle, PLC programs can be more easily represented with state transition systems. However, when it comes to property specification, model checkers require temporal logic to be used as medium that requires expertise in formal methods and mathematical modelling area. As technology progresses, using natural language processing or conversion from a tabular format to extract specifications in LTL or CTL is expected to become more powerful by the industry. Currently, it seems to be one of the most appealing research directions as well.
- *Representing PLC execution cycle* A common challenge that was faced in most of the studies above is reflecting the PLC execution cycle to the model checking environment in a convenient way. To implement a complete system, researches have chosen to model distinct phases of the PLC execution cycle as separate modules for the model checker. Moreover, a PLC cycle sequencer module is also included in the model often to mimic the timing properties between sequences of PLC cycles. In addition to those phases, modules to model interrupts and triggers are also included in the model in some studies.
- *Modelling TONs* Another very common challenge is the modelling of TONs (timer on-delay). TONs are used to enable an output of the PLC for a period of time when an input of TON receives true input. TONs are important for PLC programs, because they are excessively used to ensure timing properties of PLCs explicitly inside the program. A frequently applied solution in handling TONs is using a Timed Automata and a model checker that support real-time model checking like UPPAAL.

### 10.2 Future research directions

Apart from the mostly discussed challenges and proposed solutions above, there are also less frequently discussed solutions, which can be a good point to direct future research. One of those problems is the conformance of generated models with the original program. There are studies that directly discuss this kind of problems and also some studies that propose a solution together with their model checking approach. These solutions usually involve conformance testing of the model used, and a model is directly generated from the PLC program. Generating the PLC program and the model to be verified from a common ancestor model is another solution to this problem, but most of the studies still do not present sound discussions about the problem or the consistency concerns raised by this approach.

Another common drawback is the lack of performance considerations about the solution presented in the papers. Most of the time authors point to the number of defects found by their approach, but in our humble opinion this information becomes subjective if the data about size of the models being checked or the performance numbers are not present in the discussion. A serious discussion of the mentioned information about the study is needed to reason about practical aspects of the proposed approach.

Although still being tightly dependent to the improvements of model checking performance, there exists a small number of studies in model checking networked PLCs or multitasking PLCs. Even though it can be hard to overcome the state space explosion problem in such purposes, it can be still interesting to push the limits by applying abstractions in this area.

We would also like to mention two more interesting commonalities in the discussed studies. From the current perspective, it can be seen that SMV and UPPAAL dominate the model checking practices on PLC programs. There can be many reasons behind this, but we believe the most effective two are tool support/ease of use and simplistic syntax that can be more easily translated from a PLC program. It is remarkable that SPIN, which is another popular model checker, is used in significantly fewer studies. Another interesting fact is that none of the discussed Petri net studies were applied on the area of railway control systems even though they are known to be applied in such systems frequently.

Most of the present challenges and future studies presented in this section is related to state space explosion problem. This situation is not surprising especially for model checking practitioners and puts the emphasis on state space reduction, being the usual suspect whenever model checking is applied in verification purposes.

Lastly, we would also like to discuss the future challenges introduced by the development of Web technologies and cloud infrastructure. Latest achievements in cloud computing and Internet of Things area awaken a demand in industrial automation on gathering and integrating information from data sources to enterprise software systems via PLCs. Many mainstream PLC hardware/software manufacturers released Web-based versions of their PLC visualization and programming software like Siemens,[8] Beckhoff[9] and other software manufacturers.[10,11,12,13] PLC hardware manufacturers even make it possible to access the PLC software by Web servers that runs on the PLC itself. Together with the integration issues, becoming more accessible brings security constraints into the automation control area increasing the need for the verification of security requirements in PLC control and communication [65,70]. This situation unveils a new potential to verification studies focusing on software and protocol security like Murphi [28] and AVISPA [4] in the next decade in PLC software verification.

## 11 Conclusion

Verification of PLC programs is a very widely studied area of research, and applying model checking in such purposes contains numerous studies proposing the integration of a number of model checking tools over a variety of PLC programming and modelling methods. In this paper, we present an overview of these methods by classifying them according to the IEC 61131 standards. Additionally, we present additional sections regarding Petri nets as being almost the most widely used modelling approach when it comes to PLC programming as well as various different nonconventional techniques proposed in the area.

During our classifications, we followed a practical approach and provided the prominent properties of the studies and the relations between these properties assuming the readers intention is to get an idea about the appropriate approach according to the area, size of the system, and the type of the programming language she is planning to use in the future. We believe that our discussions are not only capable of directing the practitioners aiming to integrate model checking approaches in their software production processes but also provides an overview of the state-of-art research and open problems for the researchers interested in the area.

## References

1. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press, Cambridge (2005)
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
3. Anjos, J.M.S., Coracini, G.K., Villani, E.: A proposal and verification of a software architecture based on labview for a multifunc-

[8] Siemens WinCC/Web Navigator: http://w3.siemens.com/mcms/human-machine-interface/en/visualization-software/scada/wincc-options/wincc-web-navigator/Pages/Default.aspx.

[9] Beckhoff TwinCAT PLC HMI Web: http://www.beckhoff.com/english.asp?twincat/twincat_plc_hmi_web.htm.

[10] Atvise Scada: http://www.atvise.com/en/products-solutions/atvise-scada.

[11] Indusoft Cloud Computing for Scada: http://www.indusoft.com/Documentation/White-Papers/ArtMID/1198/ArticleID/430/Cloud-Computing-for-SCADA.

[12] Xio Cloud Scada Control System: http://www.xioio.com/wp/?page_id=92.

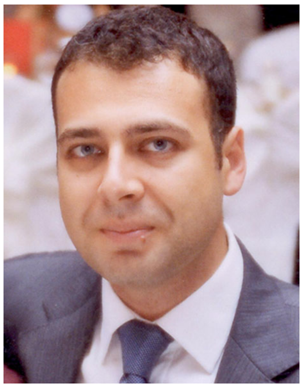
[13] PLCCloud: https://plccloud.pro/.

tional robotic end-effector. Adv. Eng. Softw. **55**, 32–44 (2013). doi:10.1016/j.advengsoft.2012.09.004
4. Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuéllar, J., Drielsma, P.H., Héam, P.C., Kouchnarenko, O., Mantovani, J., et al.: The avispa tool for the automated validation of internet security protocols and applications. In: Etessami, K., Rajamani, S.K. (eds.) Computer Aided Verification, pp. 281–285. Springer, Berlin (2005)
5. Barbosa, H., Déharbe, D.: Formal verification of plc programs using the b method. In: Abstract State Machines, Alloy, B, VDM, and Z, pp. 353–356. Springer, Berlin (2012)
6. Bauer, N., Huuck, R.: Towards automatic verification of embedded control software. In: Proceedings of Second Asia-Pacific Conference on Quality Software, pp. 375–383 (2001). doi:10.1109/APAQS.2001.990043
7. Bauer, N., Engell, S., Huuck, R., Lohmann, S., Lukoschus, B., Remelhe, M., Stursberg, O.: Verification of plc programs given as sequential function charts. In: Integration of Software Specification Techniques for Applications in Engineering, Lecture Notes in Computer Science, vol. 3147, chap. 28, pp. 517–540. Springer, Berlin (2004) DOI:10.1007/978-3-540-27863-4_28
8. Bender, D.F., Combemale, B., Crgut, X., Farines, J.M., Berthomieu, B., Vernadat, F.: Ladder metamodeling and plc program validation through time petri nets. In: Schieferdecker, I., Hartman, A. (eds.) Model Driven Architecture Foundations and Applications, Lecture Notes in Computer Science, vol. 5095, pp. 121–136. Springer, Berlin (2008)
9. Berthomieu, B., Ribet, P.O., Vernadat, F.: The tool tina-construction of abstract state spaces for petri nets and time petri nets. Int. J. Prod. Res. **42**(14), 2741–2756 (2004)
10. Biallas, S., Brauer, J., Kowalewski, S.: Arcade.plc: a verification platform for programmable logic controllers. In: 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 338–341 (2012). doi:10.1145/2351676.2351741
11. Bochot, T., Virelizier, P., Waeselynck, H., Wiels, V.: Model checking flight control systems: the airbus experience. ICSE Companion, pp. 18–27 (2009)
12. Bornot, S., Huuck, R., Lukoschus, B., Lakhnech, Y.: Verification of sequential function charts using smv. In: In PDPTA 2000: International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, pp. 2987–2993 (2000)
13. Brayton, R.K., Hachtel, G.D., Sangiovanni-Vincentelli, A., Somenzi, F., Aziz, A., Cheng, S.T., Edwards, S., Khatri, S., Kukimoto, Y., Pardo, A.: Vis: a system for verification and synthesis. In: Alur, R., Henzinger, T.A. (eds.) Computer Aided Verification, pp. 428–432. Springer, Berlin (1996)
14. Brinksma, E., Mader, A.: Verification and optimization of a plc control schedule. In: Havelund, K., Penix, J., Visser, W. (eds.) SPIN Model Checking and Software Verification, pp. 73–92. Springer, Berlin (2000)
15. Budha, M., Thapa, D., Park, S., Wang, G.N.: Generation of plc ladder diagram using modular structure. In: 2008 International Conference on Computational Intelligence for Modelling Control Automation, pp. 1194–1198 (2008). doi:10.1109/CIMCA.2008.125
16. Canet, G., Couffin, S., Lesage, J.J., Petit, A., Schnoebelen, P.: Towards the automatic verification of plc programs written in instruction list. In: IEEE International Conference on Systems, Man, and Cybernetics, vol. 4, pp. 2449–2454. IEEE (2000)
17. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: Nusmv: a new symbolic model verifier. In: Computer Aided Verification. Springer, Berlin, pp 495–499 (1999)
18. Cofer, D.D., Whalen, M.W., Miller, S.P.: Model-checking of safety-critical software for avionics. ERCIM News 75 (2008)
19. Couffin, S.L., Lesage, J.J.: Formal verification of the sequential part of plc programs. In: Boel, R., Stremersch, G. (eds) Discrete Event Systems. Springer, Berlin, pp. 247–254 (2000)
20. da Silva, L.D., de Assis Barbosa, L.P., Gorgonio, K., Perkusich, A., Lima, A.M.N.: On the automatic generation of timed automata models from function block diagrams for safety instrumented systems. In: 34th Annual Conference of IEEE Industrial Electronics, IECON 2008, pp. 291–296 (2008). doi:10.1109/IECON.2008.4757968
21. Dacharry, H.P., Giambiasi, N.: A formal verification approach for devs. In: Proceedings of the 2007 Summer Computer Simulation Conference, Society for Computer Simulation International, SCSC '07, San Diego, CA, USA, pp. 312–319 (2007)
22. de Assis Barbosa, L.P., Gorgonio, K., da Silva, L.D., Lima, A.M.N., Perkusich, A.: On the automatic generation of timed automata models from isa 5.2 diagrams. In: IEEE Conference on Emerging Technologies and Factory Automation, 2007. ETFA, IEEE, pp. 406–412 (2007)
23. de Vasconcelos Oliveira, K., da Silva, L.D., Perkusich, A., Lima, A.M.N., Gorgônio, K.: Automatic timed automata extraction from ladder programs for model-based analysis of control systems. In: IEEE International Symposium on Industrial Electronics (ISIE), pp. 90–95. IEEE (2010)
24. Dierks, H.: Plc-automata: a new class of implementable real-time automata. In: Bertran, M., Rus, T. (eds.) Transformation-Based Reactive Systems Development, pp. 111–125. Springer, Berlin (1997)
25. Dierks, H.: PLC-automata: a new class of implementable real-time automata. Theor. Comput. Sci. **253**(1), 61–93 (2001)
26. Dierks, H.: Comparing model checking and logical reasoning for real-time systems. Form. Asp. Comput. **16**(2), 104–120 (2004)
27. Dierks, H., Tapken, J.: Tool-supported hierarchical design of distributed real-time systems. In: Proceedings of 10th Euromicro Workshop on Real-Time Systems, pp. 222–229. IEEE (1998)
28. Dill, D.L.: The murphi verification system. In: Proceedings of the 8th International Conference on Computer Aided Verification, CAV '96, pp. 390–393. Springer, London (1996)
29. Enoiu, E.P., Doganay, K., Bohlin, M., Sundmark, D., Pettersson, P.: Mos: an integrated model-based and search-based testing tool for function block diagrams. In: 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE), pp. 55–60 (2013a). doi:10.1109/CMSBSE.2013.6605711
30. Enoiu, E.P., Sundmark, D., Pettersson, P.: Model-based test suite generation for function block diagrams using the uppaal model checker. In: Proceedings of Sixth IEEE International Conference on Software Testing, Verification and Validation. IEEE (2013b)
31. Faivre, A., Benoit, P.: Safety critical software of meteor developed with the B formal method and the vital coded processor. In: WCRR'99, World Congress on Railway Research, Tokyo, Japan (1999)
32. Fantechi, A., Gnesi, S.: On the adoption of model checking in safety-related software industry. In: Flammini, F., Bologna, S., Vittorini, V. (eds.) Computer Safety, Reliability, and Security, pp. 383–396. Springer, Berlin (2011)
33. Farines, J.M., de Queiroz, M.H., da Rocha, V.G., Carpes, A.M.M.: A model-driven engineering approach to formal verification of plc programs. In: IEEE 16th Conference on Emerging Technologies Factory Automation (ETFA), pp. 1–8 (2011). doi:10.1109/ETFA.2011.6058983
34. Ferrari, A., Fantechi, A., Magnani, G., Grasso, D., Tempestini, M.: The metrrio case study. Sci. Comput. Program. **78**(7), 828–842 (2013). doi:10.1016/j.scico.2012.04.003
35. Frey, G.: Hierarchical design of logic controllers using signal interpreted petri nets. In: Proceedings of the IFAC Conference on Analysis and Design of Hybrid Systems, p. 12 (2003)

36. Frey, G., Litz, L.: Verification and validation of control algorithms by coupling of interpreted petri nets. In: 1998 IEEE International Conference on Systems, Man, and Cybernetics. vol 1, pp. 7–12. IEEE (1998)
37. Frey, G., Litz, L.: Formal methods in plc programming. In: IEEE International Conference on Systems, Man, and Cybernetics, vol. 4, pp. 2431–2436. IEEE (2000)
38. Frey, G., Wagner, F.: A toolbox for the development of logic controllers using petri nets. In: 8th International Workshop on Discrete Event Systems, pp. 473–474. IEEE (2006)
39. Fujino, K., Imafuku, K., Yuh, Y., Hirokazu, N.: Design and verification of the sfc program for sequential control. Comput. Chem. Eng. **24**(2), 303–308 (2000)
40. Gergely, E.I., Coroiu, L., Gacsadi, A.: Design of safe plc programs by using petri nets and formal methods. In: 11th WSEAS International Conference on Automation and Information, Romania, pp. 86–91 (2010)
41. Gourcuff, V., Smet, O.D., Faure, J.M.: Efficient representation for formal verification of plc programs. In: 8th International Workshop on Discrete Event Systems, pp. 182–187. IEEE (2006)
42. Grobelna, I.: Formal verification of embedded logic controller specification with computer deduction in temporal logic. Electr. Rev. **12a**, 47–50 (2011)
43. Grobelna, I.: Control interpreted petri nets-model checking and synthesis. In: Pawlewski, P. (ed.) Petri Nets—Manufacturing and Computer Science. InTech, Rijeka (2012). doi:10.5772/47797
44. Grobelna, I., Adamski, M.: Model checking of control interpreted petri nets. In: Proceedings of the 18th International Conference Mixed Design of Integrated Circuits and Systems (MIXDES), pp. 621–626. IEEE (2011)
45. Halbwachs, N., Lagnier, F., Ratel, C.: Programming and verifying real-time systems by means of the synchronous data-flow language lustre. IEEE Trans. Softw. Eng. **18**(9), 785–793 (1992)
46. Hall, A.: Seven myths of formal methods. Softw. IEEE **7**(5), 11–19 (1990)
47. Hanisch, H.M., Thieme, J., Luder, A., Wienhold, O.: Modeling of plc behavior by means of timed net condition/event systems. In: 1997 6th International Conference on Emerging Technologies and Factory Automation Proceedings, ETFA'97, pp. 391–396. IEEE (1997)
48. Harel, D.: Statecharts: a visual formalism for complex systems. Sci. Comput. Program. **8**(3), 231–274 (1987). doi:10.1016/0167-6423(87)90035-9
49. Havelund, K., Lowry, M., Park, S., Pecheur, C., Penix, J., Visser, W., White, J., et al.: Formal analysis of the remote agent before and after flight. In: Proceedings of the 5th NASA Langley Formal Methods Workshop, vol. 134 (2000)
50. Heimdahl, M.P., Rayadurgam, S., Visser, W., Devaraj, G., Gao, J.: Auto-generating test sequences using model checkers: a case study. In: Formal Approaches to Software Testing, pp. 42–59. Springer, Berlin (2004)
51. Heiner, M., Menzel, T.: A petri net semantics for the plc language instruction list. In: Workshop on Discrete Event Systems (WODES 98), pp. 161–166 (1998)
52. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: A user guide to hytech. In: Brinksma, E., Cleaveland, W.R., Larsen, K.G., Margaria, T., Steffen, B. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, pp. 41–71. Springer, Berlin (1995)
53. Holzmann, G.J.: The model checker spin. IEEE Trans. Softw. Eng. **23**(5), 279–295 (1997)
54. Huuck, R.: Software verification for programmable logic controllers. Ph.D. thesis, University of Kiel (2003)
55. Huuck, R., Lukoschus, B., Bauer, N.: A model-checking approach to safe sfcs. In: IMACS Multiconference on Computational Engineering in Systems Applications (2003)
56. James, P., Lawrence, A., Moller, F., Roggenbach, M., Seisenberger, M., Setzer, A., Kanso, K., Chadwick, S.: Verification of solid state interlocking programs. In: Counsell, S., Nez, M. (eds.) Software Engineering and Formal Methods, Lecture Notes in Computer Science, pp. 253–268. Springer, Berlin (2014). doi:10.1007/978-3-319-05032-4-19
57. Jee, E., Jeon, S., Cha, S.D., Koh, K.Y., Yoo, J., Park, G.Y., Seong, P.H.: Fbdverifier: interactive and visual analysis of counterexample in formal verification of function block diagram. J. Res. Pract. Inf. Technol. **42**(3), 171–188 (2010)
58. Jensen, K.: Coloured Petri Nets. Springer, Berlin (1987)
59. Jeon, S.: Verification of function block diagram through Verilog translation. Master's thesis, KAIST, Republic of Korea (2007)
60. Jiménez-Fraustro, F., Rutten, É.: A synchronous model of iec 61131 plc languages in signal. In: 13th Euromicro Conference on Real-Time Systems, pp. 135–142. IEEE (2001)
61. John, K.H., Tiegelkamp, M.: IEC 61131–3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids. Springer, Berlin (2010)
62. Jouault, F., Kurtev, I.: Transforming models with atl. In: Satellite Events at the MoDELS 2005 Conference, pp. 128–138. Springer, Berlin (2006)
63. Klein, S., Weng, X., Frey, G., Lesage, J.J., Litz, L.: Controller design for an fms using signal interpreted petri nets and sfc: validation of both descriptions via model-checking. In: Proceedings of the 2002 American Control Conference, vol. 5, pp. 4141–4146. IEEE (2002)
64. Klotz, T., Fordran, E., Straube, B., Haufe, J.: Formal verification of uml-modeled machine controls. In: IEEE Conference on Emerging Technologies and Factory Automation, ETFA 2009, pp. 1–7. IEEE (2009)
65. Kornecki, A.J., Zalewski, J.: Safety and security in industrial control. In: Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research, CSIIRW '10, pp 77:1–77:4. ACM, New York (2010). doi:10.1145/1852666.1852754
66. Kowalewski, S., Engell, S., Preußig, J., Stursberg, O.: Verification of logic controllers for continuous plants using timed condition/event-system models. Automatica **35**(3), 505–518 (1999)
67. Lahtinen, J.: Model checking timed safety instrumented systems. Technical report TKK-ICS-R3. Department of Computer Science, Michigan State University (2008)
68. Lahtinen, J., Valkonen, J., Bjrkman, K., Frits, J., Niemel, I., Heljanko, K.: Model checking of safety-critical software in the nuclear engineering domain. Reliab. Eng. Syst. Saf. **105**, 104–113 (2010). doi:10.1016/j.ress.2012.03.021, ESREL
69. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. Int. J. Softw. Tools Technol. Transf. (STTT) **1**(1), 134–152 (1997)
70. Lawton, G.: Industrial control systems face more security challenges. http://www.computer.org/portal/web/computingnow/news/industrial-control-systems-face-more-security-challenges/ (2011). Accessed 20 July 2014
71. LeGuernic, P., Gautier, T., Borgne, M.L., Maire, C.L.: Programming real-time applications with signal. Proc. IEEE **79**(9), 1321–1336 (1991)
72. Leuschel, M., Butler, M..: Prob: a model checker for b. In: FME 2003: Formal Methods, pp. 855–874. Springer (2003)
73. Leuschel, M., Falampin, J., Fritz, F., Plagge, D.: Automated property verification for large scale b models with prob. Form. Asp. Comput. **23**(6), 683–709 (2011). doi:10.1007/s00165-010-0172-1
74. Lewis, R.R.W.: Programming industrial control systems using IEC 1131-3. 50, IET (1998)

75. L'Her, D., Parc, P.L., Marcé, L.: Proving sequential function chart programs using automata. In: Automata Implementation, pp. 149–163. Springer (1999)
76. Mader, A., Wupper, H.: Timed automaton models for simple programmable logic controllers. In: Proceedings of the 11th Euromicro Conference on Real-Time Systems, pp. 106–113. IEEE (1999)
77. Mader, A., Brinksma, E., Wupper, H., Bauer, N.: Design of a plc control program for a batch plant vhs case study 1. Eur. J. Control **7**(4), 416–439 (2001)
78. Mazzolini, M., Brusaferri, A., Carpanzano, E.: Model-checking based verification approach for advanced industrial automation solutions. In: IEEE Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1–8. IEEE (2010). doi:10.1109/ETFA.2010.5641209
79. McMillan, K.L.: Symbolic Model Checking. Springer, Berlin (1993)
80. McMillan, K.L.: The SMV Language. Cadence Berkeley Labs, Berkeley (1999)
81. Mertke, T., Frey, G.: Formal verification of plc programs generated from signal interpreted petri nets. In: IEEE International Conference on Systems, Man, and Cybernetics, vol. 4, pp. 2700–2705. IEEE (2001)
82. Miller, S.P., Whalen, M.W., Cofer, D.D.: Software model checking takes off. Commun. ACM **53**(2), 58–64 (2010)
83. Mokadem, H.B., Berard, B., Gourcuff, V., Smet, O.D., Roussel, J.M.: Verification of a timed multitask system with uppaal. IEEE Trans. Autom. Sci. Eng. **7**(4), 921–932 (2010). doi:10.1109/TASE.2010.2050199
84. Moon, I.: Modeling programmable logic controllers for logic verification. IEEE Control Syst. **14**(2), 53–59 (1994)
85. Németh, E., Bartha, T.: Formal verification of safety functions by reinterpretation of functional block based specifications. In: Cofer, D., Fantechi, A. (eds.) Formal Methods for Industrial Critical Systems, pp. 199–214. Springer (2009)
86. Olderog, E.R.: Correct real-time software for programmable logic controllers. In: Olderog, E-R., Steffen, B. (eds.) Correct System Design, pp. 342–362. Springer, Berlin (1999)
87. Pakonen, A., Mtsniemi, T., Lahtinen, J., Karhela, T.: A toolset for model checking of plc software. In: Proceedings of 18th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA2013 (2013)
88. Pang, C., Vyatkin, V.: Automatic model generation of iec 61499 function block using net condition/event systems. In: 6th IEEE International Conference on Industrial Informatics, 2008. INDIN 2008, IEEE, pp 1133–1138 (2008)
89. Pavlovic, O., Ehrich, H.D.: Model checking plc software written in function block diagram. In: Third International Conference on Software Testing, Verification and Validation (ICST), pp. 439–448 (2010). doi:10.1109/ICST.2010.10
90. Pavlovic, O., Pinger, R., Kollmann, M.: Automated formal verification of PLC programs written in IL. In: 4th International Verification Workshop, Bremen, Germany (2007)
91. Peleska, J., Haxthausen, A.E.: Object code verification for safety-critical railway control systems. In: Proceedings of 6th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 2007), pp. 184–199 (2007)
92. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice Hall PTR, Upper Saddle River (1981)
93. Pnueli, A.: The temporal logic of programs. In: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, SFCS '77. IEEE Computer Society, Washington, pp. 46–57 (1977). doi:10.1109/SFCS.1977.32
94. Probst, S.T., Powers, G.J., Long, D., Moon, I.: Verification of a logically controlled, solids transport system using symbolic model checking. Comput. Chem. Eng. **21**(4), 417–429 (1997)
95. Rausch, M., Krogh, B.H.: Formal verification of plc programs. In: Proceedings of the 1998 American Control Conference, vol. 1, pp. 234–238. IEEE (1998)
96. Rossi, O., Schnoebelen, P.: Formal modeling of timed function blocks for the automatic verification of ladder diagram programs. In: Proceedings of 4th International Conference on Automation of Mixed Processes: Hybrid Dynamic Systems (ADPM2000), Dortmund, Germany, pp. 177–182 (2000)
97. Rumbaugh, J., Jacobson, I., Booch, G.: Unified Modeling Language Reference Manual. The Pearson Higher Education, Upper Saddle River (2004)
98. Sacha, K.: Verification and implementation of dependable controllers. In: Third International Conference on Dependability of Computer Systems, 2008. DepCos-RELCOMEX'08, pp 143–151. IEEE (2008)
99. Sarmento, C.A., Silva, J.R., Miyagi, P.E., Filho, D.J.S.: Modeling of programs and its verification for programmable logic controllers. In: Proceedings of IFAC 17th World Congress, pp. 10546–10551 (2008)
100. Schlich, B., Brauer, J., Wernerus, J., Kowalewski, S.: Direct model checking of plc programs in IL. Dependable Control of Discrete Syst. **2**, 28–33 (2009)
101. Smet, O.D., Rossi, O.: Verification of a controller for a flexible manufacturing line written in ladder diagram via model-checking. In: Proceedings of the 2002 American Control Conference, 2002, vol. 5, pp. 4147–4152 (2002). doi:10.1109/ACC.2002.1024580
102. Smet, O.D., Couffin, S., Rossi, O., Canet, G., Lesage, J., Schnoebelen, P., Papini, H.: Safe programming of plc using formal verification methods. In: Proceedings of 4th International PLCopen Conference on Industrial Control Programming (ICP'2000), Utrecht, The Netherlands, pp. 73–78 (2000)
103. Soliman, D., Frey, G.: Verification and validation of safety applications based on PLCopen safety function blocks. Control Eng. Pract. **19**(9), 929–946 (2011). doi:10.1016/j.conengprac.2011.01.001. special Section: DCDS09 The 2nd IFAC Workshop on Dependable Control of Discrete Systems
104. Sreenivas, R.S., Krogh, B.H.: On condition/event systems with discrete state realizations. Discrete Event Dyn. Syst. **1**(2), 209–236 (1991)
105. Thapa, D., Park, J., Wang, G.N., Shin, D.: Timed-mpsg: a formal model for real-time shop floor controller. In: International Conference on Computational Intelligence for Modelling, Control and Automation, and International Conference on Intelligent Agents, pp. 101–101. Web Technologies and Internet Commerce. IEEE (2006)
106. Turk, A.L., Probst, S.T., Powers, G.J.: Verification of real time chemical processing systems. In: Maler, O. (ed.) Hybrid and Real-Time Systems, pp. 259–272. Springer, Berlin (1997)
107. Vulgarakis, A., Causevic, A.: Applying remes behavioral modeling to plc systems. In: XXII International Symposium on Information, Communication and Automation Technologies, ICAT 2009, pp 1–8. IEEE (2009)
108. Vyatkin, V., Hanisch, H.M., Pfeiffer, T.: Object-oriented modular place/transition formalism for systematic modeling and validation of industrial automation systems. In: Proceedings of IEEE International Conference on Industrial Informatics, INDIN 2003, pp. 224–232. IEEE (2003)
109. Wang, R., Song, X., Gu, M.: Modelling and verification of program logic controllers using timed automata. IET Softw. **1**(4), 127–131 (2007)
110. Wardana, A., Folmer, J., Vogel-Heuser, B.: Automatic program verification of continuous function chart based on model checking. In: 35th Annual Conference of IEEE Industrial Electronics, IECON'09, pp. 2422–2427. IEEE (2009)
111. Weißmann, M., Bedenk, S., Buckl, C., Knoll, A.: Model checking industrial robot systems. In: Groce, A., Musuvathi, M.

(eds.) Model Checking Software, pp. 161–176. Springer, Berlin (2011)
112. Weng, X., Litz, L.: Model checking of signal interpreted petri nets. In: IEEE International Conference on Systems, Man, and Cybernetics, vol. 4, pp. 2748–2752. IEEE (2001)
113. Willems, H.: Compact timed automata for plc programs. Technical report CSI-R9925, University of Nijmegen, The Netherlands (1999)
114. Witsch, D., Vogel-Heuser, B., Faure, J.M., Marsal, G.: Performance analysis of industrial ethernet networks by means of timed model-checking. In: Proceedings of the 12th IFAC Symposium on Information Control Problems in Manufacturing, INCOM 2006, Saint-Etienne, France (2006)
115. Yoo, J., Cha, S., Jee, E.: A verification framework for fbd based software in nuclear power plants. In: 15th Asia-Pacific Software Engineering Conference, APSEC '08, pp. 385–392 (2008). doi:10.1109/APSEC.2008.26
116. Younis, M.B., Frey, G.: Formalization of existing plc programs: a survey. In: Proceedings of CESA, pp. 0234–0239 (2003)
117. Yovine, S.: Kronos: a verification tool for real-time systems. Int. J. Softw. Tools Technol. Transf. (STTT) **1**(1), 123–133 (1997)
118. Zhou, M., He, F., Gu, M., Song, X. Translation-based model checking for plc programs. In: 33rd Annual IEEE International Computer Software and Applications Conference, COMPSAC '09, vol. 1, pp. 553–562 (2009). doi:10.1109/COMPSAC.2009.80
119. Zoubek, B., Roussel, J.M., Kwiatkowska, M.: Towards automatic verification of ladder logic programs. In: Proceedings of IMACS-IEEE'CESA'03': Computational Engineering in Systems Applications (2003)



**Tolga Ovatman** is working as an Assistant Professor in the Computer Engineering Department of Istanbul Technical University (İTÜ), Turkey. He received his B.Sc degree in computer engineering from Hacettepe University, Turkey, in 1999 and his M.Sc and Ph.D. degrees in computer engineering from İTÜ in 2005 and 2011, respectively. He participated the National Railway Signalization Project of Turkey between 2009 and 2011 and contributed in testing and verification activities of PLC software. His research interests include Model Checking, Object-Oriented Software Analysis and Design, and Parallel Programming for CMP.

**Atakan Aral** received the B.Sc degree from Istanbul Technical University, Turkey (2009), and the M.Sc degree from Politecnico di Milano, Italy (2011), both in Computer Engineering. He is currently a Ph.D. candidate and Research/Teaching assistant at Istanbul Technical University, Faculty of Computer and Informatics. His primary research interests include Software Modeling/Analysis and Cloud Computing.

**Davut Polat** received his B.A in Computer Engineering from Istanbul Technical University (İTÜ) in 2011, he is still graduate student at İTÜ. He is currently working for a company delivering software solutions mainly on PLC-based data collection in industrial automations. His research interests span the field of model checking, machine learning, and digital design.

**Ali Osman Ünver** graduated from Mathematics Engineering Department at Istanbul Technical University in 2012 and currently continues his M.Sc education at the same university in Department of Defense Technologies. His research interests include applied mathematical modeling and model checking.