Formalization of existing PLC Programs: A Survey

M. Bani Younis and G. Frey
Juniorprofessorship Agentenbased Automation
Kaiserslautern University of Technology
D-67653 Kaiserslautern, Germany
E-Mail: {frey, baniy}@eit.uni-kl.de

Abstract: In recent years, the interest in the formalization of PLC programs increased. The paper provides a classification scheme for the works done in this field. This scheme includes the sources used for formalization, the level of the formalization process (i.e. the complexity of structures that could be handled by the approach), the aim of the formalization (Re-Engineering or Verification) and the formal model used to represent the formalized PLC program. The scheme is applied to several examples.

I. Introduction

Programmable Logic Controllers (PLCs) are a special type of computers that are used in industrial and safety-critical applications. The purpose of a PLC is to control a particular process, or a collection of processes, by producing electrical control signals in response to electrical process-related inputs signals.

The systems controlled by PLCs vary tremendously, with applications in manufacturing, chemical process control, machining, transportation, power distribution, and many other fields. Automation applications can range in complexity from a simple panel i.e. to operate the lights in a conference room to completely automated production systems like e.g. a brewery in which the machinery for everything – from dispensing and mixing ingredients to controlling the brewing process and even filling and sealing of bottles – is under PLC control.

There are two main reasons for the formalization of PLC programs (cf. Fig. 1). The first is the need for formal Verification and Validation, Simulation and Analysis of existing systems due to increased awareness of safety and quality. The second is the constant progress involved in production and its automation. This means that existing programs have to be changed in order to meet new production demands or have to be transferred to new controller hardware (re-implementation). Since in most cases there is no formal description of the programs available that could be used for these tasks, this description has to be generated from the existing code. The only other solution would be to do a completely new implementation.

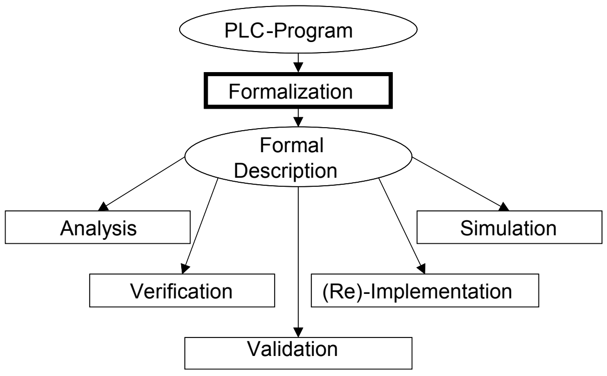

Figure 1: Reasons for Formalization of PLC Programs.

The process of *formalization* of already implemented PLC code is also known as *reinterpretation* [1] or as *translation of the PLC program form one source to another* (formal language or programming language) [2].

Due to the broadness of the field with practitioners and academics from different specialized areas working in it, this Survey cannot be complete in any sense. However considerable effort was made to give a concise overview. The paper provides a classification scheme for the works done in formalization of PLC programs and applies this scheme to several examples.

The paper is structured as follows: in Section II the classification scheme is introduced. The four sections following (III to VI) discuss the four classification criteria in detail. Section VII presents a list of examples and classifies them according to the presented criteria.

II. Classification

Research on the formalization of PLC programs can be classified according to four main criteria:

- The sources used for the formalization, which are
  - the language the PLC program is written in
  - and – if applicable – the additional information needed for the formalization.
- The level of the formalization of the PLC based on the complexity of structures that the formalization process could handle (simple algorithm, complete

program, or even complete configuration containing several programs).

- The aim of the formalization, i.e. which methods should be applied to the generated formal model (cf. Fig. 1).
- The formal model used to describe the PLC program.

III. Sources used for the Formalization

A. Programming Language

For many years, in the area of PLC control there have been only proprietary programming languages used to program one special PLC of one special vendor. However, in 1993 the International Electrotechnical Commission (IEC) published the IEC 61131 International Standard for Programmable Controllers [3]. Part 3 of this standard defines a suite of five programming languages that are used increasingly often instead of the proprietary languages. In the standard there are two textual languages: Instruction List (IL) and Structured Text (ST), and two graphical languages: Ladder Diagram (LD) and Function Block Diagram (FBD). A fifth language, the graphical or textual Sequential Function Chart (SFC) is proposed to define the structure of a PLC program.
Ladder Diagram has its roots in the USA. It is based on the graphical presentation of Relay Ladder Logic.
Instruction List is its European counterpart. As textual language, it resembles assembler.
Function Block Diagram is very common to the process industry. It expresses the behavior of a controller as a set of interconnected graphical blocks, like in electronic circuit diagrams.
Structured Text is a very powerful high-level language that is close to Pascal.
The fifth language is the *Sequential Function Chart* (SFC). SFC elements are defined for structuring the internal organization of programmable controller programs and function blocks.

B. Additional Information

In some formalization approaches in addition to the code of the controller information about the plant or expert knowledge is used.

Additional information about the control plant is needed, if in V&V properties of the controlled plant should be tested. The system environment or the controlled system is modeled and is verified together with the model of the program. The model of the system under control can also be used in simulation. Knowledge about the physical structure of the plant is especially useful in re-interpretation of the controller.

When trying to formalize a given program on a higher level of abstraction, i.e. not line by line but by identifying useful structures. The know-how of expert programmers can be used in this identification process. This know-how is either applied directly or transferred to a database for the use of non-specialists.

IV. Levels of Formalization

The approaches applied on the formalization of the PLC programs vary in their range in three different classes.

- **The formalization of parts of the control program (algorithms):** There are approaches that are suitable for the formalization of algorithms but not of complete programs because they have no means for describing all the necessary language elements of the PLC. These approaches are especially useful if a specific function of a controller has to be tested or transferred to another system.
- **The formalization of complete programs:** In this class a model of the behavior of the program is derived. Most work done in the area of the formal methods and formalization of PLC is in this class. After finishing the model it can be tested using different test methods of verification and validation. The tested and optimized program can be re-implemented on the original source system or on a new hardware.
- **The formalization of the whole control configurations:** Complete configuration of a control system consisting of several PLC programs on one or more PLCs. This approach is important for the re-implementation of control system software on new control system hardware.

V. Aim of the Formalization

Two main important fields for the formalization of PLC programs have been growing up in the recent time: Reverse-Engineering and Verification and Validation.

A. Reverse Engineering

Re-implementation or Reverse Engineering is a process of evaluating something to understand how it works in order to duplicate or enhance it.

There is a constant need for updating and renovating business-critical software systems for many and diverse reasons: business requirements change, technological infrastructure is modernized, the government changes laws etc. Therefore, in the area of software engineering the subjects of reverse engineering and system renovation become more and more important. The interest in such subjects originates from the difficulties that are encountered when attempting to maintain extremely large software systems. Such software systems are often called legacy systems, since it is a legacy of many different people that have developed and maintained them. It is not hard to understand that it is very difficult – if not impossible – to maintain them. The reverse engineering of PLC programs is required, as there is often no documentation for the implemented system.

Program transformations have been advocated as a method for accomplishing reverse engineering. The hypothesis is that the original source code can be progressively transformed into alternative forms, but with the same semantics. At the end of the process, an equivalent program is acquired, but one which is much easier to understand and more maintainable.

B. Verification and Validation

The second aim of formalization is Verification and Validation (commonly referred to as V&V) of the PLC program. In recent years the interest for analyzing PLC programs has increased to help in deciding if the program verifies specifications like safety, liveness and timing properties. In [4] an example is given that shows how V&V can help improving a controller. V&V is concerned with answering two fundamental questions. Speaking broadly, validation is concerned with building the right product, and verification is concerned with building the product right.

V&V techniques can be applied throughout the product lifecycle to help assure that the correct product is being built and that the product is being built correctly. Two levels of research are done on the verification of PLC programs: verification of the program with a model of the plant or the environment, or the verification of the program with respect to the control specification.

To make analytic techniques computationally tractable, abstract models in the language used by the analytic tools must be generated from the specifications, code, and models. Currently, the generation of these abstract models is both a practical and theoretical bottleneck in analytic V&V.

There is a variety of V&V methods (e.g. static analysis, abstract interpretation, runtime verification automated abstraction, invariant generation, slicing). However the two most promising formal methods used in V&V so far are model checking and theorem proving.

Model checking is a method for formally verifying finite-state concurrent systems. Specifications about the system are expressed as temporal logic formulas, and efficient symbolic algorithms are used to traverse the model defined by the system and check whether the specification holds or not [5].

Theorem proving proves that an implementation satisfies a specification by mathematical reasoning. Implementation and specification are expressed as *formulas* in *a formal logic.* The required relationship (logical equivalence/logical implication) described as *a theorem* has to be proven within the context of a proof calculus. The proof system is a set of axioms and inference rules (simplification, rewriting, induction, etc.)

VI. Model used for the Formal Description

The following formalisms are among the important formal models used on PLC programs:

- **Automata:** automata and also timed or hybrid automata are used in the verification of PLCs. For more information on hybrid automata see [6].
- **Petri nets:** different types of Petri nets are used as formal models.

There are also other formalisms like Condition/Event systems known as C/E, Higher Order Logic, Synchronous Languages, and General Transition Systems.

VII. Survey And Examples

In the following, examples are listed and categorized according to the Target of formalization.

A. Reverse Engineering or Re-implementation

- In [7] an automatic re-documentation, reformatting and transformation of IL programs into a hypertext on the basis of HTML is given. This method is intended for Software-visualization, static analysis, and source code navigation.
- A Reverse Engineering method for the conversion into a control description with state diagrams is given in [8]. These state diagrams are formatted according to a functional hierarchic structure. The source here is IL and additional information about the controlled system is needed.
- In [9] FBD from a source system is translated and re-implemented to transfer a complete controller configuration to a new control system (known as migration of process control system software). Here the know-how of programmers is important for the translation. In the approach the translation is not based on single FBD elements but on the identification of functional structures (e.g. a set of connected FBD elements describing some function).

To identify these structures the know-how of the programmers is used to build a data-base containing functional structures of the source system and corresponding structures of the target system.

B. Verification and Validation

These examples are further classified according to the language the original PLC program is written in:

1. Program source code written in ST

- In [10] the variables as well as the different constructs of the ST language are modeled using communicating automata. There are automaton models for the **while, for, if then else, and negation** constructs. The automata of the used variables and of the constructs are composed to express the ST blocks. The resulting model tends to be very large. This technique has been used to translate ST programs into input code for the model-checker Cadence SMV. Each component is defined as a module and can be re-used.

2. Program source code written in IL

- In [11] a model is given for instruction list. The structure of the PLC, the program logic, the process inputs, and the process outputs are modeled using Condition/Event systems [12]. The Model-Checker VERDICT [13] is used to verify the properties of the composition of the models together with the given model of the system.

- In [14] automata are used to model PLC algorithms that are programmed in a sub-set of IL. Timers of type TON are also modeled as timed automata. Complex language elements such as function and function block calls are not considered. The formalization is restricted to Boolean variables. A tool was developed based on this work described in [15]. This tool translates programs written in IL to timed automaton. Variables of type integer in this model are also allowed. The conversion of the IL program to the models is divided to timed automata and un-timed automata (which is in general larger than the timed part). The un-timed part is minimized using the toolset Caesar/Aldebaran Development Package [16]. Information about the system environment is needed and can be modeled using a timed automata synchronized with a model of the PLC program as an interface to the input and output variables. To verify the model the UPPAAL model checker is used [17].

- Static analysis is applied to programs written in IL in [18]. An abstract interpretation algorithm is presented which allows static checking for possible run-time errors and provides information about the program structure, this method checks for dead code or infinite loops.

- A method for translating an IL program into a transition system is presented in [19]. LTL is used to write behavioral properties of the controlled system and coding of the operational semantics into SMV that is used for the check for properties.

- In [20] programs in IL are modeled as Petri nets. The model of the program is then composed with Petri net models of the process into one model of the controlled system. The properties to be verified are expressed in CTL and the SMV model checker is used.

3. Program source code written in SFC

- The aim of [21] is an effective translation of the SFC syntax into SMV [22] model checking source code. Using SMV the SFC is verified for reachability properties, causal dependencies between the input variables and reachability.

- In [23] a timed automaton of the plant or the controlled system is built. The PLC program written in SFC is translated by creating a discrete transition system for the logic part and introducing a clock variable for each timer. After that composition of the timed automata of the plant and the controller model the valid ranges of the clock variables are determined using the tool Hytech [24].

- A method to convert SFC to a Hybrid Automata System (HAS) is given in [25]. The process under control is also modeled as a Hybrid Automata System. Both models are then synchronized and then an algorithmic solution to the reachability problem of the combined HAS description is applied.

- The work of [26] was carried out as a part of a case study for the EC VHS (verification of Hybrid Systems) [27]. The goal of this work is to verify and design a PLC program for an experimental chemical Plant. Promela/SPIN [28] is used for the verification of the PLC program and to derive time optimal schedules with reasonable time and space requirements.

- Further works on the verification and validation of SFC can be found in [29].

4. Program source code written in LD:

- In [30] an approach for the automated verification of LD and timed function blocks (of type TON) is presented. The algorithms are translated into state automata The SMV as symbolic model checker is used to check for the properties.

- Translation of LD programs into Complementary-Places Petri Nets [32], [33] is performed in [31]. This type of PN contains an annotation for a couple of places associated to the values for the token (true or false) and for the modeling of Boolean variables. The LD operators are modeled by a PN type structure and the whole specification of the LD is then simulated with the PN.

5. Program source code written in FBD:

- In [34] a toolset called PLCTOOLS has been introduced. The FBD programs are modeled and are described as High Level Timed Petri Nets (HLTPN) [35]. HLTPN are used for validating the design and generating the code. *MATLAB / S1MULINK* provides suitable means for specifying and simulating the plant. This work can be considered also as Re-engineering method since from the FBD and using this tool a code in C++ of the FBD program can be generated and the reuse of the existing software on a PC system is possible.

- Controllers defined according to IEC 61499 [36] are formalized in [36]. The controller code is in FBD format and the overall system is organized in IEC 61499 Function Blocks. These Blocks contain Execution Control Charts (ECC), which are state machines connecting event inputs with algorithms and event outputs. In the approach this complete structure is automatically translated to Signal-Net Systems (SMS). The tool VEDA allows the modeling of the controlled plant and the controller by means of Signal-Net Systems [38]. On the combined model of plant and controller model-checking is performed using SESA [39] (Signal/Event system analyzer) – a powerful model-checker for Signal-Net models.

C. Summary

Table 1 summarizes the examples according to the criteria discussed in the last sections. It has to be mentioned that – besides all efforts – there is no method at the moment that is capable of the automatic formalization of complete PLC projects according to IEC 61131-3.

VIII. Conclusions and Outlook

Our interest was to present different approaches to formalize PLC programs and to give examples on this field. These works are categorized in four criteria: according to the source in which the program is written in, according to the level of formalization – the whole program or only part of it – is needed to be formalized, according to the target and the model used for this formalization.

One reason for the restriction of formalization approaches to single programs or algorithms is the problem of getting the project information from a PLC programming tool. At the moment there are only vendor specific formats. However, recently the PLCopen – a PLC user organization (see http://www.plcopen.org) – started a Technical Committee to define an XML based format for projects according to IEC 61131-3. This new format will ease the access of formalization tools to all relevant information of a PLC project.

IX. References

[1] G. Frey and L. Litz: *Formal methods in PLC programming*. Proc. of the IEEE Conf. on Systems, Man and Cybernetics (SMC'2000), Nashville, USA, Oct. 2000, pp. 2431-2436.

[2] S. Lampérière-Couffin, O. Rossi, J.-M. Roussel, J.-J.Lesage: *Formal Validation of PLC Programs: A SURVEY*. Proc. of the European Control Conference (ECC99), Karlsruhe, Germany, Sept. 1999, paper N° 741.

[3] International Electrotechnical Commission. IEC International Standard 1131-3, Programmable Controllers, Part 3, Programming Languages, 1993.

[4] O. De Smet, S. Couffin, O. Rossi, G. Canet, J.-J. Lesage, Ph. Schnoebelen, H. Papini*: Safe programming of PLC using formal verification methods.* Proc. 4th Int. PLCopen Conf. on Industrial Control Programming (ICP'2000), Utrecht, the Netherlands, Oct. 2000, pp. 73-78.

[5] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill und L.J. Hwang: *Symbolic Model Checking: 10^{20} States and Beyond.* Information and Computation*, 98: pp. 142-170, 1992.*

[6] T. A. Henzinger: *The Theory of Hybrid Automata*, Proc. of the 11th Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, July 1996, pp. 278-292.

[7] R .Kliewer: *Reverse Engineering von Steuerungssoftware*. Ph.D. thesis, University of Kaiserslautern, Germany, Institute for Production-Automation (PAK), 1999.

[8] A. Storr und S. Kraneis: *Restrukturierung und Reverse Engineering von SPS- Programmen.* Fachtagung Entwurf komplexer Automatisierungssysteme (EKA'97), Braunschweig, 1997, pp. 446-461.

[9] Fay, A.: *Methoden zur Unterstützung der Migration von Prozessleitsystem-Software.T.A.* atp 44, Heft 6 2002, pp. 39-44.

[10] G. Canet: *Vérification des programmes écrits dans les langages de programmation IL et ST définis par la norme IEC 61131-3.* Thèse ENS de Cachan, December 2001.

[11] H. Treseler, N. Bauer, S. Kowalewski: *Model-Checking von AWL-Programmen*. Lambda. Technischer Bericht, 10. Februar 2000, gekürzte Version in Fachtagung Verteilte Automatisierung, Magdeburg, 22./23. März 2000, pp 286-293.

[12] R.S. Sreenivas and B.H. Krogh: *On Condition/Event Systems with Discrete State Realizations.* Discrete Event Dynamic Systems: Theory and Applications, Kluwer Academic Publishers, Boston, USA, Vol. 1, 1991, pp. 209-236.

[13] S. Kowalewski, N. Bauer, J. Preußig, O. Strusberg, and H. Treseler: *An environment for model-checking of logic control systems with hybrid dynamics*. In Proc. IEEE int symp. On Computer Aided Control System Design, 1999, pp 97-102.

[14] A. Mader , H. Wupper: *Timed Automaton Models for Simple PLC.* Proc. of the Euromicro Conference on Real-Time Systems 1999, IEEE Computer Society Press, June 1999, pp. 114-122.

[15] H. X. Willems: *Compact timed Automata for PLC Programs*. Technical Report CSI-R9925, University of Nijmegen, November 1999.

[16] CADP home-page: http://www.inrialpes.fr/vasy/cadp/.

[17] UPPAAL home-page: http://www.uppaal.com/.

[18] S. Bornot, R. Huuck, B. Lukoschus, Y. Lakhnech: *Utilizing Static Analysis for Programmable Logic Controllers*. Proc. of the 4th International Conference on Automation of Mixed Processes (ADPM), Dortmund, Germany, Sept. 2000, pp. 183-187.

[19] G. Canet, S. Couffin, J.-J. Lesage, A. Petit and Ph. Schnoebelen. *Towards the automatic verification of PLC programs written in Instruction List*. Proc. of the IEEE Conf. on Systems, Man and Cybernetics (SMC), Nashville, USA, Oct. 2000, pp. 2449-2454.

[20] T. Mertke, T. Menzel: *Methods and tools to the verification of safety-related control software*, IEEE International Conference on Systems, Man and Cybernetics, (SMC), Nashville, USA, Oct. 2000, pp. 2455 - 2457.

[21] S. Bornot, R. Huuck, B. Lukoschus, Y. Lakhnech: *Verification of Sequential Function Charts using SMV*. Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), Las Vegas, USA, June 2000, Vol. V, pp. 2987-2993.

[22] K.L. McMillan. The SMV system. Carnegie-Mellon University, February 1992. Draft version describing SMV revision 2.2.

[23] S. Kowalewski, S. Engell, R. Huuck, Y. Lakhnech, B. Lukoschus, and L. Urbina: *Using Model-Checking for Timed Automata to Parameterize Logic Control Programs*. 8th European Symposium on Computer Aided Engineering, Brugge, Belgium, 1997.

[24] Henzinger, T.H., Ho, P.H., Wong-Toi, H. (1997) A User guide to HyTech. http://www-cad.eecs.berkeley.edu/~tah/HyTech/

[25] G. Hassapis, I. Kotini, Z. Doulgeri: *Validation of a SFC software specification by using Hybrid Automata*.Proc. of the 9th Symposium on INformation COntrol in Manufacturing INCOM'98, Nancy-Metz, France, June 1998, Vol. II, pp. 65-70.

[26] Ed. Brinksma and A. Mader: *Verification and optimization of a PLC control Schedule*. Int. Journal on Software Tools for Technology Transfer, 4 (1), 2000, pp. 21-33.

[27] A. Mader, E. Brinksma, H. Wupper, and N. Bauer: *Design of a plc control program for a batch plant, VHS case study 1*. European Journal of Control, 7 (4), 2001, pp. 416-439.

[28] G.J. Holzmann: *The model checker spin*. IEEE Trans. on Software Eng., 23(5): May 1997, pp. 279- 295.

[29] J.-M Roussel and J.-J. Lesage: *Validation and Verification of grafcets using finite state machine*. Proc. of the IMACS-IEEE Multiconference on Computational Engineering in Systems Applications (CESA'96), Lille, France, July 1996, pp. 758-764.

[30] O. Rossi, Ph. Schnoebelen: *Formal Modelling of Timed Function Blocks for the Automatic Verification of Ladder Diagram Programs*. Proc. 4th Int. Conf. Automation of Mixed Processes: Hybrid Dynamic Systems (ADPM), Dortmund, Germany, Sept. 2000, Shaker Verlag, Aachen, Germany, 2000, pp.177-182.

[31] I. Hatono, K. Baba, M. Umano, H. Tamura: *Automatic Generation of Fault Detection Models for Programmable Controller-Based Manufacturing Systems Using Complementary-Places Petri Nets.*, IFAC World Congress 1996.

[32] S. Christensen, and N.D. Hansen: *Coloured Petri Nets Extended with Place Capacities Test Arcs and Inhibitor Arcs*. Proceedings of 14th International Conference on Application and Theory of Petri Nets, Chicago, USA, Springer-Verlag 1993, pp. 186-205

[33] C. Lakos, S. Christensen: *A General Systematic Approach to Arc Extensions for Coloured Petri Nets*. Proc. of the 15th International Conference on Application and Theory of Petri Nets, Zaragoza, Spain, 1994, Springer-Verlag, pp. 338-357.

[34] L. Baresi, M. Mauri, A. Monti, and M. Pezze. *PLCTools: Design, Formal Validation, and Code Generation for Programmable Controllers*. Proc. of the IEEE Conference on Systems, Man, and Cybernetics (SMC), Nashville, USA, Oct. 2000, pp. 2437-2442.

[35] Ghezzi, D. Mandrioli, S. Morasca, and M. Pezzè: *A Unified High-Level Petri Net Model for Time-Critical Systems*. IEEE Transactions on Software Engineering, 17(2): Feb. 1991, pp 160-172.

[36] Function Blocks for Industrial Process Measurement and Control Systems International Electrotechnical Commission, Tech. Comm. 65, Working group 6, Committee draft.

[37] V. Vyatkin, H.-M. Hanisch: *Modelling of IEC 61499 function blocks - a clue to their verification*. Proc. of the XI Workshop on Supervising and Diagnostics of Machining Systems, Karpacz, Poland, March 12-17, 2000, pp. 59 – 68.

[38] P. Starke: *Symmetries of signal-net systems*. Workshop on Concurrency, Specification and Programming, October 2000, pp. 285-297.

[39] P. H. Starke and S. Roch*: Analysing Signal-Net systems. Report, Humboldt University Berlin, Institut für Informatik, Aug. 2000.* http://www.informatik.hu-berlin.de/lehrstuehle/automaten/tools/#sesa.

Table 1: Classification of the Examples

Classification						
Reference		Language	Source Additional Information	Level	Aim	Model
Kliewer	[7]	IL	Without additional information	Program	Re-Engineering	HTML
Storr	[8]	IL	Plant	Program	Re-Engineering	Automata
Fay	[9]	FBD	Programmers Know-How in database	Configuration	Re-Engineering	No model
Treseler et al.	[11]	IL	Plant	Program	Verification	Automata
Bornot et al.	[21]	SFC	Without additional information	Program	Verification	SMV Input Code
Willems	[15]	IL	Plant	Program	Verification	Timed Automata
Mader et al .	[14]	IL	Without additional information	Algorithm	Verification	Timed Automata
Brinksma et al.	[26]	SFC	Plant	Program	Verification	SPIN model
Kowalewski et al.	[23]	SFC	Plant	Program	Static analysis	Automaton
Bornot et al.	[18]	IL	Without additional information	Program	Verification	No model
Canet et al.	[19]	IL	Without additional information	Program	Verification	Automata
Roussel et al.	[29]	SFC	Without additional information	Program	Verification	FSM
Rossi et al.	[2]	SFC& LD	Without additional information	Program	Verification	Automaton
Mertke et al.	[20]	IL	Plant	Program	Verification	PN
Hassapis et al.	[25]	SFC	Plant	Program	Verification	Hybrid Automata
Rossi et al.	[30]	LD	Without additional information	Program	Verification	FSM
Baresi et al.	[34]	FBD	Without additional information	Algorithm	Verification	PN
Hatono et al.	[31]	LD	Without additional information	Program	Verification	PN
Vyatkin et al.	[36]	FBD	Plant	Program	Verification	SNS
Canet	[10]	ST	Without additional information	Algorithm	Verification	Automata