

Technische Universität München – Fakultät für Maschinenwesen

Application concept and evaluation of a formal specification approach
usable by engineers for retrofitting production automation by software
changes

Suhyun Cha

Vollständiger Abdruck der von der Fakultät für Maschinenwesen
der Technischen Universität München zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs
genehmigten Dissertation.

Vorsitzender: Prof. Dr.-Ing Wolfgang A. Wall

Prüfende/-r der Dissertation:

1. Prof. Dr.-Ing. Birgit Vogel-Heuser
2. Prof. Dr. rer. nat. Bernhard Beckert

Die Dissertation wurde am 09.02.2021 bei der Technischen Universität München
eingereicht und durch die Fakultät für Maschinenwesen am 14.04.2021 angenommen.

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <<http://dnb.ddb.de>> abrufbar.

Application concept and evaluation of a formal specification approach usable by engineers for retrofitting production automation by software changes

Autorin:

Suhyun Cha

ISBN: 978-3-96548-110-7 (Print)

ISBN: 978-3-96548-111-4 (E-Book)

1. Auflage 2021

Cover: sierke WWS GmbH
sierke MEDIA, Göttingen

Copyright-Hinweis:

Das Buch einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlags unzulässig und strafbar. Der Nutzer verpflichtet sich, die Urheberrechte anzuerkennen und einzuhalten.

© 2021 sierke VERLAG

sierke WWS GmbH

Sternstraße 7

37083 Göttingen

Tel. +49 551 503 664 5

info@sierke-verlag.de

<http://www.sierke-verlag.de>

Acknowledgments

I would like to thank my advisor, Professor Birgit Vogel-Heuser, for her support, including many valuable discussions of my research. She has been a great professor and a mentor. I would also like to thank my thesis committee members, Professor Bernhard Beckert, for giving me great feedback and opinions, and Prof. Wolfgang A. Wall for serving as the committee head.

I am also very grateful to all the colleagues of the Chair of Automation and Information Systems at the Technical University of Munich for all the research-related discussions and personal conversations which encourage and motivate me all the time.

Most of the dissertation is based on the German Research Foundation (DFG) Priority Programme 1593 (SPP1593) IMPROVE APS project result in close cooperation with the group of Professor Bernhard Beckert, Dr. Mattias Ulbrich, and Alexander Weigl in Karlsruhe Institute of Technology; I would like to thank all project members. Thanks also to Thomas Mikschl, who has been a great support in this project for implementation.

In addition, I would like to thank my family, Soobin and Sunyul in Munich, and Mrs. Kang, Mr. Cha, Sujeong, Mrs. Park, Mr. Lee, and Jiyoung in Korea, who support me all the time.

Table of Contents

1.	Introduction	1
2.	Field of Investigation	5
2.1.	Technical characteristics of automated production systems	5
2.1.1.	Process automation and aPS	5
2.1.2.	Control software of aPS.....	6
2.2.	Control software engineering of automated production systems.....	10
2.2.1.	Implementation of the aPS – focusing on the software development.....	10
2.2.2.	Control software quality assurance.....	11
2.3.	Formal verification terms and basics	12
2.3.1.	A framework for formal verification	13
2.3.2.	Regression verification	14
2.3.3.	Formal specification of the requirement.....	16
3.	Requirements on a Formal Specification of Reactive System’s Control Software Behavior.....	19
3.1.	Requirement considering target system characteristics	19
3.2.	Requirements considering engineering processes of aPS	21
3.3.	Requirements considering the users’ point of view.....	23
3.4.	Summary of requirement	25
4.	State of the art.....	27
4.1.	Formal specifications behavior: language and utilization	27
4.1.1.	Text based temporal logics	27
4.1.2.	Graphical and signal pattern based languages	30
4.1.3.	Specification approaches for automation systems	31
4.2.	Monitoring of aPS execution	36
4.3.	Usability studies of specification approaches.....	37
4.4.	PLC software programming tools.....	38
4.5.	Discussion of the research gap	38
5.	A Concept of the Table-Based Formal Specification Language For Reactive System Software.....	41
5.1.	Concept overview	42
5.2.	Generalizing test tables as a specification representation.....	44
5.2.1.	Structure of GTTs.....	44
5.2.2.	Value referencing and generalization	45
5.2.3.	Generalizing duration	47
5.2.4.	Simplifying exclusion.....	48
5.3.	Application example.....	50
5.3.1.	Description of the module	51
5.3.2.	Requirements of the module.....	52

5.3.3.	Test cases for the module in test tables.....	52
5.3.4.	Behavior specification in GTTs.....	53
5.4.	Extension of the developed approach	56
5.4.1.	Debugging through the counterexample presented as related to GTT(F).....	57
5.4.2.	Achieving preliminary specification out of IEC 61131-3 program in GTT (B1).....	58
5.4.3.	Obtaining monitors by the transformation of the specification into IEC 61131-3 function block (B2).....	62
5.5.	Summary: table-driven two-way quality assurance process – embedding user-friendly formal specification in the engineering process.....	66
6.	Implementation	69
6.1.	Overview.....	70
6.2.	Generating and editing the generalized test tables.....	71
6.3.	Connecting to the verifier	72
6.4.	Verification result representation.....	72
7.	Evaluation of the approach	75
7.1.	Feasibility analysis through application case studies.....	76
7.1.1.	Behavior description of a single component – in the viewpoint of module developer (use case 1).....	77
7.1.2.	Behavior description of a multi-component module – in the viewpoint of application engineer (use case 2).....	79
7.1.3.	Obtaining changed specification based on the preliminary specification generated (use case 3).....	81
7.1.4.	Monitoring of the target function block – automated generation of the monitoring block (use case 4).....	83
7.2.	Empirical study of the usability evaluation.....	87
7.2.1.	Experimental hypotheses	87
7.2.2.	Experiment planning.....	89
7.2.3.	Participants profile	92
7.2.4.	Experiment results	95
7.2.5.	User evaluation summary	100
7.3.	Expert evaluation from industry	102
8.	Assessment of the Fulfillment of the Requirements.....	105
9.	Conclusion and Outlook.....	109
10.	Literature.....	113
11.	List of Figures.....	131
12.	List of Tables	135
Appendix A.	Materials of experiment 1 for Chapter 7	137
Appendix A.1	Demographic data and prior knowledge questionnaire	137
Appendix A.2	Evaluation tasks with possible answers.....	141

Appendix A.3	Given handout to be used during evaluation.....	153
Appendix B.	Materials of experiment 2 for Chapter 7.....	157
Appendix C.	Materials of experiment 3 for Chapter 7.....	164

1. Introduction

Automated Production Systems (aPS), including manufacturing machines and logistics, are required to cope with various and varying production requirements over their long lifecycles [VFST15]. These various requirements lead to the increased complexity of aPS [DKKK12, FKWV19, McBu00], introducing a lot of changes in various forms and objectives with different reasons, and, accordingly, their engineering becomes more complex, following current trends and increasing customer flavor varieties [VFST15]. Malfunctions (i.e., severe failures) of aPS might cause severe issues economically and safety-relatively [UIVo18a], as aPS are socio-technical systems, which are defined as operational processes and people (i.e., operators) are inherent parts of the system [Somm15]. Therefore, aPS have high requirements on availability and reliability during the running period [VFST15] not only for its productivity concerning dramatic maintenance or product producing cost but also for its safety not to harm any personnel or customer of the product. In the meantime, the proportion of system functionality realized in software is increasing [Thra10]; the control software of aPS is, therefore, to be developed and revised continually, e.g., due to bug fixing or changed/new functionalities [VFFL15], requiring continuous quality (re-)assurance activities accordingly.

In today's industrial practice, software quality is commonly achieved by dynamic validation either by manual stepwise testing or by running automated testing [UIVo18b] as a quality assurance activity "in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component" according to the definition of ISO [ISO10]. However, one of the main weaknesses of traditional testing is coverage. One test case covers only a particular execution of the software and, thus, many possible cases remain uninvestigated. Therefore, testing is useful for typical or expected faults while unpredictable and rare cases, which can also have severer consequences, are less likely to be discovered using testing. In aPS engineering processes, changes also occur during the commissioning and start-up procedure [VFST15] for on-site assembling [Voge09] and fine-tuning. According to the time delay analysis [Hack18, ReWü07, VFST15], software debugging during the commissioning takes more than half of the effort though it requires small changes. In detail, it is observed that 15-25% of the total project time is used for commissioning, and almost 63% of it is used for control software debugging [ReWü07]. This commissioning procedure also requires testing (mostly manual [VFFU17]) under the high time pressure [UIVo18a] and the high chances of damage either by operating incompletely integrated machine or by switching the operation mode [VFRF15] for testing, since quality assurance through testing requires real equipment executions.

As one other main method of quality assurance, there is formal verification. In this formal verification framework, the behavior of the target artifact (typically formal model) is examined mathematically and proved (or guaranteed) as conforming/violating given conditions [CWAC96] through the mathematical and exhaustive checking, providing full coverage in contrast to testing. Since it calculates the given artifacts by (mathematical) analysis, the equipment, i.e., plant and machine in aPS' case, is not needed to be running in contrast to testing and, thus, damages during the (testing) execution do not have to be concerned. The conditions to be satisfied are actually the requirement that is desired to be implemented (or designed in case of early stage of the development) in and satisfied by the artifact. These are stated and documented in a way that could be comprehended and processed, which is called as *specification*. Although the specification is the essential entity within the formal verification context, providing the criteria to decide the target-under-verification is correct or not, a good specification is still lacking [Sesh12, UIVo18b]. Specifications for many aPS in operation are not easily found [VFST15] because, on the one hand, aPS have been built decades ago going through a lot of changes in the meantime without appropriate and precise history records, and on the other hand, the way of providing formalism, such as temporal logics, is not what typical automation engineer would take and develop [Holz02]. Ultimately, the absence of formal specification hinders applying formal verification along the automation engineering processes.

The importance of the specification is highlighted not only for verification purpose. Specifications are involved in almost all engineering processes including the maintenance phase, not only for the initial development processes but also for modification (i.e., change implementation) processes [Somm15] as it indicates the intention to be implemented. Explicit documentation is also required to comply the legal regulations and guidelines, such as Good Automated Manufacturing Practice (GAMP [ISPE20]), for some particular type of product production. Furthermore, formally specified requirements could be handled systematically to synthesis or generate another artifacts (e.g., program code) by transforming it or forming/extracting information into/from formalism respectively [Li14]. It would become more beneficial in the context of aPS engineering, for which various types of developers and engineers collaborate by communicating with each other and delivering their results to others, even only within the software discipline [VFST15].

The objective of the research in this thesis is to provide a support for increased control software quality for aPS in production automation. A formal specification approach for the aPS control software behavior description to support software is proposed aiming at accessibility from the automation engineers and systematic manipulation for further utilization, tackling the motivations and challenges above. Accessible notations to describe comparably small scale software changes

retrofitting the production automation are approached, grounded on the practices which are typically used in the industry settings. Besides used as a formal specification in formal verification framework, additional concepts of utilizing specifications are also presented to strengthen the control software quality assurance through the proposed approach: generating monitoring block from the specification and mining the preliminary specification in a user-guided way (Figure 1).

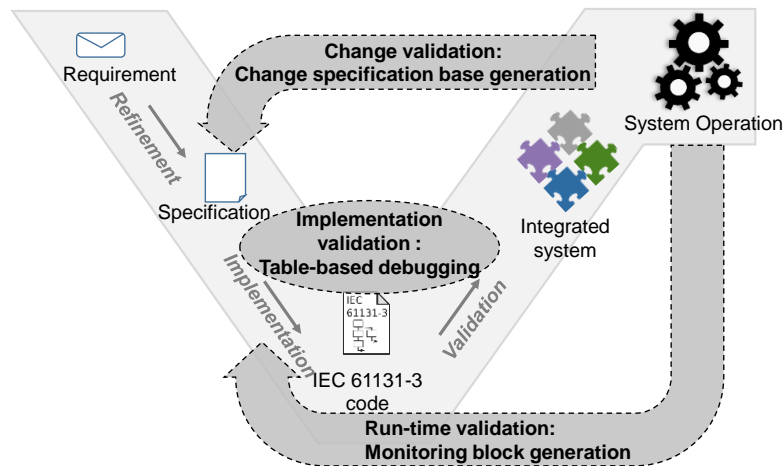


Figure 1: Schematic representation of the thesis scope: complementing quality control driven by specification

This thesis has been developed in the context of the German Research Foundation (DFG) Priority Programme 1593 (SPP1593) IMPROVE APS project in close cooperation with the group of Professor Bernhard Beckert and Dr. Mattias Ulbrich in Karlsruhe Institute of Technology (KIT). The formal basis of the introduced concept (i.e., Generalized Test Tables) has been proposed by KIT (Alexander Weigl) as well as the regression verification tool (cf. [Weig21]). The contents and contributions of this thesis are based on previous publications by the author, which are [BCUV17, CUVW17, CUWB19, CVWU21, CWUB18a, CWUB18b, VoCh20, WWUU17]. The overall requirements for the concept have been derived jointly, and the focus of this thesis is put on the automation engineering perspectives and applications of the concept from the mechanical requirements and use cases, to the notation without its formal underpinning. The work in this thesis also include the empirical evaluation with students and the expert evaluation with industrial engineers.

The thesis is structured as follows. First, an overview on the field of investigation together with basic definitions is provided in Chapter 2, considering the characteristics of the target system and its engineering domain that presented approaches are to be applied to, precisely aPS as a specific type of mechatronics system. In Chapter 3, considered requirements as a formal specification method are presented, separated into the different views on system characteristics, engineering characteristics, usability to be satisfied by the presented approach. The related works which have

tackled the presented requirements are introduced and analyzed in Chapter 4 to identify the research gap as a current state of the art. In Chapter 5, the approach developed in the course of this thesis to satisfy the identified requirements and research gap are presented. The approaches are presented into two parts: presenting the formal specification syntaxes and their applications as the concepts for further utilization. The prototypical tool concept, embedded within the IEC 61131-3 code development environment, is overviewed in Chapter 6 to demonstrate how the proposed approach is to be implemented as an add-on to the aPS control software development process. The approaches are evaluated in Chapter 7 in three different manners: feasibility studies with practical application examples, usability studies through empirical observation from the survey, and industry expert feedback through the intensive discussion. In Chapter 8, the conclusive finding and the assessed results obtained throughout the evaluation are presented. Finally, the thesis concludes the summary of the achieved results and the outlook on future works in Chapter 9.

2. Field of Investigation

The approach is researched and developed to strengthen the quality of the implemented control software, easing engineering processes of aPS, especially for the case of control software changes. To specify the target of the presented approach and the domain requirements, descriptions about the target system, engineering processes, as well as the related techniques are to be introduced. First, the target system has to be narrowed down regarding the relevant characteristics to apply the presented approach. The root of the formal specification with respect to the formal verification comes from the computer science field targeting pure software behavior verification; this work aims at aPS, which realize the technical processes being interwoven with the multi-discipline artifacts. Second, among the various constituents of aPS, the focus is put on the control software part of the system and its small changes, especially during/after commissioning and start-up, more specifically. The control software gets more complicated to cope with the increasing and varying requirements and takes part in a bigger role than before by being attended as its flexibility [VFST15]. The description of these two characteristics answers the question – where this approach aims at to be applied. Along the engineering process of the aPS control software, the quality assurance step will be focused as the target stage that the presented approach is to be applied, especially while implementing small changes. Among various techniques of system validation, formal methods and verification concept are under consideration of the approach. Thus, the fundamentals to understand the approach will be introduced in this chapter so that the following concept description could be easily understood.

2.1. Technical characteristics of automated production systems

The presented approach aims at the formal behavior specification of the software changes of aPS. Therefore, special properties of the target system and its control software will be presented in the following to allow for a better understanding of the requirements and the approach.

2.1.1. Process automation and aPS

The target systems, i.e., aPS, are process automation systems specialized in controlling production processes. APS are defined in [VFST15] as “...comprised of mechanical parts, electrical and electronic parts (automation hardware) and software, all closely interwoven, and thus represent a special class of mechatronic systems.” In the viewpoint of engineering and operations, “aPS, a particular type of mechatronic system on which this paper focuses, is designed-to-order systems. These are complex manufacturing systems, and they have a typical lifetime in the operation of several decades”. In the functionality point of view, aPS refer to the manufacturing and logistics

plants like production lines for automobiles or bottling beverages focusing on the production automation.

During the operation, the system status is changed by the technical processes with relevant parts (de)activated. The technical processes proceeded within aPS are based on the definition in [VFRF15] – a process by which the involved object, i.e., matter, energy, or information, is altered in its state (Figure 2). Thereby, a technical process is the totality of all operations in which material, energy, or information is converted, transported, or stored. Automation [IEC13] in general requires to access to information from this technical process (via sensors) to influence the technical process (via actuators) based on its behavior. Here, material, energy, and information are the objects of the technical processes. Also, human interaction is an important element in aPS since a human is one of the main elements contained in the process automation as an entity, who develops, follows the process event, controls and influences process, and handles the faults [VFRF15].

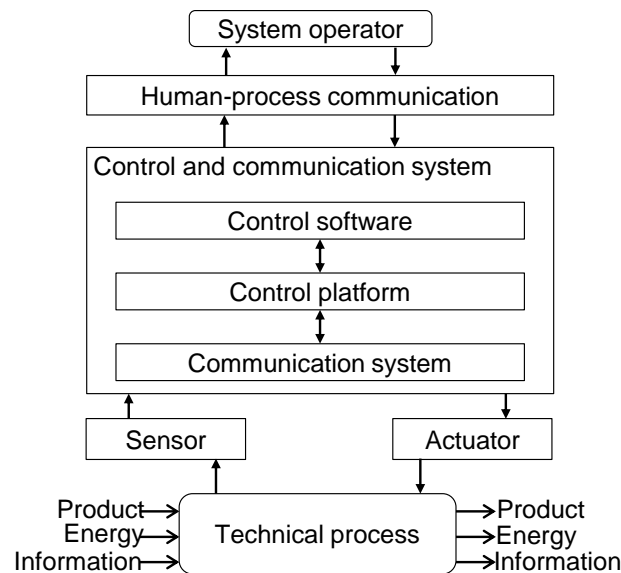


Figure 2: Technical process and technical system (reproduced from [VFRF15])

2.1.2. Control software of aPS

The execution of the aPS control software is based on the operation mode of the system [VRFS16]. As presented by Güttel et al. [GüWF08], the main operation modes of function blocks can be defined as follows:

- Automatic mode: the normal and usual behavior of the machine part
- Setup mode: the behavior of a machine part before or during the operation (if necessary) to set up the machine part to a specific status to proceed to the further operation mode
- Manual mode: the behavior to move the machine part as the manual input drives

- Semi-automatic mode: the behavior to operate the machine part automatically but also guided by the manual input
- Initialize: the behavior of a machine part executed at the beginning or after a resume followed by the halt of the machine (part) to be prepared for the further modes
- Shut down: the behavior of a machine part when the machine (part) finishes the execution
- Save stop: the behavior of a machine part when the machine (part) gets into the safe mode

These operation modes of aPS are explicitly defined and modeled including the states valid within each mode and the transitions of those states, as represented in OMAC standard typically for food and beverage production systems, and OMAC state machine is a part of PackML [Omac09], which accommodates the operational consistency of packaging as a standard. Thus, most of the control software is implemented its various functionalities considering its operation mode.

The control to execute computing and signaling is conducted by Programmable Logic Controllers (PLCs) for aPS. It consists of a central processing unit(s) (CPU), memory, and I/O units with all connected by bus systems [Hans15]. The control software is loaded on the memory part and executed by CPU, accessing I/O units, where all the sensor and actuators are connected ultimately for signaling. The control software, which is loaded on the PLC, is similar to the typical software in the sense that they cause some changes on the system as its task by being fetched, calculated, and update the system status. Typically, software running on a PC is executed to get a result at the end or in the middle of the execution by updating variable values on each statement in a program. Different from this typical software system, the PLC control software should react to every environmental signal concerned and, thus, this has to be under consideration of the executing by reading the values frequently. Therefore, PLC operates in a cycle-based way, which consists of internal processing, reading inputs, program execution, and updating output [LaGö99] (Figure 3). First, the PLC checks its state to ensure the availability of the operation regarding any error or interrupt in the initial processing. After that, the input status (i.e., sensor values) is read from the clamp and stored in the memory part. Based on the stored value to be participated in the calculation, the program execution takes place. Once all the output and internal variable values are calculated and decided, the output memory is read and put on the clamps so that the corresponding signals could be updated. These four steps repeat in a cyclic manner within PLC and continue until the controller is turned off. As the name already infers, a PLC requires a high-level language element, i.e., control software, to control the connected element to execute the technical process as the user of the

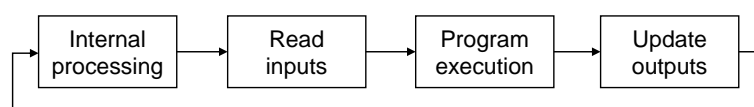


Figure 3: Schematic view of a cyclic operation of PLCs (reproduced from [LaGö99])

system intends [IEC09]. In other words, functional operations of the PLC is achieved by an appropriate (to the system mission) program. Many PLC programming standards have been suggested over the years and converged to IEC 61131 standard IEC 61131-3 [IEC09], being followed by most of the major PLC manufacturer [Hans15].

IEC 61131-3 consists of three types of program organization units (POUs) of *Programs*, *Function Blocks*, and *Functions*, implemented in five language varieties, two textual languages, which are Instruction List (IL) and Structured Text (ST), and three graphical languages, which are Ladder Diagram (LD), Function Block Diagram (FBD), and Sequential Function Chart (SFC). *Function* type POUs could be considered as a subroutine without any memory inside of it, meaning that it depends on the parameter value when it is called by other POUs and results in the same output whenever it is called as long as the input parameter is the same. In contrast, *Programs* and *Function blocks* could have the individual state of the execution inside of them, meaning that POUs in these type change their internal state and result in different output depending on the state, in which the POU is, even if they get the same input. *Programs* are the highest level to define the execution of the program within a *Task*, which is an execution control element (of the resource) as a whole, and a *Program* consists of a network of *Function blocks* and *Functions* within its implementation. Thus, Task(s) could be defined for the missions of the target system on a resource (i.e., PLC) in a particular configuration, and the mission is realized by calling the Programs and following Function Blocks and Functions.

Among the five different languages of IEC 61131-3, ST and SFC are mainly focused on in this thesis. ST is a high-level language with similar characteristics to Pascal language as well as C and C++ [Hans15]. It has the strength to handle arithmetic calculations and structured data types with compressed expression (e.g., compared to Ladder Diagram) as a textual language benefitting from its higher degree of freedom. SFC is one of the graphical IEC 61131-3 languages, derived from GRAFDCET (IEC 60848), typically used to describe the control flow structure of the system in a sequential manner. With the advantage that it is suitable to used various levels (i.e., higher level to define the abstract process description and lower level to define the code events in the detailed level) [Hans15]; it is accepted as one of the major languages of PLC programming. Since the syntax of ST is similar to the other conventional languages as mentioned and the syntax itself is not the focus of the work of the thesis, it is not described in detail, while that of SFC is to be introduced here mainly following [IEC09] and [Hans15] as a preliminary knowledge to understand the application concept further especially Section 5.4.2.

Basics of SFC

SFC consists of three main elements: steps, transitions, and actions. Steps are the unit of the sequence controls. Transitions indicate turning over the control from one step to another step decided by the transition guard. Actions are the events included in the step and executed while the step is active (e.g., Figure 4). Execution consists of activation, execution of the actions, and deactivation of the step in order. When the control is in the previous step, and the entry transition guard is satisfied (*Entry Guard of the Step* in Figure 4), then the step is activated, and the actions are executed. Actions shall be either associated with action blocks or declared in the nested form in all kinds of IEC 61131-3 languages; the action block cases are only considered in this thesis. After one cycle of the action execution, step deactivation is decided by checking the exit transition guard (*G2* in Figure 4). The following step is activated once the guard is satisfied. Each step might have entry and exit action, which is executed when the step is activated and deactivated, respectively.

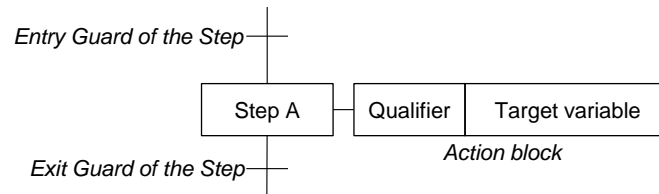


Figure 4: A step of SFC with entry and exit guard

An action block consists of a qualifier to indicate relevant action type and a Boolean variable as the target of the indicated action. Among all the behaviors of the qualifiers (Table 1), *SD* (Stored & Delay) and *SL* (Stored & Time Limited) are excluded since their timing behaviors are independent of the activation of the step. They violate the step activation, so they are not recommended due to the unpredictable risks [Hans15]. It is only considered that the final scan is disabled, so *P1* has the same behavior as *P*, although the action control can have a “final scan” option to enable or disable all actions executed one extra time after the deactivation. The enabled case can be easily derived by applying a disabled case with an additional execution at the end of the step.

Table 1: SFC action qualifiers

Requirement ID	Requirement description
<i>N</i>	Set the value during active(Non-stored)
<i>S</i>	Set and store the value (Set)
<i>R</i>	Reset and store the value (Reset)
<i>P</i>	Generate a pulse for one cycle (Pulse)
<i>P1</i>	Generate a pulse on step activation
<i>P0</i>	Generate a pulse on step deactivation
<i>L</i>	Perform for a certain time as long as it is active (time Limited)
<i>D</i>	Perform after a certain time as long as it is active (time Delayed)
<i>DS</i>	Performed after a certain time and store the value (Delayed Set)

2.2. Control software engineering of automated production systems

The presented approach is to be used for validation of developed control software, which lies in the process of aPS engineering. To define the field of investigation more clearly, aPS engineering processes focusing on the control software and its validation will be presented in the following.

2.2.1. Implementation of the aPS – focusing on the software development

Since aPS, a particular type of mechatronic system, are designed-to-order systems, each aPS engineering process goes through specific engineering methods depending on the own requirements and objectives of the system. The engineering life cycle has been depicted in [VFST15] (Figure 5), and the focus of the explanation here is put on software development. Basically and ideally, it is assumed that project-independent activities (upper part in Figure 5) are already done in the meta-level, or this could be regarded as to be done from the former project by extracting common elements over the projects. Common and reusable artifacts and their arrangement developed and kept in the solution repository to be used further within the individual project. Libraries modules are expected to be developed by the module developers [VFND18] during the project-independent. Later, customer-specific projects are designed and developed in particular (lower part in Figure 5). Coping with the customer requirements, a specific type of plant is developed. Application engineers organize the library modules adding some glue codes to generate the project-specific software. Though the overall process of engineering from requirement specification to the maintenance in the coarse level is similar to the other systems, like pure software systems, one of the big difference is that aPS have physical substances. Many plant pieces are shipped in part, assembled, and commissioned on-site [Voge09]. Although the software has been already validated regarding

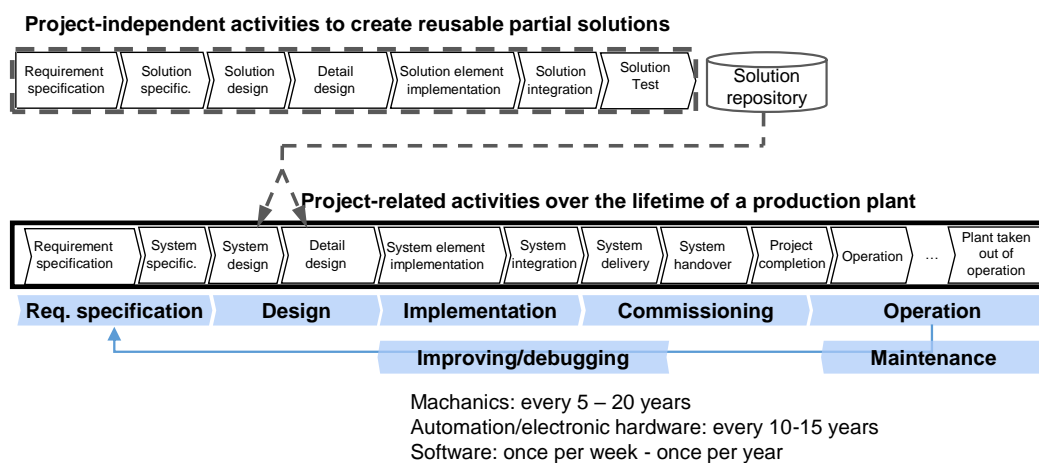


Figure 5: V-Modell XT with separation of the project-independent and project related activities (primarily presented in [VERE10], reproduced from the figure in [VFST15])

the module composition (earlier done in house), there comes further adjustment, not only for hardware adjustment but also for software modification, during the commissioning stage [VFST15]. As longer life cycle (usually decades) and corresponding aging cause components from different disciplines to go through re-engineering and modernization (mechanical engineering: 5-40 years, electrical engineering: 10-15 years, software engineering: weeks to months), it is unavoidable to face various changes with various reasons, e.g., physical wear and tear of the components [Kern19], changing requirements, lack of spare part availability [VHCR17], or technology trends. For the change implementation, project-specific engineering processes are gone through iteratively. Conclusively, aPS developers face many chances of developing (including validating) and changing/adjusting software functionalities over the lifecycle. This means that the effort to engineer aPS is not only put on the initial implementation entirely, but a lot of maintenance activities as well as following up changes including to update or add functionalities, to improve behavior, or to debug, occur very often (cf. [VoOc18]: frequency of software updates of machine and plant manufacturers were revealed as every less than 6 months for 44% and 13%, as every less than 12 months for 44% and 64% respectively) and also require quite amount of resources, i.e., cost and time (cf. [ReWü07]).

2.2.2. Control software quality assurance

During and after the implementation or the change realization, the implemented part of the system is required to be assured that it meets the specification as well as the expectation of the customer. Following the engineering steps, the implementation goes through unit level validation – for its functional conformance (in the implementation stage), integration validation – to assure the functional correctness as a part of the whole system (in the integration stage), and the acceptance testing – to be validated under the actual operating environment (commissioning and start-up stage).

These validations are typically being done in the form of testing, which is defined as “An activity in which a system or component is executed under specified conditions, the results are observed or recorded, and an evaluation is made of some aspect of the system or component” [IEEE14A]. Testing is intended to discover the system defects before its usage by executing the system (or system part) under test (SUT) with the artificial but plausible data to provide the input to the SUT, and observing the outcome [Somm15]. In other words, the correctness of the implementation is determined by controlling the system to traverse a specific execution path of the statements within the code. Strictly speaking, testing can show the executed system run is safe and correctly working without any error. However, since executing all possible execution paths is not feasible, only a limited number of test runs could be conducted, which means some existing errors (in the executed part of the system) could be figured out, but error-freeness cannot be assured. Formal methods,

which is one of the static analysis of verification techniques and represented by the formal verification, are considered as a complementary technique not only to this weakness of testing [Somm15], but also its less resource usage, i.e., state analysis does not require the physical system which could help to save cost and time of the validation process. More details on the formal verification rationale are to be discussed in the following section (cf. Section 2.3). To have a closer look at the current industry practices, although a number of companies apply automated testing, there exist still many of them adopting manual testing or some companies relying solely on the testing at commissioning: 84% and 10% of the participants respectively of the survey presented in [VoOc18]. Also, it was shown in the same research that many companies are testing every specified scenario to obtain the requirement coverage (61%) though the code coverage is comparably low (15%), resulting in low efficiency of the testing.

As an a posteriori quality control mechanism, monitoring takes an important role for safety-critical systems such as aPS [AbRo10] since any malfunction may cause not only damage on the system (itself) or the operating personnel but also the payload that the system handles and that the harm to the user of the resulting product might be inherent potentially. To avoid these critical effects, the main objective of monitoring is identifying both error situations during runtime and unexpected behavior of the technical process or the hardware not covered by the specification. Possible causes for erroneous states of these parts of the system are, for example, wear on the hardware and manipulation of the technical process or the aPS. Malfunctions and their resulting harms could be avoided by faults identified and handled as soon as possible. Today, monitoring functions are often related to a specific piece of hardware and are directly connected to functions that abstract the interface to that component (driver functions). Information about the hardware behavior is tapped from the inputs and outputs of the drivers. In case of undesired hardware behavior, warnings are given to the driver and the rest of the software system. It is of utmost importance that the monitoring functions work as intended and detect all errors and unknown states of the system to allow the system to give appropriate warnings, handle these errors or bring the system into a safe state by performing an emergency stop to prevent further damage.

2.3. Formal verification terms and basics

Formal methods are software engineering methods through the application of mathematics for modeling and analyzing systems utilizing the benefits of its rigor [BaKa08]. As the complexity of the control software and its functionality grows, and the demand for high-quality solutions as well as reduced engineering cost (including time), formal approaches in aPS engineering, or in PLC programming more specifically, are required [FrLi00]. The formalisms within aPS engineering processes have been introduced in the form of specific modeling languages like UML or SysML

for the design phases in the scope of model-based engineering, and later, also for the quality assurance purposes as formal verifications [Vyat13]. As the focus of this thesis is put on the formal specification of the aPS control software, the basics of the formal verification follow below.

2.3.1. A framework for formal verification

System verification is defined as “the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase” according to [IEEE14A] It is often stated with the “written statement,” which is the specification comparable to the “conditions” in the earlier quotation. That means verification is an activity to assure the flawlessness of the target system concerning the specification, meaning the target system behavior conforms to the required conditions, which are reflected in specification.

Such activities mainly aim to obtain dependability, including the quality properties of availability, reliability, and safety of the system [IEEE14B]. The verification result is not meaningful if there exists any ambiguity regarding the requirement interpretation, and, accordingly, system verification activities fall in the range of formal analysis in this sense since the formal descriptions do not allow any ambiguity with rigorous syntaxes and semantics. In addition, formalisms allow the automated tool supporting since mathematical expressions can be easily transformed into a form, in virtue of its formalisms, that computer systems can process.

As a static analysis technique, formal verifications do not require the system execution, and, thus, no plant is needed. They use mathematical calculations to prove the conformance of the system to the given properties. There are two different approaches to verification: one is theorem proving, and the other is model checking [BaKa08]. In theorem proving, the system is described as a set of logic formulas (i.e., system axioms), and so the properties are. Theorem proving is to find proof to obtain the properties as conclusions from the system formulas as premises. In model checking, the system is described as a finite model, and the properties are in logic formulas. Model checking methods compute whether the asserted hypotheses (properties) are valid in any case. In both approaches, it is necessary to have formal descriptions of the system (either as a set of formulas or a formal model) and formal specifications as the properties to be confirmed (Figure 6). Though this thesis focuses on the formal specification part in the formal verification, all the concepts introduced are framed within the model-checking scope since the overall approach has been developed under the model checking approach.

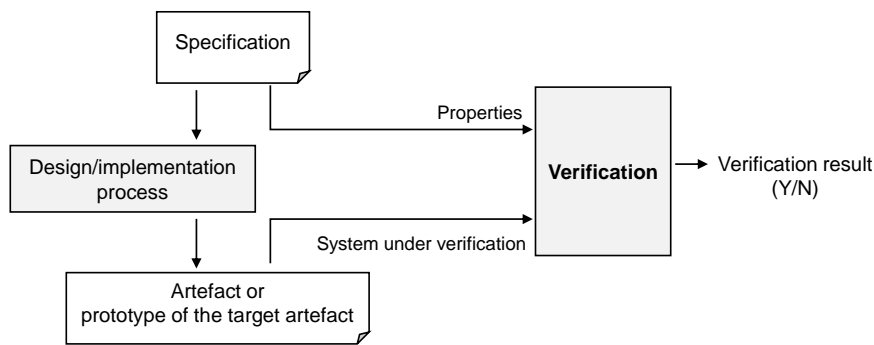


Figure 6: Generic schematic view of the system verification [BaKa08] (reproduced)

The result of the formal verification may appear in three types [HaMM16]: proven (meaning that the given system representation conforms to the specification), a violation with a counterexample, and timeout (meaning that the proof calculation takes a too long time or runs out of memory). If the software is proven as conforming to the specification, it would be the most favorable case from the viewpoint of the user (of the verification) since building a system satisfying the given requirement is the gold of the engineering. Even if the proof fails, it is still meaningful if the verification gives the information of counterexample, which shows the execution path leading the system status to a state that violates the property described as specification, starting from the initial state. The third type, timeout, is the worst case of verification, which is also regarded as one of the barriers to utilize formal verification. As easily seen in many practical cases [BaKa08], the state space could be very large, which leads to the so-called state explosion. This is critical for the approaches using state-space search algorithms, such as model-checking. State variables with a number of values, e.g., integer, or higher number of parallelisms of separate processes, entail states growing exponentially in the program model [BaKa08] and consume extremely large size of memory as well as long computation time [CaHN11].

PLC control software has been also targeted of the formal verification, using model checking, taking predefined software modifications into account [SüVZ13], approached over various IEC 61131-3 languages (e.g., [BEHL04, Huuc05]) investigates by means of the model checker. Coupling simulation and formal verification has also been approached to compensate each other (e.g., [BaKa08]).

2.3.2. Regression verification

In the single program verification, the program is examined with respect to the specification. However, formal verification has not been commonly used yet since it is not realistic to expect whole formal specification of the software when it comes to industrial size [StGo08]. Especially in many cases of the industry practices, not even an informal description of the requirements is available, which could be used as a basis for a formal specification [CWUB18a]. Different from the typical formal verification, regression verification [StGo08] focuses on the comparison of two different

versions of the programs and proves that there is no regression during the implemented change with assuming that the previous version of the program is correct. It executes a process of providing deductive evidence (i.e., a proof) showing that the behavior of one software is in a well-specified relation to the other software behavior. That is, new software revision can be proved to be “as correct as of the previous version” rather than to be “correct” absolutely.

The main advantage of regression verification is that no functional or behavioral specification is required besides the old version software. Actually, this verification technique is suitable and reasonable for the aPS evolution, in which the implementation is usually based on the previous version of the system; and benefits of the advantage could be fully realized. The applicability of this methodology on the aPS control software in PLC code in IEC 61131-3 has been shown in [BUVW15]. In this case, regression verification proves that a new revision of the PLC code shows the same behavior as the old revision, and no unintentional behavior is introduced. Additionally, it was also shown that the size of the specification can be reduced to the difference of the versions by applying regression verification in [UUWK16].

Regression verification can be described as a process to assure that the trust earned by a previous version of the system remains through the evolution steps: whether the software is still trustful (partially or entirely) after the evolution. Formal methods might be suggestive to the application engineers of “correctness,” and the regression verification, which is a sort of formal methods also might. However, the “correctness” of software in the regression verification scope is different from the one in the view of the defined specification but rather more related to the “still-trustfulness” after some changes in the software. In [CUWB19], the obligation of the regression verification in the sense of the trust preservation during aPS evolution was shown. The sources of trust on regression verification of system software are

- A (formal) analysis of the software against properties trusted to be adequate
- A (formal) analysis on a model of the system (i.e., comprising not only the software but also trusted adequate models of electrical and mechanical parts)
- Executing test runs on the systems
- Simulation executions
- Successful (test) operation
- Long-running successful in-service operation.

Also, the transference of the trust over the evolution could be classified depending on the different level of changes as a) a complete change, b) a partial change outside of the observed range, c) a change including observed range partially, d) a change including all observed range, and e) A change including all observed range. This explicit trust inheritance analysis was to clarify for the

engineers to understand the extent of the trust that regression verification provides and consequently facilitates them to utilize this formal technique for system validation.

2.3.3. Formal specification of the requirement

The formal specification is essential constituent in the context of the formal verification, which is described in logics aiming to model the situations designers/developers/testers encounter in such a way that they can reason about them formally [Gora06]. However, its meaning is not limited only to formal verification but rather spread over the entire engineering processes in this thesis. A system is built to satisfy the user's requirements, which are defined in SWEBOK [IEEE14B] as "software requirement is a property that must be exhibited by something in order to solve some problem in the real world," and SEBOK as "that describe optional, functional, or design-related aspects of a system" [IEEE14B]. The required system characteristics are specified in the *specification* regarding what and how the system is supposed to behave and what not [BaKa08]. The informality of a requirement description, i.e., specification, may cause ambiguities, which could cause severe and extensive flaws in the following artifacts eventually [HuRy04]. Also, manual verification of real-world problems is not feasible when it comes to the industrial scale. Considering these concerns, the formal specification is required, being defined including formalisms as in [Lams88] "the expression in some formal language at some level of abstraction of a collection of properties some system should satisfy." Nevertheless, it is still observed that no formally described requirement transmitted over the aPS engineering in practice; even the specification is only done oral description without any document. Almost 12% of participants answered they fall into this case in the survey conducted in [VoOc18] and this appeared even worse in the serial machine builders (this data is not published yet).

Specifications are claimed as essential in [Lams88] for almost all of the engineering phases and activities, namely designing, validation, documenting, communicating, reengineering, and reusing. Huth and Ryan also highlight the reason of specification in [HuRy04] as documentation, decreased time-to-market, reuse, through the clear specifications. Especially, Lamsweerde shows the same view with this thesis on the various utilization of the specification depending on the user of it and the usage, which is "One of the problems with the formal specification is that they may concern different classes of consumers having fairly different background, abstractions and languages - clients, domain experts, users, architects, programmers, and tools."

Within the formal verification, one checks that the system description satisfies the formal specifications in terms of mathematical logic mostly regarding the following properties: a) Functional correctness – the system should do what it is supposed to do, b) Safety – something bad (including deadlock) will not happen, and c) Liveness – something good will happen eventually [BaKa08,

HuRy04]. Formal specifications are described in the formal specification language, defined as “textual languages that use basic notions from mathematics (for example, logic, set, sequence) to rigorously and abstractly define software component interfaces and behavior” [IEEE14B]. Since Pnueli suggested the functional properties of reactive systems in temporal logic [Pnue77], these temporal logic expressions have been widely accepted [CHVB18, EiFi18]. The term ‘temporal logics’ means and is intended to express such that a logic formula is not statically true or false in a model, but its notion of truth might change dynamically depending on the states and the order of events [BaKa08, HuRy04]. Therefore, temporal logics, typically represented by Linear Temporal Logic (LTL) or Computational Tree Logic (CTL), depicts these event occurrence’s orders.

3. Requirements on a Formal Specification of Reactive System's Control Software Behavior

Today's quality assurance in aPS software engineering is done through testing most of the time. For some software code, however, not all the cases are suitable for testing, e.g., break- or tear-down, and also, it is not possible to validate the target code for all the possible test cases most of the time. In practice, many mandatory test cases are rather omitted on purpose, especially after change implementation on the existing code, due to the lack of time. Despite the known power of exhaustiveness of the formal verification, it is not widely used in aPS software engineering, and one of the main reasons is the barrier to the specification. Through the discussion with industry experts, it is learned that the stipulated specification, even in a natural language (informal), often does not exist in the current state of practices (cf. some survey result was shown in Section 2.3.3). To utilize the formal verification, formal specification of the targeted properties is a prerequisite since the verification process requires them as a main subject. Also, aside from the formal verification, formal specification takes an essential role in system validation as an uncontroversial baseline to decide (by a human or by the responsible system automatically) the correct behavior through formalism. Still, engineers in the field like module developers, application engineers, and commissioning personnel have little knowledge or experience in the formal languages [Holz02]. To approach these barriers to achieve formal specifications and ultimately to apply formal verification on the aPS engineering process, requirements are derived below.

Requirements are analyzed from various angles. First, the approach should be feasible to be applied to the technical characteristics of the target system in Section 3.1. Second, the approach should be embeddable into the engineering processes of the system in the context of validation, especially for the software change cases, in Section 3.2. Third, the approach should be usable for the field engineers to be promoted in Section 3.3 for effective effort reduction (i.e., cost and time). All requirements are summarized in Section 3.4.

3.1. Requirement considering target system characteristics

APS are basically reactive systems, which adapt the behavior based on the environment as programmed in the control software. In the viewpoint of the functional hierarchy of automation systems, the layer concerned in the approach includes field devices like sensors (as input) and actuators (as output). The control software refers to input, executes its logic, and results in output to control the target process.

Requirement specification does not have to describe the exact behaviors (or a sequence of them) in detail to specify how operations should consist, but rather have to describe what would be the result of the operations. For example, to describe a function block for multiplying calculation, the requirement specification would describe the result to be the multiplication of the operands as its input, not how the multiplication to be implemented. In the process' subject point of view, the software accepts sensor information (signal) and generates actuator information (signal), which are involved in the information flow. Thus, the developed specification language shall allow presenting the relationship of input and output of the technical system regarding the information flow effectively (R-T1).

The specification information can be classified in either functional description, structural description, or behavioral description as defined in [HSFS17]. The targeted specification shall describe the target software from the functional point of view macroscopically; however, the function is realized by the behaviors representing states and procedures. If the system is non-causal, whose output depends only on the current input, the expected output can be described only using the current input value; however, the calculation is complicated. For a causal system, whose output is decided not only based on the current input but also the past input (or, in other words, the current state of the system), behaviors appear differently depending on the current state and form a sequence of them to realize a specific functionality. Thus, the specification in the developed language shall present the state changes regarding its condition and the results including timing constraints (R-T2). As the target of the specification is not describing specific execution sequence with concrete values but describing states (i.e., transition conditions, results, and timing constraints) and their procedures for general behavior properties, the specification shall present the state in abstracted value ranges (R-T3).

Industrial processes can be classified as continuous (including batch) or discrete. In the continuous process, although the behavior changes over the process, it is not easy to divide into separate states clearly due to its continuous change of the material's state. Therefore, stopping the process, returning the system status back to a certain status, and resuming it cannot be simply done. Rather, the process should be handled as a whole. Contrastively, a discrete process can be separated into apparently divided behavioral steps, recognizing exact input and generating corresponding output. Since it is targeted to verify the state-based behavior regarding the input and output signals, the target of the specification shall be the discrete event processes (R-T4).

The control software of the targeted technical system is loaded and executed in PLC. Instead of control circuits consisting of relays, switches, and clocks, PLC allows programmable logic and input/output interfaces [Hans15]. PLC operates in a cyclic-manner in which a procedure to execute

the control code repeats at every certain time. Each cycle consists of scanning values (input), execution, and updating values (output), so the unit of the input recognition and output change cycles (cf. Section 2.1.2). Thus, the specification in the developed specification language shall be capable of the cyclic execution representation (R-T5). Also, IEC 61131-3 is the standard language to program the PLC software and is applied throughout the production automation industries [VFST15]. Thus, the specification language shall be compatible with the software in IEC 61131-3 as well as its execution (R-T6). In this way, common library functions do not have to be described regarding their detailed behavior when they are used within the description, but rather could be used just as they are (as assumed that no fault exists in the library functions); using the common library functions with the description, therefore, will provide both description efficiency and readability.

3.2. Requirements considering engineering processes of aPS

A technical system realizing its corresponding technical process is developed throughout aPS engineering processes. As a designed-to-order system and also for serial machines (which means a number of machines in a type), the aPS engineering process goes through specific engineering methods depending on the own requirements and objectives of the system or the type of the system. Nevertheless, a general engineering life cycle can be defined as described in Section 2.2 (Figure 5). Over the engineering steps, system requirement specification is to be generated and used. In this section, the requirements regarding the engineering activities related to the requirement specification are described.

Control software is more promoted to consist of module libraries to speed up the development and validation [VFST15], developed by module developers and integrated by the application engineers into an application project [VFND18]. When a module is developed for the first time for new behavior, a behavior specification grounded on the customer requirement would be delivered to module developers or created by them interpreting the requirement intention to develop the software [VFND18]. Then, the module is developed in the company's convention programming language, and an application engineer is supposed to build up the desired project out of the modules inserting glue codes [KBSK10].

The developed software module is to be validated. The correctness of the behavior is determined based on the previously defined specification regarding how it is supposed to behave to the expected input sequences by testing and verification. A formal specification language is required over the aPS engineering processes to describe requirement as the most important prerequisite of all engineering activities (R-E1). That is, the characteristics of the expected output sequences are

defined according to the input (or input sequence) sorts. Thus, the specification shall allow describing the input/output trace characteristics of modules, and this also holds in the engineering point of view. In this way, the changed part or behavior, which is mainly aimed in this work, can be described. As a prevalent validation method, applicability to the testing would also be beneficial to the current practices right away. In the form of model-based testing, the test cases shall be instantiated easily from the specification (R-E2). For sure, the specification in the developed formal approach would be a form that a model checker can interpret and process since the approach also targets the specification to apply formal verification not only to document requirements (cf. [Weig21]). Through the verification, conformance of the specification is checked. In case of any proof failure, a counterexample exists, and this could be fed back to the engineers for further debugging. Therefore, the approach shall provide a way for automation engineers to use the verification result effectively for further debugging (R-E3).

The completed developed project is delivered to the real production environment and executed for the test run during the commissioning stage by the on-site start-up technician [VFFU17]. The personnel (i.e., commissioner) adjusts system parts (hardware and software) to accustom the integrated plant to the best fitting condition. In the meantime, some parts of the software are discovered to be revised or reconfigured. Although the best way to make any change on the software would be analyzing the inappropriate behavior, re-designing, re-implementing, and validating by the development department, the commissioning stage often lacks of time so this entire re-engineering process is not possible to be conducted [Voge09]. An optimized and ideal way to solve it is that the start-up technician implements the change, validate it regarding the correctness of the behavior and side-effect freeness as much as possible, and document it delivering to the corresponding office (then the further proof would be done within the development department). The same requirement as R-E2 and R-E3 applies here for the validation process. However, this organized procedure is still lacking in practices, appearing as just individual development forms; thus, it threatens the following maintenance and update efficiency by generating unauthorized variants and versions [VFFU17].

During the operation, significant signals within the system are supposed to be monitored to detect unexpected behaviors or unconsidered conditions during the requirement specification stage. Monitoring functions are often related to a specific piece of hardware and are directly connected to functions to obtain information about the hardware behavior by tapping from inputs and outputs, and to invoke alarms if necessary. In case of undesired hardware behavior, warnings are given to the system. It is of utmost importance that the monitoring functions work as intended and detect all errors and unknown states of the system to allow the system to give appropriate warnings,

handle these errors or bring the system into a safe state by performing an emergency stop to prevent further damage. As the monitoring block conducts a role of dynamic validation, the most efficient way is to generate the code directly and automatically from the specification to conduct the monitoring function. Thus, the specification shall be in the suitable form to generate the monitoring block, and the generation method is to be developed (R-E4).

Over the aPS' lifecycle, changes are caused by various reasons unavoidably [VFST15], e.g., to fix some bugs, to implement additional functionality, or to come up with some technology trend. Thus, the aPS control software often has to be adapted to the changed requirements for these reasons. The change has to be carefully implemented not only to realize the intended change effect but also not to introduce any side effect. Over this change implementation activity in the maintenance stage including functionality updates, debugging, and behavior improvement, intended change specification would be conducted and these shall also be supported by the specification and validation activities similarly. What is different from the initial development process is that not the entire part of the system is to be checked, but the change-related part is the scope of the validation, which is especially effective for small changes. Nevertheless, the unchanged part of the system also should be validated regarding the side effect caused by the change. Therefore, the changed part should be described clearly and formally at best to apply validations methods systematically, e.g., test case generation, model validation, or formal verification (R-E5). Overall, documenting the software behavior with various objectives and utilizing them should be easily embedded as a method of each engineering process and different users by exchanging them (R-E6, cf. Figure 7).

3.3. Requirements considering the users' point of view

For all descriptions necessary over the engineering processes so far, other specification forms or languages, e.g., a path-time diagram, might work theoretically since it can describe the multiple signals, their value sequences, and synchronization of them. Although it is a very intuitive graphical representation of signals, it is effective with a fewer number of signals and time sequences to read it at a glance. Actually, one of the machine and plant manufacturing company expressed to reject timing diagram explicitly in an interview due to its scalability and also its additional notations for automation engineers to learn and familiar with, but proposed test tables as a more preferable and already familiarized means instead. When the value ranges are wider, or for some aggregately representable values, graphical notation has some limitations. Also, as a graphical notation, it is not easy to export and import the information systematically and automatically. Therefore, the approach shall support the systematic handling of the information with structured data representation from usable practices and, correspondingly, a software tool shall enable various users to utilize the approach (R-U1). This premises to ease the applications of the approach by

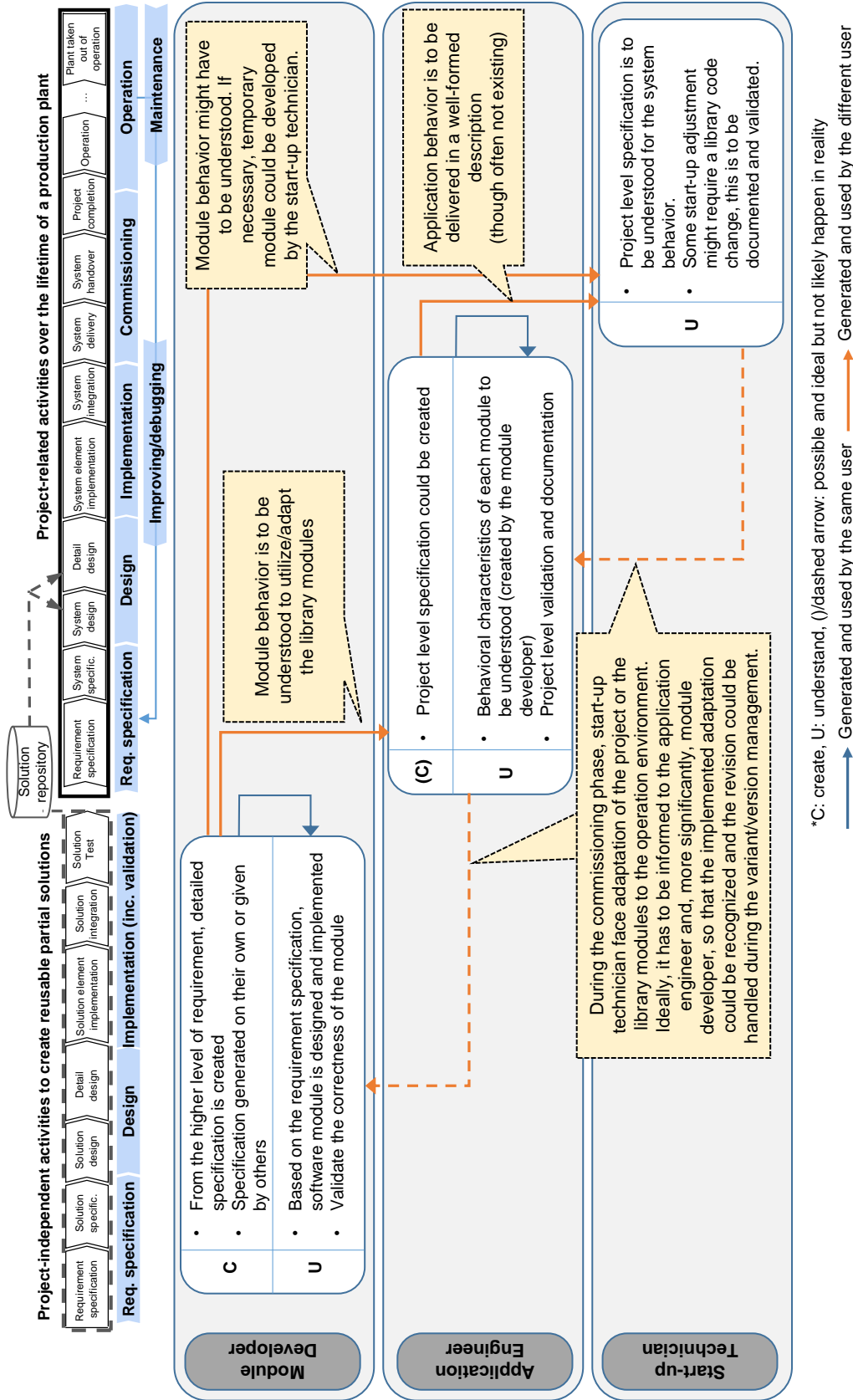


Figure 7: Activities and users of the specification over the engineering process – depicting the information exchange (arrows)

various type of users (i.e., module developers, application engineers, as well as the commissioning/start-up personnel, as stated).

Thus, the approach shall be accessible regarding its usage (R-U2), namely to be understandable (R-U2), to enable to create (R-U3), and to be learnable (R-U4). Considering the applicability to the industrial cases, scalability is another factor to be evaluated (R-U5). Ultimately, the developed method is expected to satisfy the user of the approach so that they would like to use it for the intended purpose with minimum resistance (R-U6).

3.4. Summary of requirement

In short, the overall requirement could be summarized as “the specification language shall enable automation engineers to express the requirement and the behavior of the system in a formal way without knowing the conventional formal logic to specify and understand small changes.” Detailed requirements and its description is given in Table 2.

Table 2: Summary of the requirements

Requirement ID	Requirement description
R-T1	The developed specification language shall allow presenting the relationship of input and output of the technical system regarding the information flow effectively
R-T2	The specification in the developed language shall present the state changes regarding its condition and the results, including timing constraints.
R-T3	The specification shall present the state with the abstracted value range.
R-T4	The target of the specification shall be the discrete event processes.
R-T5	The specification in the developed language shall allow the cyclic execution representation.
R-T6	The specification language shall be compatible of the software in IEC 61131-3 as well as its execution.
R-E1	Formal specification language to be used over the aPS engineering process shall be developed.
R-E2	The test cases shall be instantiated from the specification.
R-E3	The approach shall support for implementation debugging with respect to the specification.
R-E4	The specification shall be a suitable form to generate the monitoring block.
R-E5	The changed part should be described clearly and formally at best to apply validations methods systematically.
R-E6	Documenting the software behavior with various objectives and utilizing them should be easily embedded as a method of each engineering process and different users by exchanging them.
R-U1	The approach shall support the systematic handling of the information through a tool with structured data representation.
R-U2	The approach shall be understandable, meaning the typical automation engineers can understand (after reasonable training) by reading the specification in GTT.
R-U3	The approach shall support creation of specification, meaning the typical automation engineers can create the specification in GTT (after reasonable training).
R-U4	The approach shall be learnable, meaning that it requires reasonable time and effort from the typical automation engineers to utilize the language regarding reading and creating activities.
R-U5	The approach shall be usable for the scaled behavior description.
R-U6	The approach shall be satisfactory, meaning that the users of it (i.e., automation engineers) would be willing to use it.

4. State of the art

There have been approaches introduced that could be applied to satisfy the requirements presented in the previous chapter. To give an overview, specification approaches that are targeted to be applied over the aPS control software engineering processes are discussed in this chapter. Not only the basic characteristics of the specification approach targeting technical aspects of the aPS (T1 – T6) but also the engineering viewpoint (E1 – E6) and usability aspects (U1 – U6) are discussed (Section 4.1). As one of the major concepts of the thesis, monitoring techniques which are not connected to the formal specification approach is separately reviewed (Section 4.2). The usability aspect focused in Section 4.1 is more general; as one other primary requirement of the approach introduced in this thesis is usability to the automation engineers, research and evaluation approaches concerning this point are discussed (Section 4.3). The conclusive research gap is derived at the end of this chapter.

4.1. Formal specifications behavior: language and utilization

Formal modeling and verification techniques are originally created in the pure computer science field, and aPS behavior verification is a comparably recent research topic [HLLT06]. Since initiated by the early works such as [MPBC92], verifying PLC software has been actively researched (e.g., up to [ShVy20]), motivated by the fact that software engineering techniques in automation engineering today are not sufficient to thoroughly assure the required level of quality efficiently along with the system operation duration with shortened evolution cycles. Formal methods provide proofs that the implemented steps are correctly conducted with respect to the specification [Berg82]. Behavioral specifications define how the system is supposed to process, i.e., requirements, and the system model is verified regarding its conformance to the specification in the formal verification. In this section, formal specification languages introduced for or utilized in automation system engineering are reviewed and compared with respect to the requirements presented in the previous chapter (Chapter 3). Note that mature languages with more utilization methods with respect to the requirements were selected here among various newly introduced languages. Namely, test case generation, specification inferencing, and monitoring/debugging approaches will be focused for each formal specification language.

4.1.1. Text based temporal logics

Pnueli suggested the functional properties of reactive systems in temporal logic [Pnue77] as almost the earliest one, and these temporal logic expressions have been widely accepted [CHVB18][EiFi18] by providing the expressions for sequences or order (causality) of the states

in a system [PiPn18] and relations over time in reactive systems [LÅ FE14]. Consisting of the symbols Boolean operators, temporal operators, and path quantifiers, the sequencing of the system states along the path is defined [HuRy04]. LTL and CTL are common temporal logics so far among other temporal logics [LÅ FE14]. LTL, introduced in [Pnue77], frames the property in the linear-times view, as the name indicates, “in the sense that at every moment in time there is only one possible future”; thus, other possible executions are treated as independent sequences, as an excerpt from and explained in [PiPn18]. The branching-time framework was considered deriving CTL, introduced in [EmCl82] and [QuSi82], accepting that there may be multiple possible future options from a state. Although the third type of temporal logic, CTL*, was invented as a logic combining LTL and CTL [EmHa86], it is not as common as LTL or CTL since many model checking tools do not support it yet [BBFL13, Rozi11]. Added with timing concept, they have been extended to Timed Adaptive-LTL (TA-LTL) [ZJMC00], LTLt [KrPA03], and Timed-CTL [AIDi92].

As additional feasibility targeting embedded or automation systems, time constraints limit the state duration or transition with the timed notation in the viewpoint of state changes concerning corresponding input and output; the state transition is not clearly seen regarding whether the specification describes the mandatory state changes of the system. Still, LTL and CTL have been often applied to describe the behavior of IEC 61131-3 for the formal verification as presented exemplarily in [HBZY19] or [VFAM15]. There are some researches such as [CCDM05] or [PiQu13] for supporting run-time monitoring, but it has not been actively researched further. Test case generation has also been approached as seen in [ZhZK07] or [AHDR18], deriving possible test cases from the specification; however, many are mostly based on the model checker result, not directly from the specification itself. That is, the counterexample, which the model checker results in when it fails the proof, indicates a signal sequence that does not satisfies the target system. The test cases generated from these counterexamples are limited to the case that they represent, but cannot cover generally what the original specification intended.

Initiated from these basic languages, there have been introduced many approaches focusing on the specification languages targeting industrial systems. Property Specification Language (PSL) [IEEE10] developed as a standardized specification logics to reduce the incompatibility of the different temporal logics and data exchange [TuSc05], targeted to be compatible circuit design languages. As an extension of LTL, it follows most of the characteristics, adding various practical and standardized features, such as cycle representation, to improve the readability [PPBV16]. In [OdMB06], Test vectors are approached to be generated from the properties in PSL, targeting to generate the input of the micro-control circuit (namely, VHDL). Instead of explicit supports for

the debugging of the implemented artifact (model or code), debug pattern specification language was introduced in [GhFu09] to match the bug to the defined patterns.

ASTRAL [GhKe91] uses a state machine process specification with state variables and transitions. Specific transition timing is also specified within the specification when the state starts or ends. Targeting the real-time software requirement specification, it was applied to the embedded system analysis [BBKL98], putting the focus more on the interfacing and communication of the processes. The ASTRAL Software Development Environment provides design and analysis tool support for the specification. Specifying the transitions in detail with precise timing annotations might work as the documentation of the function block specification documentation though the specification is still fully in not-structured texts. Research about its adaptation or application to support specification inferencing was not found.

LUSTRE [CPHP87] is a dataflow specification language, which regards programs as simple functions that process the input to obtain the output with loops that are unfolded and assumptions of no circular dependencies among variables. Therefore, usually, an output variable is presented as an algebra equation over input and local variables [CGKT16]. It is known that this simplicity provides easy programming and debugging environment. LUSTRE has been approached to be used for IEC 61131-3 software mainly done by Kabra (e.g., [KBKW12]). Formalization in LUSTRE of SFC programs to apply model checking, similar to the presented method in this thesis; however, it was to have a rigorous model for SFC program as a model to verify, not as a specification. Test case generation from LUSTRE has been actively researched. Test objective was guided by LUSTRE property in [RNHW98], and a method to build Boolean test input constrained by LUSTRE property was presented [BORZ99], both provided with tooling. Covering the limitation of testing invariants of these approaches, [MaAr00] presented a method and a tool to derive a test sequence from leading the test to an arbitrary state. The tool has been stabilized with SCADE [Ansy21] in the industry. Still, the inconvenience of such dataflow style description is regarded as to improve further [Halb05] to be used as readable and accessible documentations. As an executable specification, efficient code generation was shown to be feasible through the construct of the finite automata in [CPHP87]. CoCoSPEC [CGKT16] extends LUSTRE by adding constructs to specify contracts for individual nodes (which are similar to states in the context of input/output pair), as LUSTRE comments. Authors of [CGKT16] claimed that a mode-specific requirement is necessary (similar to [MaAr00]) and introduced the mode-awareness manner of the LUSTRE description.

4.1.2. Graphical and signal pattern based languages

Specification patterns were introduced in [DwAC99] with the motivation of accessibility from practitioners so that they can define similar requirements instantiating the patterns. Commonly occurring requirements in the abstracted description were introduced as property specification patterns in the finite-state model, and the pattern must hold to the extent of the program execution. The authors also showed that the majority of the used examples (92% of 500 examples) matched to the presented patterns. The pattern-based specification has been developed in further researches such as [GrLa06], [KoCh05], or [SACO02], attracted by lower barriers to access to the formalisms [GrLa06]. However, the presented approach was claimed recently as hard to directly apply when it comes to the industrial control domain [PPBV16]. Nevertheless, code synthesis approaches and tools have been developed as presented in [DDHM02] for pure software systems. Very few approaches were found for test case generation from the specification pattern except that extracting finite-state machine was considered as abstract test case generation indirectly in [NgMT12]. For specification inferencing, patterns are useful to handle the non-decidable behaviors by restricting the possible behaviors within patterns (e.g., [GaSu10]).

Both Live Sequence Chart (LSC) and Property Sequence Chart (PSC) are based on the UML statechart diagram and interaction sequence diagram, respectively, which are UML behavioral diagrams. LSC [DaHa99] was presented aiming at the description of liveness. Based on the state sequence on a lifeline for each component, interactions between them, including message transmissions, dependencies, cause, and effect are represented in the graphical representation. Many features of LSC is adopted within the UML 2.0 interaction sequence diagram [AuIP07], and Property Sequence Chart (PSC) [AuIP07] was developed to cover and extend the features of LSC in UML 2.0. Thus, PSC also targets to describes the cause-effect relationships reusing the concept of ‘chain’ (introduced in [DwAC99]), which indicates the message sequences. Although the specification is not used directly in formal verification, some translation approaches are suggested to obtain the formal specifications in automata, into LTL [KuMB09] or CTL*[KHPL05]. Based on the clear and intuitive behavior expectation from the specification, the monitoring method based on the specification [ZLWG11] and its tool [ZSZL10] followed the primitive approach. Lo et al. presented the method to mine the LSC specification from the execution trace in [LoMK07]. Both are more suitable languages to describe the interactions but not a stand-alone component to be specified. In addition, researches about supporting debugging through those languages were hardly found.

Petri nets (PN) are one other well known graphical and analytical specification language to be used for formal specification or model with a firm mathematical foundation [HuKi11]. The desired system is characterized as a structured process in this executable type specification languages

[Hoar78], including PN. Theoretical results concerning PN are plentiful being extended to be applied to a wide variety of systems, also depending on the analysis objectives, e.g., concurrencies, synchronization, or probability models (cf. [Mura89] for PN overview); and there the Condition/Event type PN is introduced as a most typical for automation system engineering and comparable form to the presented method. A PN comprises places (representing locations where objects await processing or the conditions that objects are in), transitions (representing processes or events), arcs (representing path or process evolving from place to transition or from transition to place), and tokens (representing the resources or readiness of the places to fire the transition) [MoGu96]. Satisfying the requirements of discrete event system modeling, it has been actively used in manufacturing system engineering [GiDi93]. Inspired by PN, GRAFCET was proposed and adopted as a standard to represent specifications for software control systems accepted as IEC (IEC 60848) [GiDi93]. Steps, transitions, and links in GRAFCET correspond to places, transitions, and arcs in PN, respectively. As similar to the PN, various formalisms and applications are being researched actively (e.g., [JTFN19]), benefiting from its graphical representations. The major difference between these two languages is: GRAFCET was intended to specify how the system processes input unambiguously to reach the output, while nondeterminism is allowed in PN based on the nature that the language was developed to describe the system [GiDi93]. Mature research on both languages include code generation (exemplarily for IEC 61131-3 as [Frey00, JSSF17, MuGM05, QaTP17]), test case generation ([SPMK15]) and also tooling (e.g., [WKMS19] or [GoMH16]) while researches for the specification mining (e.g., [LHFL15]) and usability study are rather less found comparably (e.g., partially [SaLo05]).

4.1.3. Specification approaches for automation systems

The approaches so far (summarized in Table 3) could be regarded as originated from and targeted to a more general-purpose or informatics point of view, and these have been reviewed with respect to the derived requirements. As the focus of this thesis is put on the property specification of the automation systems, the target and taste could be scoped down to the more specific aPS boundary conditions.

First of all, the targeted systems experience a lot of changes over the longer lifecycles up to several decades, and the target method has to be compatible to handle it properly. Changes appear in different forms and objectives for different reasons. Typical changes considered are fixing the detected bugs, for example, if any faulty behavior is detected due to the incorrectly implemented control software. Coping with the changed or new requirement is another cause of control software changes. Experiencing changing and growing customer requirements and demands, adding or fixing functionalities are required.

Table 3: Formal specification approaches with respect to the requirements (See Table 2 for each requirement description)

Approaches	R-T1	R-T2	R-T3	R-T4	R-T5	R-T6	R-E1	R-E2	R-E3	R-E4	R-E5	R-E6	R-U1	R-U2 – U6
LTL _[Pnue77] CTL _[EmC182]	0	0	+	+	[ZJMC00] [KrPA03] [AIDi92] [AIDi92]	[HBZY19] [VFAM15]	0	0 [ZhzK07] [AHDR18] [JSCL14]	0*	+	[BGFF12]	-	+	+
ST-LTL _[LAFY10]	+	+	+	+	+	+	+	+	0	-	0	-	-	0
PSL [IEEE10]	0	0	+	+	+	-	0	+	[GhFu09]	+	+	-	[EiFP09]	0
ASTRL [GhKe91]	+	+	+	+	+	-	0	0	-	-	0	0	+	-
LUSTRE [CPHP87] CoCoSPEC [CGKT16]	+	+	+	+	+	0	+	[RNHW98] [MaAr00]	+	+	[KBKW12]	0	+	[Ansy21] [HCRP91] [PGLN20]
Pattern based [DwAC99] _i , [Grla06], [KoCh05] _i , [SACO02]	0	+	+	+	+	+	-	0 [NgMT12]	+	+	[DDHM02] [GaSu10]	-	+	+
LSC, PSC [DaHa99] _i , [AuIP07]	+	0	-	+	-	-	-	-	-	+	[LoMK07]	-	+	+
PN [Mura89] GRAFCEIT [GiDi93]	0	+	+	+	+	[Frey00] [JSSF17] [MuGM05] [QaTP17]	+	+	+	+	+	0	0	0 [SaLo05]

+: the approach fulfils the requirement, 0: the approach fulfils the requirement partially, -: the approach does not fulfill the requirement

Moreover, aPS are typically unique systems, which are implemented on the basis of customer requirements, also taking into account the boundary conditions at the customer's premises [FBSS18] often with a great number of variants not only in plants tailored to specific customers but even in case of serial machines which also often need to be adapted to their specific purpose, e.g., to the properties of the used production material [FBSS18]. Another specific change characteristic of aPS is it also appears in the stage of system commissioning. Even if the system has been gone through the validation phases during its development, some last adaptations for a complete operation combined with another part of the machine or plant is usually necessary. During this commissioning, control software adjustment is many times unavoidable, as adapting the software is easier than exchanging the hardware, and the quality still needs to be assured after having performed such adjustments despite the high time-pressure during this phase and. The changes stated so far are deeply related to the quality of the control software. Additionally, aPS are specifically required to be reliable not only regarding their productivity but also regarding the business sector they are developed for. Different laws and standards need to be considered, especially as a wide range of aPS is applied to safety-critical business sectors like medical or pharmaceutical. Systems in this sector have to be handled strictly according to the laws and corresponding guidelines (e.g., quality management in medical applications (MedTech) [ISO16] or hygiene regulations in the food and beverages sector); and this means the approach also has to be compatible to this environment satisfying the constraints required. Easing engineering effort is another point to be satisfied by the approach since an increasing proportion of system functionality is implemented by software [Thra10], and companies need to reduce their time-to-market to stay competitive on a global scale. Repeated change implementations and validation activities are time and cost consuming tasks, and, thus, appropriate techniques such as transformation of the specification to the executable control software, or vice versa, to lower the engineering cost in this context is mandatory to the approach development. Furthermore, an aspect to be considered for all the characteristics mentioned so far is that there are various stakeholders involved who need to handle and understand the control software [BVFS19]. While module developers are in charge of implementing and validating the library modules intending for reuse in the earlier development stage, application engineers organize and connect these library modules to generate the customer-specific software project. Subsequently, developed and validated system is handed over to the start-up technicians for the commissioning, who are under high time pressure to interface various machine parts, adjust them for the best fit for the operation, and, in the end, make the whole system to operate. Finally, maintenance and operation personnel need to understand the control software in order to be able to identify and react to faults during the system's operation phase to keep the system downtime at a

minimum, which sometimes requires changes of the control software. All these involved stakeholders, the situation they are in, and their educational background as well as specific goals or tasks are to be targeted and considered throughout the approach.

PN and GRAFCET have been actively approached for the manufacturing system engineering enabling the description of discrete event system modeling [GiDi93]. A modeling scheme of control logics using PN was presented in the model level in [PaTK01] but not the PLC control software level yet. The metamodel of the GRAFCET was considered as a modeling tool for the logic control design and analyzed to be transformed to the control code in IEC 61131-3 [ScSF13]. Focusing on the absence of formal models of production plants, derivation of the PN model from the signal traces [LHFL15] to ease the engineering processes [VDFJ14], and this has been extended to the code generation of control software from the formal models [JSSF17] succeeded from [Frey00, MuGM05]. GRAFCET metamodel was also developed for the graphical editor tooling to improve the utilization of the language by the control technicians [JTFN19].

UML state-chart was approached to be integrated with IEC 61131-3 through the bi-directional mapping targeting the model checking of the control code [WiVo11]; in this work, actual signal based behavior representation was introduced, namely “PLC-statechart,” and additional notation was suggested to provide a cyclic execution formalism for the model checker. In a similar way, the UML sequence diagram was tailored for the executable test case description, and further test case code could be generated through defined code generation semantics [KoTV12].

Timing diagrams have also been a way of information representation using known low access barriers in the context of expertise. TimeLine Editor [SmHE01] was presented to substitute LTL formula writing to describe a preamble type requirement (which is a pattern expected to be matched) in a telephone system and corresponding responses, meaning similar to the ‘chain’ as previously introduced for PSC. This focus supported well for specific properties but not generally, as shown through an example of partial ordering specification in [AuIP07]. A conversion method for test automata was also presented, but still, the targeted specification type is limited. One other aspect highlighted in [PPBV16] is that the ordering of the events could be defined, but actual timing is not addressed; this can also be a drawback in the viewpoint of aPS engineering to define signal requirements at the exact timing or cycles. Another approach using a timing diagram in the automation field was presented by Vyatkin and Bouzon in [VyBo08]. Different from the previous one, this approach separates the condition part (“if”) and the conjecture part (“then”), allowing the CTL properties in graphical representation. Its possibility to be used for (automated) testing was stated in [BoVH05], although further approaches were not found regarding the other applications such as monitoring and debugging method. The timing diagram was also approached with respect

to the automated fault injection [Rösc16, RTSV14], improving the state representation of the continuous value change through the lifeline from the conventional timing diagram. From the possible and expected fault injection, the work was improved to generate the test cases and corresponding tooling adapting the script into CODESYS [RöVo17].

ST-LTL [LAFY10] was developed specifically for IEC 61131-3 ST language based on LTL to support automation engineers who are familiar with this language. Invariant descriptions are targeted, which would apply to all possible sequences of input. The notations are aligned to the syntax of IEC 61131-3 ST and the typical signal characteristics treated in the language (e.g., rising/falling edges), including the temporal operators given in LTL. Although it was aimed to be more accessible to the formal specification from the automation engineers' point of view, what was shown in the empirical study (cf. [LÅ FE14]) was that most of the prepared properties were able to be represented in ST-LTL, and improved support was observed for specific types of the functionalities and signals. Engineers still required the representation of the specification in their wordings to use ST-LTL, as also pointed out in [DaMV15]. For this specific language, no research approach was found for code generation or specification inferencing.

One of the important points of describing and modeling variability aPS is reuse [VMKL15]. Relying on the FOCUS theory [BrSt01], which is an interface based component description, behaviors are described for each component in the viewpoint of interfaces, and these are applied to the physical interface and connection to capture the multidisciplinary characteristics [LMCH14]. This component-based system specification is interconnected with the separately described process specification [HCLM14]. This work has been extended to include availability metrics in the probabilistic specification in [MJBC17] to analyze the system availability analysis based on the component faulty probabilities.

Darvas et al. introduced PLCspecif [DaBM15], a specification scheme also to support a convenient way for PLC programmers. In the approach, several (semi-)formal methods, including truth tables, state machines, and data flow diagrams [DaMB16], were combined to specify various aspects like state changes or output definitions. As it includes a detailed description of the various aspect of input, output, behavior state change logic (called as 'core logic' within PLCspecif), it seems to be beneficial to use it as a way to do documentation. Although the integrated specification could provide broader information about the target system (or component), the part that required to be formal specification to be used within the formal verification is limited to the part that adopts originally formal specification. In this context, it was not clearly shown that how the specification approach tackle the accessibility of the formal specification from the user's point of view. As an example, authors presented how the invariants are checked in [DaMV17] using PLCspecif; the

invariants were assumed to be described in CTL or LTL. Code generation scheme was also presented in the same work and more in detail in [DaVM16], and this was based on generation of automata from state machine (described in the core logic) considering the input/output definition.

4.2. Monitoring of aPS execution

Testing is often executed limitedly, most of the time during the implementation stage disregarding the long-term behavior of the system. To complete this weakness, runtime monitoring approaches have been proposed [DoHF14]. Kustarev et al. [KBMA15] proposed a PLC behavior monitoring approach at runtime by installing an additional monitoring controller to preserve and analyze the observation results. This controller monitors actual input and output pairs of the PLC; i.e., it performs an overall system behavior monitoring. Steinegger et al. [SMZS16] presented an approach of automatic generation of diagnostic code mainly targeted at monitoring communication interfaces. Using these methods, monitoring code is generated from additionally specified models, which requires additional effort often not available in aPS engineering. Ladiges et al. [LaFL16] gave a good overview regarding the model-based diagnosis during the operation targeting production plants. They classify the diagnosis approaches into a classical manner, i.e., supervision through fault detection and isolation, and model learning techniques for the automata to capture the system behavior. [KKST13] introduced an integrated development environment (IDE) for the rapid development of a monitoring program supporting EtherCAT, the popular real-time Ethernet communication profile for PLCopen standards.

While many of the monitoring approaches are focusing on widening monitoring targets and easing the monitoring block generation, another stream appeared within the underlying idea that monitoring the conformance of the formal requirements [DoHF14]. Scoping down to the PLC software, possible signal behavior types are formally defined, and corresponding models are used as monitoring criteria during runtime for the conformance check [WeZB15]. Signal traces based system characteristic inferencing technique was integrated with the formal interface description to check the conformance of the physical signals to the specified characteristics [HLLF14]. In the meantime, the runtime enforcement monitoring paradigm [FMFR11] has been applied to PLC monitoring. That is an extended version of monitoring: similar to conventional monitoring, monitors observe current execution status but, additionally, halt the execution when it deviates from a specific property. This could be regarded as a kind of fault avoidance activity in the sense that the monitor detects abnormal executions and actively prevent further execution before it gets to the faulty states by intervening. In [LaMM20] targeting cyber-physical attack monitoring on the PLC networks and prevent potentially corrupted sensor signals and dropped actuator signals by modifying

and inserting actions while [Mcl13] approached to mediate the signals transmitted from PLC to the plant by concealing actuator commands.

4.3. Usability studies of specification approaches

Usability studies of the approaches used within the aPS engineering processes have been another major topic to consider transferring the knowledge to the field, although not all the approach researches do not count it as mandatory. In this section, usability studies done on the specification approaches will be introduced.

Domain-specific languages provide concepts, notations elaborated for a specific purpose, and higher productivity in system engineering [DeK102]. Since model-based engineering methodologies are more pervasive, metamodels as well as these types of languages, appear in various forms, conforming to the satisfaction of goals and needs. Focusing on aims often hinders the actual usage aspects, and then the language is only left in theory. There are studies regarding the usability of modeling languages that are usually used to design the system. Since our approach is aimed at requirements of formal specification languages, some works that focus on the evaluation of requirement/specification description languages are stated here.

Teruel et al. [TNLM14] present a brief result of usability evaluation in regard to a requirement modeling language showing a completeness rate and the result of a user satisfaction analysis. Snook and Harrison [SnHa04] compare a formal specification language, namely Z, with the implementation of Java in regard to comprehensibility and show that there is little difference. Carew et al. [CaEB05] experiment with the acceptance level of formal and informal specification and training time for formal language. Razali et al. [RSPG07] present the usability experiment result of the combination of semi-formal and formal language in regard to comprehensibility and preferences and conclude that the combination is useful in promoting specification. Timing diagram based language (e.g., [VyBo08]) implies usability of the conventional timing-diagram although it turned out to be not so practical in the field to be applied due to its limitation of scaling (this was figured out through an interview with one of the major packaging machine provider). Similarly, pattern-based specifications (e.g., [Bits01, CaMa09]) might be usable since they have been initiated by categorizing existing specifications and extracting the patterns. Looking at this aspect, Pakonen et al. [PPBV16] review accessibility of some specification languages in regard to coverage of predefined formal properties as effectiveness evaluation in terms of ISO standards. As highlighted in [SPGV19], formal methods are required to be evaluated empirically in industrial scenarios to be applied in the practices. Also, even though the developed or extended language is

regarded to entail a certain level of usability implicitly inherited from the origin, the usability of the developed language should be re-evaluated in regard to formation as well as objective changes.

4.4. PLC software programming tools

There are a variety of PLC manufacturers who provide controllers, such as Siemens, Beckhoff, Schneider, Festo, or ifm electronic, and some are providing their own PLC control software programming environment. On the other hand, there also exist several PLC control software IDEs providing compatibility to many different types of controllers and components. As general-purpose IDEs, MULTIPROG (Phoenix Contact Software GmbH, earlier KW Software), OpenPCS (Infoteam Software AG), and CODESYS Development System (abbreviated as CODESYS, from CODESYS GmbH) could be counted; these IEC 61131-3 IDEs enable to develop the control software for the PLCs, where the software is to be deployed and run, and these are targeting different aspects like support for the development platform (Operating Systems), compatible PLCs, simulation functionality, or IDE customization. Among these, CODESYS provides broad support of devices as well as extensive expandability through libraries and add-on components. Furthermore, a large number of manufacturer-specific PLC IDEs are based on CODESYS [KeMa19, RBKP13], for example, IndraLogic (Bosch Rexroth), TwinCAT (Beckhoff), EcoStruxure™ Machine Expert (Schneider Electric), or e!COCKPIT (WAGO) but do not offer all of its features and have additional functionality implemented, tailored to their customer base. In detail, it provides all five IEC 61131-3 for a great number of devices worldwide, mostly for programmable automation components within more than 400 industrial companies [Code20a]. CODESYS also provides various add-ons, and one of these is CODESYS Test Manager [Code20b] to support the automated testing. Defined test cases can be converted into executable test cases within the control software project and the corresponding test report is provided after the test execution [Ulew18]. Within this testing environment, table format is embedded to represent the test case variants, and the users, who are automation engineers, are already familiar with a table-based notation to specify test cases [UIVo15] (cf. the interview comments shown in Section 3.3).

4.5. Discussion of the research gap

Approaches introduced so far satisfied the requirements derived in the previous chapter partially, and no research has been found to fulfill all the imposed requirements. In detail, the specification approaches introduced in Section 4.1.1 and 4.1.2 are more general approaches, while the ones from Section 4.1.3 are more scoped down to the aPS engineering domain. Approaches for monitoring to capture the runtime errors are developed but not systematically related to the specification approached sufficiently to maximize its utilization (cf. Section 4.2); usability of such specification

approaches has to be supported by appropriate evaluations (cf. Section 4.3). Furthermore, it is also important to consider that many automation engineers developing IEC 61131-3 are already familiar with a specific form of developing and testing (cf. Section 4.4). Conclusively, an approach for change descriptions for control software validation purposes to be used over the engineering processes, considering the current development environment, is still missing. Also, formal verification and specification have not become a familiar method to the automation engineers yet despite its advantageous characteristics while testing is prevalent as a quality assurance method. Although there have been various specification languages are developed, cost-wise efficiency as well as the accessibility from the automation engineer' point of view is still to be improved. That is, the barrier of the effort to obtain the specification would still be high from the viewpoint of automation engineers, and the concept regarding how to utilize correspondingly generated specifications more effectively to improve the aPS control software quality is also required. Therefore, the approach developed in this thesis is to fulfill the requirements, filling the derived research gap.

5. A Concept of the Table-Based Formal Specification Language For Reactive System Software

The system software of the production system is required to be dependable and reliable, so the quality of the implemented system should be controlled during the implementation process. Also, the correctness of the system execution has to keep being monitored during operation to complement the overall quality of the system. Necessary properties should be verified, and some plausible scenarios should be considered and tested. Once validated, the integrated system is under the commissioning phase and then on the operation. Expected faults and failures could be captured and prevented during the implementation; however, unexpected and unintended behavior should also be captured and handled correctly. This explains the necessity of the run-time validation or monitoring, complementing the actual environment. Both the validation during the development and the monitoring during operation requires the specification of the requirement, i.e., how the control software is supposed to behave, as a base. Generalized test tables (GTTs), the presented approach, shall take the role of this specification which can be handled by the developers and validation engineers.

In the following, the table-based formal specification approach to ease obtaining the formal specification for production system software, or reactive system in more general scope, is presented. Although the separate proof will not be delivered, the approach also works for the cyclic behavior of mobile vehicles developed in IEC 61131-3 (e.g., software developed for CR0133 controller of ifm electronics Ltd., a programmable controller for mobile machines) as the system software characteristics are similar.

The goals of the presented concept are basically to embed formal specification and its usage within the control application development and operation regarding quality control. During the development, assured quality is required to obtain reliability and dependability and can benefit from formal verification, which requires a formal model of the system part and formal specification of the requirement part. Typical development activities for the control applications are to implement the required behavior newly or to change the existing code. After that, the implemented code should be validated regarding the conformity to the requirement. Formal verification could step in the validation of the model or the code level requiring formal model and specification. After the verification, its result shall be useful to debug and fix the code (cf. R-E3). Also, for the purpose of regression validation, regression verification reduces the effort of obtaining formal specifications

for the unchanged part. Thus, in any cases, formal specification is required, and the effective specification method is beneficial that users (i.e., module developer, application engineers, and commissioning technician) describe the desired system behavior precisely from the initial states and further state changes (cf. R-E5, R-E6) with respect to the cycles or specified timings (cf. R-T1, R-T2, R-T5). The created specifications should be useful to describe not only a specific behavior but also the same sort of behavior so that the verification coverage could be as wide as possible (cf. R-T3). In addition, the fact that the user (module developers, application engineers, and start-up technician, cf. Figure 7) should be able to easily access to the language to create, understand, and modify, is required (cf. R-U2 – R-U6).

The presented approach aims at reaching these goals by specifying the software behavior in the form of tables, which are based on the expert feedback from industrial companies of machine and plant manufacturing regarding the development processes. The concept of utilizing formal specification will be overviewed in Section 5.1, followed by the detailed approach description (Section 5.2) and application example (Section 5.3). The created specification in this concept takes a role of a formal specification in the format of tables with abstracting notations in cells describing constraints instead of specific values. Since it is grounded on the knowledge and experience of automation engineers, specification in this language shall be easily handled by them. Thus, the existing specification can be used for the validation process, which is to be proven in chapter 7, or, if not available, a base of the sound specification can be generated so that engineers can complete it to their tastes as a further use case of the presented approach, introduced in Section 5.4.1. As a requirement description, the created specification in this language can be transformed into a program for monitoring block as a supplementary run-time validation in Section 5.4.3.

5.1. Concept overview

One of the most significant problems to apply formal verification lies in the preparation of the formal specification. The requirement specification is not generated even in a natural language in the previous engineering processes for new modules or existing modules. Thus, the developers have to generate the specification newly in addition to handle (i.e., generate or revise) the test cases for testing. These are typically in the form of regression testing, meaning to check the original behavior, which should not be affected by the code revision. In a similar concept, regression verification [StGo08] (cf. Section 2.3.2) could be applied to the IEC 61131-3 control software change [BUVW15] using the original software as the original behavior description. However, still, the changed part, although it is small, has to be described to complete the change implementation verification. With conventional formal languages, it is hard due to the low accessibility from the

developers. Many logic languages are not easy to learn and also to be applied for the description of complicated processes.

Testing, or also for regression testing, requires the automation engineers to specify test cases, and these are done in table form in the industry. Focusing on the point that test cases and their procedures are described in a consistent way (not always perfectly but most of the time partially), which is concrete test tables, a formal specification language is developed providing generalizing concepts on the existing test tables. Suggested concepts are threefold: abstracting concrete values, referencing other values in the same or other timing, and abstracting timing. Presented concepts are enlightened by being applied in an application example over the engineering processes.

The suggested method can take a significant role to control the quality of the control software and be embedded over the lifecycle of a production system (Figure 1). First, during the development, the software is implemented after the requirement specification is understood by the module developer. Even if the specification has not been generated in the previous step, developers would generate one to be used in the validation step. The implemented software is validated by formal verification and testing. In the formal verification, the specification and the implemented model (including code) is the input of the verification process, and the conformance of the given model to the specification is the output, not requiring exact execution while the testing requires the exact execution in a given case. At this point, the requirement specification is a basis to decide the correctness of the behavior. After modules are developed, application engineers understand the behavior of modules from the specification and select the appropriate modules to consist a project.

Once the integrated system is validated, it is delivered to the site and gone through the final adjustment during the commission by the on-site commissioning technician. The detailed behavior is understood by reading the specifications for the integrated system or the module level if necessary. In case any small change is to be made on-site, the change is to be documented in the change specification and implemented change is verified regarding that.

During operation, significant signals have to be monitored for contingencies using monitoring functions. Monitoring functions are also generated directly from the specification to be independent of the implemented software, aimed at preventing any violation against the in-tension and recognizing any unconsidered case occurring during the operation. Also, the software could be requested to change its behavior. In this case, the specification would be updated if any exists. If not, running software could be a source to generate the specification base, on which the change intention can be added. After the change specification is correctly generated and the software change is implemented, the application developer can execute the validation process again with

regard to implemented change. System quality of the initial implementation of the change implementation can be supported in this two-way (forward and backward) quality assurance process.

5.2. Generalizing test tables as a specification representation

In this section, the formal specification approach is presented, namely GTTs. The name comes from the concrete test tables from which the structure was conceptually taken, taking benefits regarding the requirements of input/output traces and sequence of state changes. Generalization concepts are added on top of the structure of GTTs to sort the same type of states as well as the same type of state cascading could be represented specifically suitable for the change description. This part of the approach was preliminarily published in [BCUV17, CWUB18a, WWUU17].

5.2.1. Structure of GTTs

A control software initializes at the beginning by the initialization part of the program if required. It might just continue for a certain time regardless of any input or keep checking a specific status (input) of the system to terminate the initialization process. After the initialization is done, the software waits for a certain signal to execute a specific behavior, to result in certain output status, or already executes proactively. This kind of state changes, i.e., executing a certain behavior until it gets some signals and changing the behavior, repeat over the entire execution of the control software. Therefore, first, a correct description of the control software behavior would include conditions to change status and the resulting behavior necessarily. Also, as the change appears cascading, the sequence of the state changes should be intuitively represented. Second, the format should be easily accessible by the engineers (i.e., module developers or application engineers mostly), exploiting or extending existing practices optimally so that they could be easily embedded within the present engineering workflow.

In a concrete test table, each cell contains a specific value corresponding to the variable. Input values are what is assumed to be given to the test target, and output values are what is expected to be obtained from the software execution result of that cycle(s) at a specific time or time duration. Though there is no formally defined notations of concrete test tables, the characteristic of clearly structured input/output sequence traces could be taken as a promising structure for the specification language. Furthermore, the applicability can be reserved as they are the current practices.

A generalized test table is constructed in the form of a table including columns of the sequence number, variables, and duration (Figure 8).

- Sequence number: A *sequence number* is the index of each row. Example: the sequence number of the 1st row is 1.

- Variables: variables are assorted into *input* (IVar1 or IVar2 in Figure 8), *output* (OVar1 or OVar2 in Figure 8), and *local variable* if necessary. Each column header indicates the name of the corresponding one. Input is not limited to the exact input wires (or variables) of the function block but might include any global variables which affects the behavior of it. Similarly, output includes all the signals that would be asserted after the execution of the function block in the cycle, assuming the variable values in *input* section. (cf. R-T1)
- Duration: a *duration* (typically shortened as Dur.) indicates for how many cycles or for how long time the set of variables would take place as valid behavior (cf. R-T2, R-T5).
- As an optional column, a *remark* column could be added to annotate a row with text for better understandability of the behavior description.
- In each row, each variable value are written within the corresponding cell in order from the top row.

The input in a row is the assumed variable value(s) to be induced to the system at a specific time. Therefore, the input section is described with the values that would be expected to occur while the system is being in the previous state. Also, the output section is filled with the output variable values that would be observed mandatorily. Values that would be assumed to be given after the target function block starts to execute and corresponding output are found in row 1. Further sequences appear in the following rows in order, e.g., the input part of row 2 indicates the condition that makes the state which produced the output indicated in the output part of row 2.

Seq.	Input			Output			Duration	Remark
	IVar1	IVar2	...	OVar1	OVar2	...		
1								
2								
...								

Figure 8: Basic structure of the GTTs

The structural definition of the GTT does not include any concept of generalization but provide the standardized structure of the specification in table form. It could be applied to the typical test tables.

5.2.2. Value referencing and generalization

As reactive systems, the control software of aPS accepts the input trace and results in the corresponding output trace continuing to changes behaviors depending on the states. The signals affect each other, mostly output is affected by input (history) and sometimes input is also affected in reverse as the previous output changes the status of the environment. Therefore, the relationship between the signals is important information for the behavior. In concrete test case, exact values of signals (input and output) are designated in the corresponding cells, but exact values cannot allow to represent the relationship of them.

Having a test table (or also in a timing diagram which represent the concrete value in a graphical way) implies a concrete value trace of input/output values, which goes through a specific program run path. As our target is to describe the behavior of the program path, not the exact value trace, possible values are to be aggregated to present a corresponding path that the same sort of value trace would go through. If it is described in the concrete test tables, a bunch of test tables would be required to cover all possible value combinations. It is worse in a timing diagram to represent a varying possible range of a signal envelope. Therefore, it is required to formally describe a range of a signal or a constraint of it to be more general. This will be related to the coverage of the verification in the sense that it allows the model checking to prove the software to satisfy with respect to all possible values within the range or under the constraints.

In GTTs, therefore, constraints of the signal of each state is allowed to be represented as presented initially [BCUV17, WWUU17]. Constraints mean the possible eligible values to appear or to have on that state (or row in the table point of view). Constraints might appear in different forms:

- A variable is to be a certain value as a typical concrete test table does. The values should follow in the data type of variable, including enumeration.
- A variable is to be within the range of an interval with either minimum limit or maximum limit, or both. This case also includes a specific impossible value. The values are indicated by using mathematical symbols such as equals sign (“=”), inequality signs (“>”, “<”, “≠”, etc.), and intervals (“[,]”, “(,)”).
- A variable might not be effective as an assumption (input) or assertion (output). In this case, the cell value is indicated as ‘don’t care’ with a dash symbol (“—”).
- Mathematical and logical expressions using operators in IEC 61131-3 ST are also allowed as a cell value. That is, the constraints explained earlier could be concatenated with logic operators and even library functions can also be used to describe the cell value.

A variable cell could also refer to the other cells to expand the expressiveness of the constraint (cf. R-T3). As discussed, signals in aPS depend on each other in many cases. Also, the system behavior depends on not only the current input but also previously input values. Two expressive means to formulate such dependencies are possible in GTTs.

- Past-referencing: Values can be referred by using a specific variable name with a timing index which indicating the backtracking cycle amount. If the value is referred from the current row (i.e., in the same cycle), the variable name is to be used along with the timing index omitted.
- Global-referencing: Values can be referred to by using a specific symbol over relevant cells. The symbol is typically assumed to be selected from a lower-case alphabet letter not

overlapping with the variable names. After designating a cell with the selected symbol, the relational value is described referring to it.

As IEC 61131-3 expressions are allowed within a cell as a constraint representation, reference symbols used for past- and global-referencing are eligible to be the operands for those expressions. (cf. R-T6). Table 4 shows allowed expressions and values in the cells of a GTT under the variables section:

Table 4: Possible cell values and notations

	Cell value	Description	Examples
1	Specific values	as typical concrete test tables could, cells under variable sort can include specific values in the type of variable including defined enumerations.	1, 0, TRUE, GREEN
2	Intervals	Range of the values	[1,5], [7,∞]
3	Constraints	Restriction on the value	≥ 5 , $\neq 7$
4	ST-expression	Expressions allowed in ST language including library functions	$=I_3 * O_5$, $=\text{MIN}(I_1, 10)$
5	–	Don't care	–
6	References	Referring to the other cell with a column name or a symbol	$=I_3$, $=O_5$, $=a$
7	Past references	Referring to the same cell or the other cell with timing index	$=I_3[-1]$, $=O_3[-2]$,
8	Boolean constraints	Conditions to be adhered by the input/output values	$I_2 > I_3$ AND $I_3 \geq 5$

5.2.3. Generalizing duration

Signals are the motive to evolve the process within aPS. In addition to this, timing is also a critical condition to be observed carefully since, first, the signals affecting each other and they are supposed to be synchronized correctly in complex systems like aPS, and second, the elapsed time is also related to the production performance which is put first as the main goal of aPS engineering.

A certain state may stay in a state resulting in output. That is, as long as the input condition meets, the output would be repeated for a designated number of cycles. Or, a state might wait for a specific signal with a timeout and this should also be enabled to represent. Thus, it can be said that the cells in the duration column determine the number of repetitions for each row. In this column, as presented [BCUV17, WWUU17] and further extended in [CWUB18b], the following values are possibly entered:

- The cell could include a certain value as a typical concrete test table does regarding the allowed repetition number of cycles or the exact time specifying the unit.
- The duration can be defined within the range of an interval with either minimum limit or maximum limit or both. This case also includes a specific impossible value. The values

are indicated by using mathematical symbols such as equals sign (“=”), inequality signs (“>”, “<”, “≠”, etc.), and intervals (“[”, “]”, “(”, “)”) indicating unit.

- The undefined duration is indicated with a dash symbol (“–”) meaning ‘don’t care’ (i.e., $[0, \text{INF}]$), also meaning ‘it might not occur.’
- The undefined duration guaranteeing the occurrence is represented by a dash symbol with ‘INF’ subscripted, i.e., “–_{INF}”. This notation assumes that the row occurs at least once and the system keeps staying in that state. Thus, any row appearing after this notation is not meaningful.
- Block repetition: A duration could also be defined for some consecutive rows (i.e., row group) to be repeated as a block with the symbol “**I**” over the corresponding row span with its own duration constraints subscripted, e.g., “**I**_[1,3]”

Table 5 shows allowed values in cells under the duration section:

Table 5: Possible durations and notations

	Cell value	Description	Examples
1	Specific values	Exact values for cycles (default) or time (with specific unit)	1, 10, 7 sec
2	Intervals	Range of the timing duration	[1,5], [1,∞], [7sec,∞]
3	–	Means $[0, \infty]$ which allows row to be skipped or repeated arbitrary times.	–
4	– _{INF}	Strong repetition: program state stays in this row	If a program should arrive a certain state and stay in that state
5	I	Indicating row group which would be repeatedly executed	I _[1,3]

5.2.4. Simplifying exclusion

In GTTs, each row could be mapped to the conditions to change the state of the function block (input) and the corresponding state of the result (output). Sequential state changes are the target to describe. During execution, a system could stay in a state for a while (i.e., multiple cycles) as described in the duration section it waits for a certain condition, i.e., input constraint in the next row in the context of GTTs, satisfying the exact input constraint in the row and the timing condition in duration section. With this characteristic, there sometimes appears tedious, and longer expressions and following notations are to achieve description efficiency

Exclusion over the adjacent row

During the execution conforming to the GTT, the PLC control software could be said as staying in a (defined) state or defined row in the term of GTT. Valid execution would be decided by

checking the input to decide the row and the corresponding output from that row. That is, the input will be continuously checked first whether the current state (row) is eligible to retain, meaning that the current input is satisfying the input column of the row with eligible duration, or whether the next row condition is satisfied as the enabled state. Violation of the output is to be checked after the row eligibility based on the input is determined.

Within the GTTs, the conditions, which decide the state retention or the state transition, are the input section. Describing the adjacent input section with overlapped values confuse the tester or let the state space bigger, meaning the code execution control could stay in multiple adjacent states (rows) although it is expressively possible. To obtain compressed expression but to avoid confusing meaning, progress flag, a specially described don't care duration with an additional annotation (" \neg_p "), is developed to discriminate the input condition easier by introducing a symbol than having several rows, meaning 'this row could be repeated as long as the condition of the successor row is not satisfied.' Suppose the indication of the initialization of a function block having *EN* as an enabling signal and *I* as an input signal with initialization criteria '*EN* = TRUE and *I* > 10 for two seconds'. The system is not under the concerned condition yet at the beginning most probably. One might write the initial condition as '*don't care*' and put the initialization condition on the second row (Figure 9 – a). In this case, every input value combinations could be accepted as valid for the first row even if the initialization is over and the module is supposed to behave normally, resulting in different output (e.g., *READY* = FALSE even after satisfying the initialization criteria would be regarded as valid behavior since it satisfies the first row) though it is not the intended behavior. GTTs should be described precisely, distinguishing the adjacent rows explicitly to avoid and overcome the unintended ambiguity; this might require more rows or complicated conditions to describe the inversion of the target condition. So one might also want to check enabling signal (*EN*) first and then check input signal (*I*) if *EN* is satisfied (Figure 9 – b). However, this will cause more rows to distinguish different states, e.g., *EN* is not satisfied in the first row, and *EN* is satisfied but *I* is not satisfied in the second row. If there are variables, describing transient states before initialization would be more complicated. To ease this exclusive expression, the duration could be specially described with the progress flag (Figure 9 – c). Thus, as soon as the input condition of the successor row is satisfied, the current row is not valid anymore and the validation should progress to the next row.

Seq.	Input		Output	Dur.
	EN	I	READY	
1	-	-	F	-
2	T	>10	F	<=2sec
3	T	>10	T	-
	...			

(a)

Seq.	Input		Output	Dur.
	EN	I	READY	
1	F	-	F	-
2	T	<=10	F	-
3	T	>10	F	<=2sec
4	T	>10	T	-
	...			

(b)

Seq.	Input		Output	Dur.
	EN	I	READY	
1	-	-	F	$\neg p$
2	T	>10	F	<=2sec
3	T	>10	T	-
	...			

(c)

Figure 9: GTTs to check the initialization condition (EN = TRUE and I > 10 for 2 seconds). (a) not correct with too relaxed condition at the beginning, (b) correct but requiring more rows, and (c) compact exclusion representation with progress flag

Indication of triggering

Input condition changes appear in separate rows in GTTs. When a certain sensor value should trigger a certain behavior, the observing pattern would be FALSE continuously and a sudden TRUE (rising edge) then appears. The triggering often occurs in a moment and then the rest of the signal (until the software expects another condition) is not concerned after triggered. Thus, the output section is identical in both the triggering moment and the rest (Figure 10 – a, #2-3, # meaning the sequence number hereafter). In this case, the input condition could be represented as “trigger” signal by adding a corresponding symbol, i.e., “v” within the cell, in front of the exact value (Figure 10 – b, #2). This reduces the number of rows to describe the triggering signal, increasing the readability of the table by showing the signal characteristic (triggering) explicitly.

Seq.	Input	Output	Dur.	Remark
	X	Y		
1	F	F	-	
2	T	T	1	Rising edge observed
3	-	T	99	
	...			

(a)

Seq.	Input	Output	Dur.	Remark
	X	Y		
1	F	F	-	
2	vT	T	100	Rising edge observed
	...			

(b)

Figure 10: GTTs to check the rising edge of X. (a) waiting for the input condition to be true and then output continues for 100 cycles by designating two different values of X in two rows, and (b) compact rising edge check expression with triggering symbol

5.3. Application example

To show the concepts of GTTs more intuitively from the potential user’s point of view as this is an important requirement (R-U2 – R-U6), a case of conventional logic is shown here exemplarily and the main features are presented. The overall description (Section 5.3.1) and the requirements (Section 5.3.2) are the information which has been recognized by the engineer. Using the test tables (Section 5.3.3), a GTT to describe the required behavior will be created (Section 5.3.4).

5.3.1. Description of the module

A linear conveyor module transports a work piece from one edge to the other (Figure 11). During the transportation, work pieces could be separated by being guided into different direction depending on the material types: one is metal and the other is plastic. There is one storage for plastic (i.e., non-metal) type work pieces at the end of the conveyor. Metal work pieces are supposed to be redirected to another conveyor for a further process (Figure 12). One work piece is put and assorted into the appropriate place (i.e., storage or the further conveyor) at one time and the next one comes after the earlier one is processed.

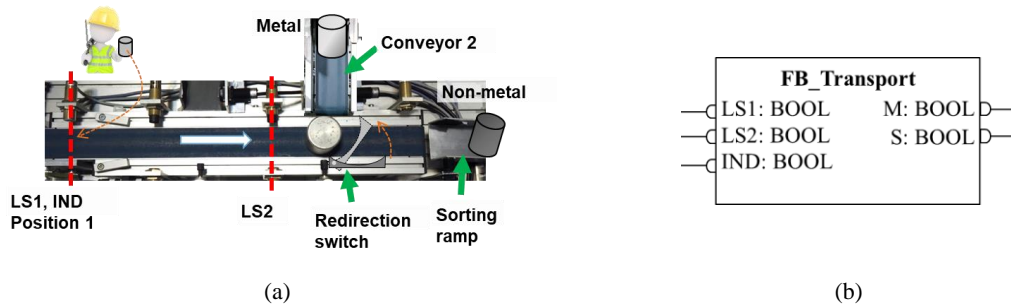


Figure 11: Conveyor separation module to sort different type of work pieces and the corresponding function block to control the separation process: (a) graphical overview of the machine, and (b) schematic view of the function block with interface indicated

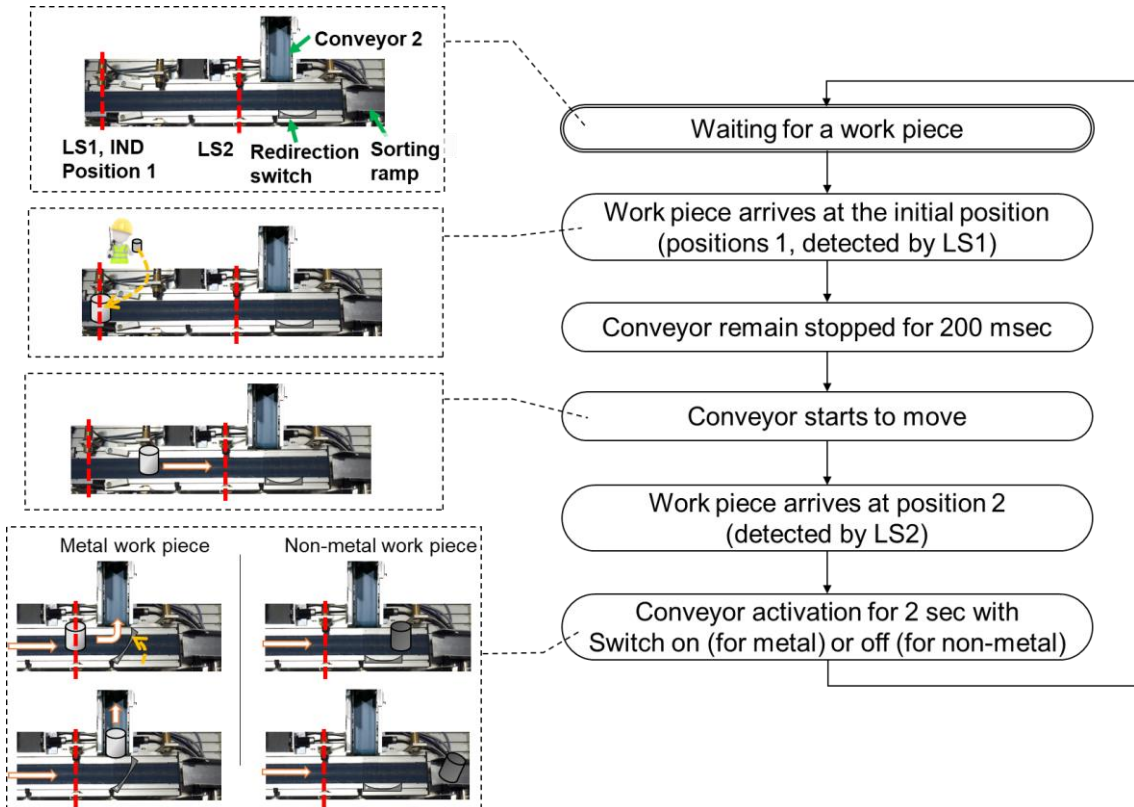


Figure 12: Separation behavior flow chart with the illustration of the machine and the work piece

5.3.2. Requirements of the module

For an automatic control of the separation process, requirements in the textual description are put as below:

- A work piece is put on the conveyor at position 1. The presence of the work piece and its type is detected by the presence sensor LS1 and the inductive sensor IND respectively. Both are binary sensors indicating positive with True value.
- The conveyor is actuated by the motor, controlled with a Boolean variable M. (True: move, False: stop)
- For stabilization after being put on the conveyor, the conveyor starts its operation 200 ms after the arrival of the work piece.
- Conveyor moves until the work piece arrives at position 2, indicated by the additional presence sensor LS2.
- When a work piece arrives at LS2, the switch is controlled either to stay open or to close depending on the type of the work piece material. If it is metal type, then the switch, which is controlled by a Boolean variable S, closes so that the work piece is guided into the conveyor 2.
- After the switch is controlled while the work piece is passing by the LS2, the conveyor continues to operate for 2 seconds so that the separation is completely done.

5.3.3. Test cases for the module in test tables

Based on this requirement, an application engineer compose the existing function blocks (e.g., switch function block, conveyor function block) which are developed earlier by the module developers, using the glue code. The developed code is supposed to be validated so some test cases would be generated, as depicted in Figure 13 showing two possible cases in the test tables (cf. R-E2).

First, the plastic work piece separation test is shown (Figure 13–a). Once the conveyor module is started to operate, it waits for the work piece and it is assumed to insert the work piece after three second although it is fine to place the work piece at any time (#1). Then, the work piece is detected and conveyor should remain as stopped for 200 ms (#2). After this time passage, the conveyor operation is enabled (#3, M=TRUE). In the meantime, the work piece leaves position 1 (#4, LS2=FALSE), and arrives at position 2 (#5, LS2=TRUE). At this moment, although the conveyor continues the operation, the switch has to be controlled. As the work piece is detected earlier as non-metal (#2, IND=FALSE), the switch is to remain open so that the work piece is delivered to the storage at the end (#5, S=5) after two seconds of the conveyor operation. After the separation is done, the conveyor stops.

In the second test case, another separation behavior is tested for the metal work piece type (Figure 13–b). In this case, the timing of putting work piece is selected differently to benefit of diversity, which is five second. Since the testing material is metal, the inductive sensor gives positive value (#2, IND=TRUE). The next steps process similar to the first test case up until the work piece arrives at LS2. When it is passing by LS2, the switch has to be closed so that the work piece can be guided to another direction, which is conveyor 2 (#5, S=TRUE).

Seq.	Input			Output		DUR.
	LS1	LS2	IND	M	S	
1	FALSE	FALSE	FALSE	FALSE	FALSE	5 sec
2	TRUE	FALSE	TRUE	FALSE	FALSE	200 ms
3	TRUE	FALSE		TRUE	FALSE	–
4	FALSE	FALSE		TRUE	FALSE	–
5	FALSE	TRUE		TRUE	TRUE	2sec
6	FALSE	FALSE		FALSE	FALSE	

(a)

Seq.	Input			Output		DUR.
	LS1	LS2	IND	M	S	
1	FALSE	FALSE	FALSE	FALSE	FALSE	3 sec
2	TRUE	FALSE	FALSE	FALSE	FALSE	200 ms
3	TRUE	FALSE		TRUE	FALSE	–
4	FALSE	FALSE		TRUE	FALSE	–
5	FALSE	TRUE		TRUE	FALSE	2sec
6	FALSE	FALSE		FALSE	FALSE	

(b)

Figure 13: Test cases for (a) non-metal work piece, and (b) metal work piece

5.3.4. Behavior specification in GTTs

Test cases are generated and they can be used to test those exact cases, missing other possible cases. For example, the timing of the work piece put on the conveyor, or the time passage for the work piece traveling from LS1 to LS2 vary. Also, some sensor values do not have to be considered all the time. For example, the inductive sensor value while the module is waiting for the work piece should not affect the behavior. One might think it is necessary to test such a situation (i.e., no work piece but the inductive sensor is detected – although this is obviously malfunction of the sensor, sensor correctness is irrelevant to the subject at this moment, so is not considered as a case to resolve). The variety of possible values and their combination makes an infinite number of test tables. However, all these cases are included in the same sort of behavior. Changes often occur within the machines – in this separation module case, there could be many points to be changed such as reestablishment of the sensors due to the physical restrictions of the machine position, adjustment of the conveyor belt length due to interfacing with the other machine part, and so on. In the case of changes, especially when it is implemented on-site, it is hard to execute all test cases. Instead, verification could be done statically using the changed code part and the specification. Now, the generalized behavior is presented in GTTs as the formal specification language using the introduced concepts (Figure 14).

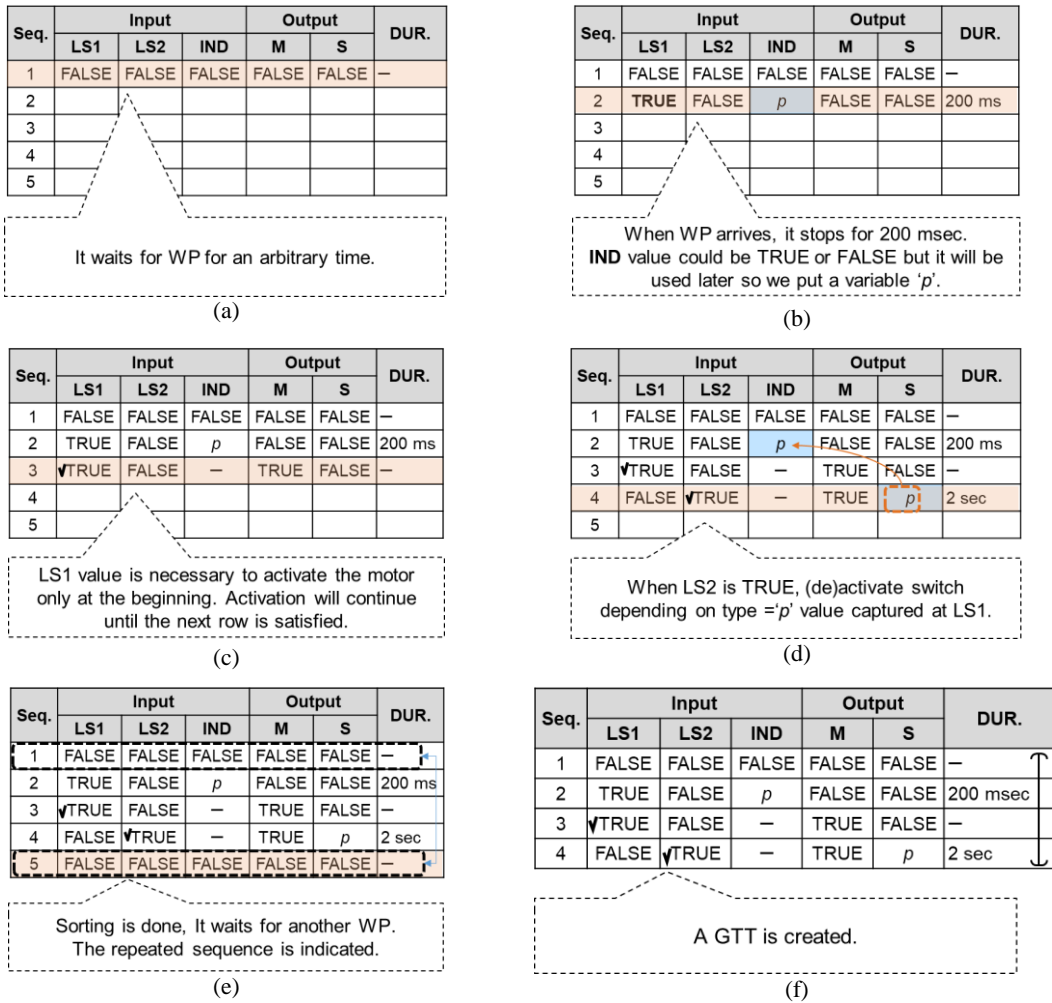


Figure 14: Generation of a GTT

At the beginning of the module execution, the motor and switch are disabled as long as there is no work piece on the conveyor (Figure 14 – a #1, LS1=FALSE, LS2=FALSE). Since it is obvious that the inductive sensor value is invalid, it is not considered as an effective condition, so that the output still remains as inactivated. Thus, ‘don’t care’ symbol (“—”) can be put for this signal. Considering duration, the module may wait for a work piece for an arbitrary time because it is not deterministic when the work piece will be inserted (#1, DUR.). This will cover the test case of putting work piece at various timing. As ‘don’t care’ in the duration cell indicates that the state (row) might happen multiple cycles and even might not happen. In other words, there might exist a work piece already when the module starts to execute.

When a work piece arrives, or when a module starts and a work piece is already detected (i.e., the state of row#1 has not appeared), the conveyor should wait further for 200 ms in case the work piece is to be stable. During this time, the existence of the work piece is indicated as well as the material type. From the requirements, it could be learned that this material type information affects

one of the actuator, i.e., switch. But as the timing, when this information is used, comes later, a variable named 'p' is put in this IND cell (Figure 14 – b, #2).

After the stabilization time (200 ms) exceeds, the conveyor starts to move with the exit condition that change the output (M and S) state as 'LS2=TRUE' (Figure 14 – c, #3). Until that moment, the conveyor should keep moving. During this movement, the work piece is detected by LS1 at the beginning, and then not detected anymore as it leaves the position 1 although the movement should go on. Therefore, LS1=TRUE is the condition that is satisfied at the beginning definitely but it does not have to be considered anymore (#3, LS1 = $\sqrt{\text{TRUE}}$). This triggering abstracts the possible timings when the work piece would be out of sight of LS1, which cannot be exactly decided in fact.

The output status changes when work piece passes by the position 2. The separation is prepared at this point with the switch open or closed depending on the material type. The material type has been detected at the position 1 (thus, reading IND at this point is meaningless as the work piece has been already moved away) and this will decide the switch signal. By the requirement, the metal work piece (IND=TRUE) will be moved to conveyor 2 (S=TRUE), and the plastic one (IND=FALSE) will be moved to the storage at the end (S=FALSE). Therefore, the relationship between the IND and S based on the effect of IND can be described simply (Figure 14 – d, #4). For a tweaked example, if the separation storage is swapped, it would have been described as inversion (i.e., \bar{p}). Though this output change is started to be affected when the work piece passes by position 2, the change should be effective for two seconds until the work piece is handed over to the target. Therefore, the LS2 value should also have a triggering symbol as the condition of the output so that the output remains after the work piece is out of sight of LS2.

After the separation is completed, the conveyor should stop and the switch should also be open (to the default state) as long as no work piece appears (Figure 14 – e). In fact, this condition and the behavior are same as the initial state. So the function block could be said that it comes back to the initial state, which is #1. In addition to this, it could be easily expected that further execution would follow the behavior shown so far. Therefore, the repetition indicator is inserted with its duration condition omitted, which means it repeats infinitely.

Conclusively, the formal specification of the conveyor separation module is prepared (Figure 14 – f). Based on this, one might generate possible test scenarios in the case of testing. This could also be inserted into the model checker together with the IEC 61131-3 code for formal verification. Later on, another application engineer would need to know the behavior of this separation block; (s)he would read this specification and then figure out the behavior by seeing the relationship

between the signals and state changes together with the conditions for that. Though the shown example was assumed for GTTs to be used by application engineer, module developers also could do similar to document the formal specification of the target module for the validation activities and to keep the description for the others or further uses (cf. R-E6).

5.4. Extension of the developed approach

Software specification is an essential entity in the formal verification context, and this forward direction verification has been the main focus so far to explore the possible formal specification language which is accessible to the automation engineers (forward validation, Figure 15 – F). Extending the scope, further utilizations of the developed formal specification technique are presented in this section to support better software quality. After the verification, its result includes counterexample information with regards to the corresponding specification in case of proof failure. The counterexample is a very important source to inform the developer, in which execution trace the specification is violated; and it is, thus, beneficial to present the counterexample to the developer in an intuitive way (forward, F, Section 5.4.1). During operation, or even during commissioning, developed systems face changes to fix some unresolved bugs or to cope with the requirement change, and there has to be validated (backward, B1, Section 5.4.2). Also, expected faults and failures could be captured during operation and prevented during the implementation; however, unexpected and unintended behavior should also be captured and handled correctly by the run-time validation or monitoring (backward, B2, section 5.4.3).

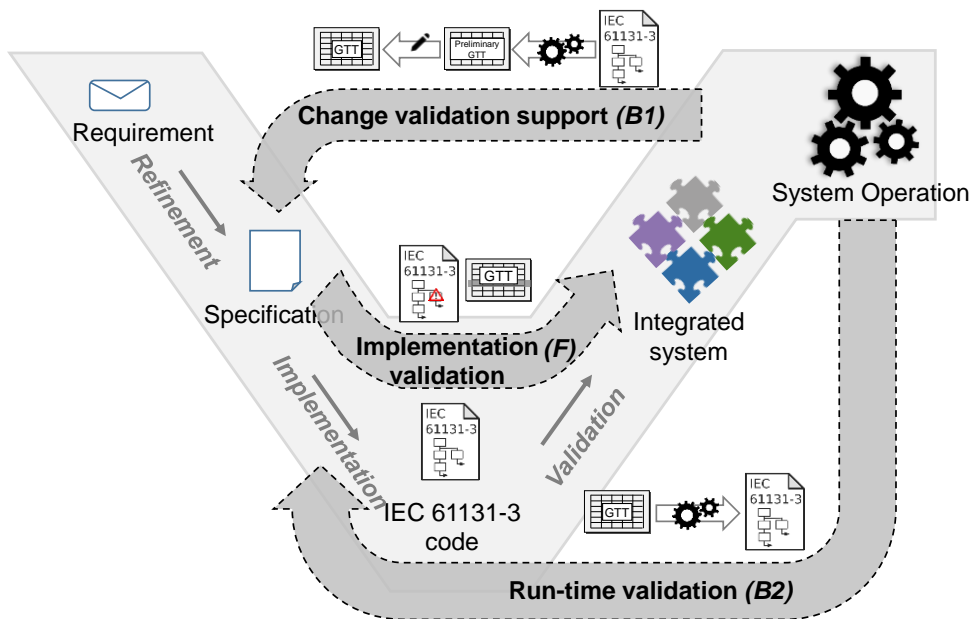
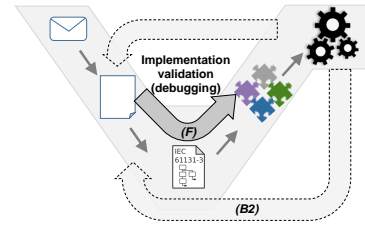


Figure 15: Two-way quality assurance process – forward (F) and backward (B1 and B2)

5.4.1. Debugging through the counterexample presented as related to GTT(F)

The counterexample provided from the model checker is a good source for developers since such a counterexample consisting of relevant variable traces exhibits a particular execution where the proof failure, meaning the execution of the implemented code does not satisfy specification. In other words, the counterexample represents the execution traces up to the violation point, and the users can follow the input/output traces step by step with the program code. As the focus during the validation (through the verification here) of the developer is put to assure the conformance of the implemented code to the specification, any violation is what to be resolved right away (cf. R-E3). The variable value tracking with regards to the specification could also be manually; however, it is a tedious and vulnerable (to make mistakes by hand) task when the signal trace is long or many variables are involved in the trace.



Graphical representation of the counterexample

The variable values could be presented in a timing diagram also to present the sequences of the concrete values with each variable has its own lane to present the value sequences presented horizontally with the unit of the cycle. For this graphical representation, some additional concepts could be added to improve the effectiveness. First, the last few timing cycles before the violation occurs are to be shown with hiding all the earlier values. This allows the developers to focus on the last few value or state changes (Figure 16 – i). At the same time, the visible variables are also to be contracted since typically many variables are involved; but the violated variables are only to be visualized (Figure 16 – ii).

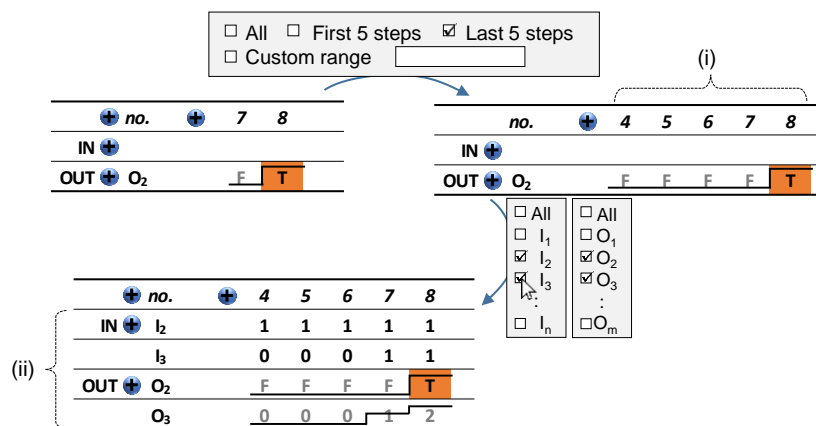


Figure 16: Counterexample representation (i) timing contracting/expanding, and (ii) variable contracting/expanding

Connecting traces to the GTT

The counterexample is a sequence of concrete execution values resulted from the code execution regarding all relevant variables. Therefore, it is an instance of the GTT with all concrete values, and all durations and repetitions unfolded. This means again, each tuple of variable values in the counterexample are corresponded to a specific row of the GTT in order. Therefore, presenting the corresponding GTT rows for the value tuples of counterexample helps developers to observe the changes of the value and state in the counterexample more effectively.

One exemplary way of putting the GTT besides the counterexample representation with displaying the connection between them is shown in Figure 17. Basically, all the counterexample value tuples correspond to a specific row of the GTT from the trace until the violation conforms to the GTT. On the cycle where the violation occurs, the violated variable value within the GTT is highlighted to inform the developer the direct violation point graphically, together with the entire row. In this way, the developer can intuitively recognize which row of the GTT is violated (from GTT highlighting) with which value (from counterexample).

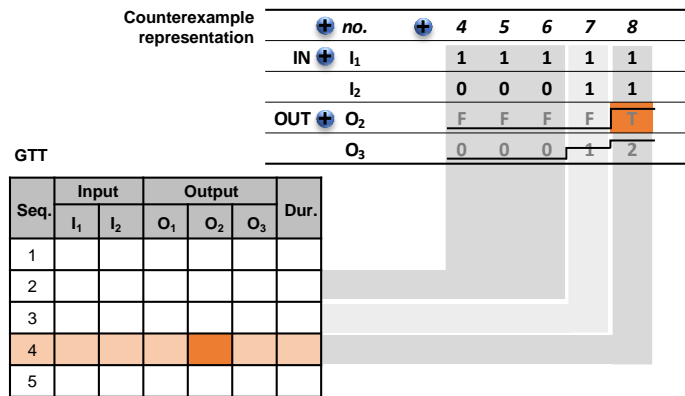
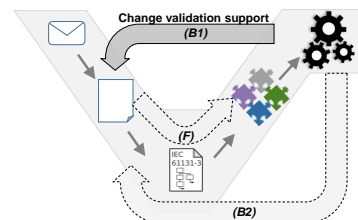


Figure 17: Connecting the counterexample to the GTT

5.4.2. Achieving preliminary specification out of IEC 61131-3 program in GTT (B1)

GTTs shall lower the barrier for automation engineers regarding software, such as module developers or application engineers, to write the formal specification of the program code that they are implementing. Still, it is a burden to them that they have to generate a formal specification from scratch to utilize formal verification.



Usually, the formal specification of the existing software code is mostly absent and the developer needs to be aware of its exact behavior to describe delta correctly. As developers lack time for development and validation, it is necessary to have an efficient way to create a specification,

especially in the case of implementation of changes (cf. R-E5). This part of the approach was preliminarily published in [CWUB18b].

Software consists of state changes and these state transitions are motivated by the input or timing condition. Following the control flow within the software assuming conditions, a program run could be extracted and the assumed conditions as well as the extracted program part could be represented as a partial specification of the software. This is especially useful for the automation software change that is small. In the small change case, a specific program run is supposed to be affected and this fits very well with the characteristics of GTTs as they target to depict a specific program run.

SFC is one of the graphical IEC 61131-3 languages. It is used to describe the control flow structure of the system in a sequential nature. SFC consists of three main elements: steps, transitions, and actions. Steps are the unit of the sequence controls. Transitions indicate turning over the control from one step to another step decided by the transition guard. Actions are the events included in the step and executed while the step is active. Each SFC step execution consists of the execution of the actions, which could be regarded as an output state. In other words, if a program control is located in a step within SFC program, this means that the actions within the step is repeatedly executed un the exit condition is satisfied.

Converting from SFC into GTT

SFC software is implemented following the control sequence, which is also the nature of GTT. Thus, this can be systematically converted into GTT form by mapping its SFC elements into GTTs'. A step with actions is converted into two rows (for non-timed related actions: N, S, R, P and P0) or three rows (for time-related actions: L, D and DS) in the Table 6.

Table 6: SFC action qualifiers and its converting pattern into GTT

(a) N	<table border="1"> <thead> <tr> <th>Seq_Name</th> <th>Input</th> <th>Output</th> <th>Dur.</th> </tr> </thead> <tbody> <tr> <td>Step</td> <td>G1</td> <td>TRUE</td> <td>T_c</td> </tr> <tr> <td>Step(w)</td> <td>!G2</td> <td></td> <td>$[0,^*]$</td> </tr> <tr> <td>Next step</td> <td>G2</td> <td>FALSE</td> <td>T_c</td> </tr> </tbody> </table>	Seq_Name	Input	Output	Dur.	Step	G1	TRUE	T_c	Step(w)	!G2		$[0,^*]$	Next step	G2	FALSE	T_c	(e) P0	<table border="1"> <thead> <tr> <th>Seq_Name</th> <th>Input</th> <th>Output</th> <th>Duration</th> </tr> </thead> <tbody> <tr> <td>Step</td> <td>G1</td> <td>FALSE</td> <td>T_c</td> </tr> <tr> <td>Step(w)</td> <td>!G2</td> <td>FALSE</td> <td>$[0,^*]$</td> </tr> <tr> <td>Next step</td> <td>G2</td> <td>TRUE</td> <td>T_c</td> </tr> </tbody> </table>	Seq_Name	Input	Output	Duration	Step	G1	FALSE	T_c	Step(w)	!G2	FALSE	$[0,^*]$	Next step	G2	TRUE	T_c
Seq_Name	Input	Output	Dur.																																
Step	G1	TRUE	T_c																																
Step(w)	!G2		$[0,^*]$																																
Next step	G2	FALSE	T_c																																
Seq_Name	Input	Output	Duration																																
Step	G1	FALSE	T_c																																
Step(w)	!G2	FALSE	$[0,^*]$																																
Next step	G2	TRUE	T_c																																
(b) S	<table border="1"> <tbody> <tr> <td>Step</td> <td>G1</td> <td>TRUE</td> <td>T_c</td> </tr> <tr> <td>Step(w)</td> <td>!G2</td> <td></td> <td>$[0,^*]$</td> </tr> <tr> <td>Next step</td> <td>G2</td> <td></td> <td>T_c</td> </tr> </tbody> </table>	Step	G1	TRUE	T_c	Step(w)	!G2		$[0,^*]$	Next step	G2		T_c	(f) LT#T_x	<table border="1"> <tbody> <tr> <td>Step</td> <td>G1</td> <td>TRUE</td> <td>T_c</td> </tr> <tr> <td>Step(t)</td> <td>!G2</td> <td>TRUE</td> <td>$[0, T_x - T_c]$</td> </tr> <tr> <td>Step(w)</td> <td>!G2</td> <td>FALSE</td> <td>$[0,^*]$</td> </tr> <tr> <td>Next step</td> <td>G2</td> <td>-</td> <td>T_c</td> </tr> </tbody> </table>	Step	G1	TRUE	T_c	Step(t)	!G2	TRUE	$[0, T_x - T_c]$	Step(w)	!G2	FALSE	$[0,^*]$	Next step	G2	-	T_c				
Step	G1	TRUE	T_c																																
Step(w)	!G2		$[0,^*]$																																
Next step	G2		T_c																																
Step	G1	TRUE	T_c																																
Step(t)	!G2	TRUE	$[0, T_x - T_c]$																																
Step(w)	!G2	FALSE	$[0,^*]$																																
Next step	G2	-	T_c																																
(c) R	<table border="1"> <tbody> <tr> <td>Step</td> <td>G1</td> <td>FALSE</td> <td>T_c</td> </tr> <tr> <td>Step(w)</td> <td>!G2</td> <td></td> <td>$[0,^*]$</td> </tr> <tr> <td>Next step</td> <td>G2</td> <td>-</td> <td>T_c</td> </tr> </tbody> </table>	Step	G1	FALSE	T_c	Step(w)	!G2		$[0,^*]$	Next step	G2	-	T_c	(g) DT#T_x	<table border="1"> <tbody> <tr> <td>Step</td> <td>G1</td> <td>FALSE</td> <td>T_c</td> </tr> <tr> <td>Step(t)</td> <td>!G2</td> <td>FALSE</td> <td>$0, T_x - T_c$</td> </tr> <tr> <td>Step(w)</td> <td>!G2</td> <td>TRUE</td> <td>$[0,^*]$</td> </tr> <tr> <td>Next step</td> <td>G2</td> <td>FALSE</td> <td>T_c</td> </tr> </tbody> </table>	Step	G1	FALSE	T_c	Step(t)	!G2	FALSE	$0, T_x - T_c$	Step(w)	!G2	TRUE	$[0,^*]$	Next step	G2	FALSE	T_c				
Step	G1	FALSE	T_c																																
Step(w)	!G2		$[0,^*]$																																
Next step	G2	-	T_c																																
Step	G1	FALSE	T_c																																
Step(t)	!G2	FALSE	$0, T_x - T_c$																																
Step(w)	!G2	TRUE	$[0,^*]$																																
Next step	G2	FALSE	T_c																																
(d) P	<table border="1"> <tbody> <tr> <td>Step</td> <td>G1</td> <td>TRUE</td> <td>T_c</td> </tr> <tr> <td>Step(w)</td> <td>!G2</td> <td>FALSE</td> <td>$[0,^*]$</td> </tr> <tr> <td>Next step</td> <td>G2</td> <td>-</td> <td>T_c</td> </tr> </tbody> </table>	Step	G1	TRUE	T_c	Step(w)	!G2	FALSE	$[0,^*]$	Next step	G2	-	T_c	(h) DST#T_x	<table border="1"> <tbody> <tr> <td>Step</td> <td>G1</td> <td>FALSE</td> <td>T_c</td> </tr> <tr> <td>Step(t)</td> <td>!G2</td> <td>FALSE</td> <td>$0, T_x - T_c$</td> </tr> <tr> <td>Step(w)</td> <td>!G2</td> <td>TRUE</td> <td>$[T_c,^*]$</td> </tr> <tr> <td>Next step</td> <td>G2</td> <td>-</td> <td>T_c</td> </tr> </tbody> </table>	Step	G1	FALSE	T_c	Step(t)	!G2	FALSE	$0, T_x - T_c$	Step(w)	!G2	TRUE	$[T_c,^*]$	Next step	G2	-	T_c				
Step	G1	TRUE	T_c																																
Step(w)	!G2	FALSE	$[0,^*]$																																
Next step	G2	-	T_c																																
Step	G1	FALSE	T_c																																
Step(t)	!G2	FALSE	$0, T_x - T_c$																																
Step(w)	!G2	TRUE	$[T_c,^*]$																																
Next step	G2	-	T_c																																

(w): waiting, (t): timing constraint, !: negation, T_c : cycle time, T_x : timing constraint

First, entry transition guard in SFC is regarded as the input constraint of the first row with one cycle time (T_c). This row represents the activation of the step. On activation of the step, the actions are executed and, therefore, the effect of the actions should be observable in the output columns depending on the action types. For example, execution of **N**, **S**, **P**, **P0** and **L** will set the variable TRUE and execution of **R**, **D**, and **DS** will set the variable FALSE. This step activation also executes time-related actions. This row appears one cycle duration as an activation process.

The next row is added as a “*waiting*” row (represented as “(w)”). This indicates the condition to stay until the exit guard is satisfied. Thus, the input condition of this row is the negation of the exit transition guard. This lets the exit transition occur as soon as the exit transition condition is satisfied. The duration of this row differs depending on the qualifier types. For non-time related actions, i.e., **N**, **S**, **R**, **P**, and **P0**, this row can be repeated zero or multiple times, i.e., $[0, *]$, until the exit transition guard is satisfied. Output values depend on the action type (cf. Table 6 – a-e).

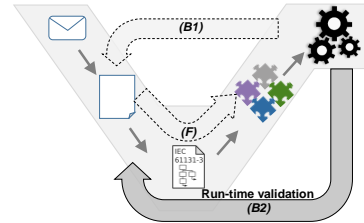
For time related actions, it is trickier because they includes time limit or time delay in their execution. The timing constraints of limit or delay is represented additional row (represented as “(t)”) before the waiting row. In other words, it requires a row to indicate the behavior before the specified time, T_x , elapses. For **L** type action, it retains the value to be set up to time limit T_x but step transition has higher priority. Thus, the timing value in the duration column is to be $[0, T_x - T_c]$. For delaying actions (i.e., **D** and **DS**), this is similar: the value should be set after the time delay T_x . One difference is that the step activated less than T_x is not allowed to avoid the risky case of step deactivation with unexpired delay as warned in [JoTi10]. This forces to have $T_x - T_c$ in the duration column. These time-related actions also have a waiting row. Obviously, the value would be reset for **L** or set for **D**, **DS**. One remark is that this row should be executed at least one cycle, i.e., $[T_c, *]$, for **D** and **DS** cases to set the value after the delay while **L** type action does not require mandatory execution of this row, i.e., $[0, *]$ (cf. Table 6 – f-h).

Additionally, a sequence name column is added to represent the step on the original GTT. One last row in the conversion patterns in Table 6 (whose sequence name is “*Next step*”) will be filled with the successor SFC step information. The last rows in each table (Italic font) is what should appear unless another action is defined on that variable in the following step.

For the initial step of SFC, which is a special case of step with no transition into the step, just one row is required in the form of S2 since there is no entry transition condition. For the selective divergences, branch designation is required since GTT has a consecutiveness characteristic. Therefore, the user needs to specify the intended sequence for the conversion.

5.4.3. Obtaining monitors by the transformation of the specification into IEC 61131-3 function block (B2)

For the identification of both error situations during runtime and unexpected behavior of the technical process or the hardware not covered by the specification, monitoring functions are commonly used. Possible causes for erroneous states of these parts of the system are, for example, wear on the hardware and manipulation



of the technical process or the aPS. To avoid malfunctions or harm to personnel, system errors or faults need to be identified as soon as possible to allow proper fault handling or shut down into a safe state (cf. R-E4). This section of the approach includes the part preliminarily published in [CUVW17].

Monitoring functions are often related to a specific piece of hardware and are directly connected to functions that abstract the interface to that component (driver functions). Information about the hardware behavior is tapped from the inputs and outputs of the drivers. In case of undesired hardware behavior, warnings could be given to the driver itself and the rest of the software system. It is of utmost importance that the monitoring functions work as intended and detect all errors and unknown states of the system to allow the system to give appropriate warnings, handle these errors or bring the system into a safe state by performing an emergency stop to prevent further damage. Therefore, these functions need to be carefully designed and thoroughly tested and require considerable amounts of resources to achieve this reliability. To improve the quality and the coverage of monitoring, a monitoring function can be generated from the specifications in GTT. The generated functions are used to assess the conformance of the system to its specification during runtime and detect situations that are unexpected, not covered by the specification and, thus, untested and unverified. With these objectives, the monitoring functions observe the input and output behavior of a driver function, detecting (a) a divergence between specified and observed behavior of the software and (b) unexpected inputs not covered by the specification.

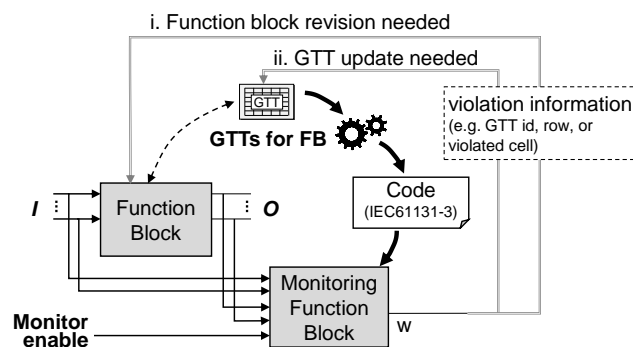


Figure 19: Overview of monitoring function block generation and its usage [CUVW17]

A test table provides a detailed, step-wise definition of the expected behavior of the function block, thus it can be transformed into a code form, which checks the allowed behavior sequences. The monitoring function block generation and functioning schematic is illustrated in Figure 19. Given a function block and (one or more) GTTs as a specification, GTTs can be converted into a monitoring function block. At runtime, this monitoring function block accepts inputs and outputs of the function block as its input as well as an enabling signal for monitoring function block. Then, the system is considered not to perform as specified if the input satisfies the test tables but the output does not (Figure 19 – i). Monitoring also can detect unexpected (i.e., unspecified) situations. If the input does not satisfy any of the specifications, an alarm can be raised to let the system know that this situation is not covered (Figure 19 – ii) in the specification. In that case, to cover the unexpected situation, either one of the existing GTTs needs to be extended or an entirely new GTT could be added.

Behavior of the Monitoring Function and its Generation

Source code for the monitoring function block can be generated by following the table row-by-row since the table describes the deterministic expected behavior of the block. The behavior of the monitoring function block can be represented in the form of a state diagram as displayed in Figure 20. The main purpose of the monitoring function block is to detect violations of the specifications, identify unspecified situations, and raise warnings correspondingly. It monitors whether any of the inputs and outputs violates the GTT description. More specifically, if the input satisfies the table but the output does not, it turns on the warning, i.e., W, the monitor's warning output value, is set to 'WARNING', which means that the output violates the table in the considered case. In other cases where the input violates the table, the warning value is set to 'UNKNOWN', which means it is not specified in the GTT. In case of warnings, this information can be used to review the original specification or the function block as a follow-up activity (cf. Figure 19 – i and ii). Surely, the monitoring function block is to be regenerated from the refined GTT in this case. The status of the function block after this kind of warning depends on the user's choice. If it requires continuing to run with the next input-output pair, it is to be reset, i.e., the state changes to the state of checking the first row. Otherwise, if it requires to hold on until the situation is sorted out by an operator or engineer, it is to be idling until the enable signal is injected again. The usage of these differentiated warning levels may be different from this and the form of the warning may also vary such as logging, displaying on HMI system, and so on. If no violation is detected, no warning is given, i.e., the monitoring output, W, is set to 'OK'.

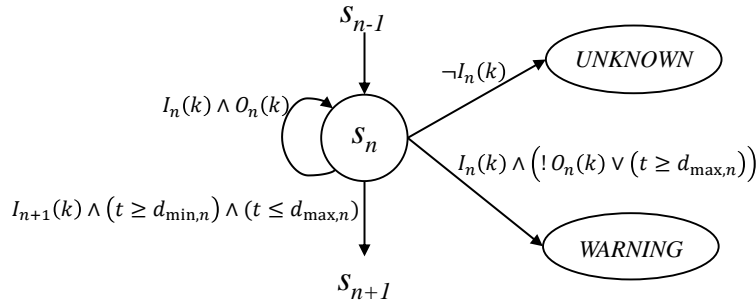


Figure 20: State diagram derived from the generalized test table with S_n relating to the currently active row in a generalized test table [CUVW17]

The automaton starts as soon as an enabling signal for the monitoring function block is detected. Once the monitoring function block is enabled, it expects an input-output pair which satisfies the constraints of the first row of the table. The notation ' $I_n(k)$ ' and ' $O_n(k)$ ' in Figure 20 refers to functions which judge if the current input-output pair (indexed with k) satisfies all input constraints ($I_n(k)$) and output constraints ($O_n(k)$) on the n -th row of the GTT respectively. The allowed number of repetitions, which is described in the duration column, is also checked. The notations ' $d_{min,n}$ ' and ' $d_{max,n}$ ' in the figure refer to the minimum and maximum value of the allowed duration on the n -th row. ' t ' represents the counter value which displays the number of repetitions of the satisfying input-output set. This is initialized when the state changes. If it satisfies these conditions, i.e., input, output, $d_{min,n}$, and $d_{max,n}$, the state transits to the next state with setting W to be 'OK' depending on the satisfied guards which are described on each edge in the figure. For performance reasons, the presented approach is limited to deterministic GTTs in which it can always be decided in which test step the system is. This means that it is not possible that input values satisfy both the input constraints of the current row ($I_n(k)$) and the ones of following row ($I_{n+1}(k)$) if repetition is allowed. Also, if the row allows the infinite number of the repetition (timeout), comparisons of t with $d_{max,n}$ is regarded as true. Once any violation arises regarding inputs or outputs, a state transition is executed to the 'UNKNOWN' or 'WARNING' states.

GTTs can be converted into code blocks with the behavior specified in Figure 20. An IEC 61131-3 Structured Text (ST) code generator can be implemented and could be extended to other standard languages such as Sequential Function Charts (SFC) and Ladder Diagram (LD). First, the function signature is organized. As seen in Figure 19, the inputs and outputs of a function block are connected to the inputs of the monitoring function block. Additionally, an input for the enabling signal and an output for the warning signal are generated. Right below these variable definitions, internal variables are defined, including a variable to distinguish states, a counter to count the allowed duration and all the reference variables within the table. The implementation part of the function is structured using a CASE statement. In each state, it is decided to change into another state or

stay within the same state depending on the comparison results of inputs and the values in the table. Figure 21 shows the basic structure of the code block.

```

FUNCTION_BLOCK monitorFuncion
VAR_INPUT
... (* I/O signals as well as the enabling signal *)
END_VAR

VAR_OUTPUT
... (* warning signal *)
END_VAR

VAR
... (* internal variables and reference values *)
END_VAR
IF EN = TRUE THEN
CASE state OF
...
END_CASE
END_IF
END_FUNCTION_BLOCK

```

Figure 21: Structure of the code block for cases 1 and 2 in IEC 61131-3 (ST)

Operations

Multiple GTTs may be used to describe the behaviors of a block for different situations. They can be combined and converted into one consolidated monitoring function block. At code level, this can be implemented by cascading CASE statements for each GTT. Using multiple tables requires differentiated processing of the individual violation statuses corresponding to each table and these are described within each CASE statement. During runtime, tables that are satisfied with the sequence of the input-output pairs in order and return 'OK' can be regarded as valid. Once any violation of the inputs happens on a table ('UNKNOWN'), as long as no GTT is valid, this table is 'excluded' to imply that the sequence is not aligned with this GTT description. Similarly, any violation of the output might raise a warning only when there is no other valid GTT. Once an activated (valid) table is scanned overall without any violation, it is excluded from monitoring. As long as one or more valid tables remain activated, the monitoring continues and the monitoring status remain as 'OK'. All tables are re-included (exclusion is reverted) if the last activated table finishes its last row. The monitoring function block will generate an overall warning signal *W* from the individual evaluations:

- **OK:** At least one of the GTTs is activated with the input-output pair and all the other GTTs are excluded.
- **WARNING:** At least one of the GTTs is activated and they are only violated with output at some point.
- **UNKNOWN:** None of the tables defines the currently given input, i.e., all tables are excluded.

By using GTTs, which provide expressive behavior specification means for reactive systems and are initially used for formal verification and validating a function block, the monitoring function block can be generated and helps to discover faults as well as unspecified situations during runtime. In addition, this runtime specification-based approach allows the function block to handle and return more practical values and the engineer to correct the function block to be aligned with specifications as well as to refine the specifications themselves. Moreover, this systematically generated code can improve the monitoring quality by avoiding human error during manual programming and reusing existing specifications, notably generalized cases. This supports more efficient engineering since additional specifications for the monitoring block may not be required.

5.5. Summary: table-driven two-way quality assurance process – embedding user-friendly formal specification in the engineering process

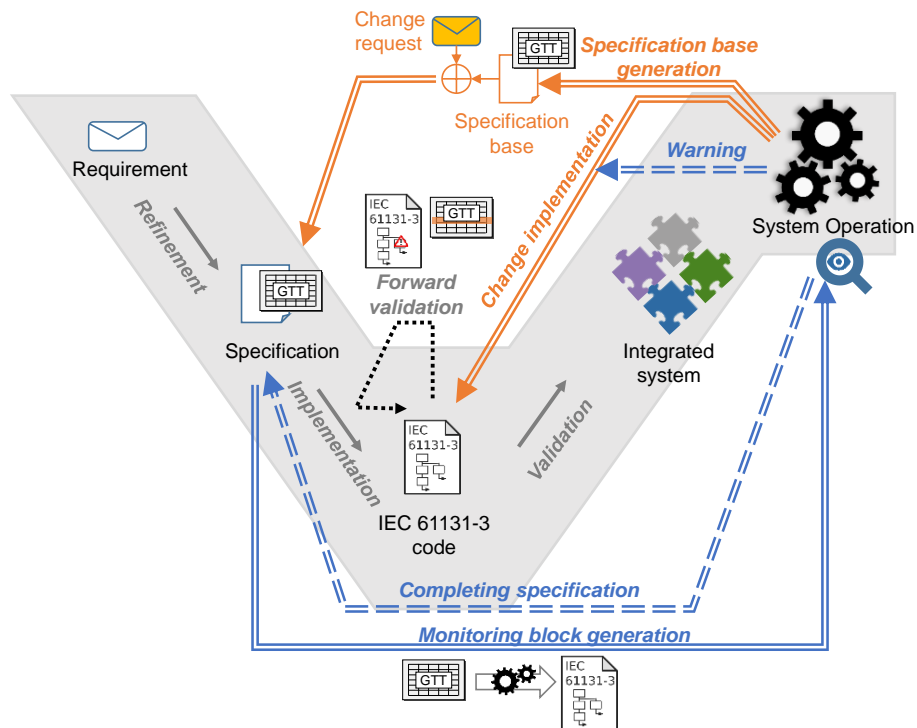


Figure 22: Embedding two-way quality assurance process using GTTs in the control software engineering process

Using the presented formal specification language, i.e., GTTs, the software behavior can be specified and this specification could be used for system software quality assurance activities. This table-based specification could be embedded over the aPS engineering lifecycle pursuing continuous quality control from the development steps to the operation steps (Figure 22). Properties are specified and verified during the development until the required properties are satisfied by the

developed software. The integrated system composed of the validated modules is under the commissioning phase and on the operation. Even validated in the earlier phase, developed systems face changes during commissioning and operation to fix bugs or to cope with the changed requirement. After the change is implemented, the change intention has to be validated and side effect freeness has to be verified. For these activities, change specification is required and a method to obtain the specification base to be given to the automation engineer was shown. Additionally, besides expected faults and failures which could be planned to prevent during the implementation, unexpected and unintended behavior could also be captured and handled correctly by the run-time monitoring. Based on the formal specification of the target function block to be monitored, an approach to generate the monitoring function block systematically was presented. The generated monitoring block is capable of detecting errors generated during the manual programming with respect to the specification and of diagnosing the cases which have not been considered during the planning and implementation. Using this formal specification based two-way quality assurance process, the correctness of the control software as well as the specification could be improved in an efficient way.

6. Implementation

The presented approach involves and spans over the engineering processes from the requirement specification to the operation and also to the change management beyond the operation. Tools that support to apply the approach systematically would enable it to be embedded in the engineering process, accommodating effectiveness and efficiency. Module developers define the module level specification, assign it as an attribute of the module through the editable tool, and run the verifier (i.e., model checker) to check the conformance of the developed module to the defined specification within the integrated development environment (IDE). Through the tool, consistent and unified information could be stored and exchanged between the automation engineers and the generated information regarding validation could be connected to the original sources of program code and the specification, compared to the manual documentation and process. (cf. R-E6, R-U1)

This functionality as a prototypical tool has been implemented in the form of the add-on in CODESYS (cf. Section 4.4) through the extendable software developer toolkit (SDK) for add-ons. As the IDE supports a great number of devices worldwide independent from PLC manufacturers, the prototypical tool would be beneficial if it is compatible with this major IDE. The add-on implementation presented in this chapter was done with a support of a professional programmer.

In the formal verification point of view, the starting point of the activity chain is to generate the formal specification in a way that could be processed further programs systematically. On the other hand, the program code will be implemented by the developer. These are to be supported by the IDE for their generation, and are to be delivered to the backend verifier. GETETA [Weig19] is the backend tool to generate the automata for the GTT and the IEC 61131-3 code to call the model checker (nuXmv). Model checking results are retrieved by the GETETA and delivered to IDE and presented to the user. Figure 23 shows the overview of the toolchain with specifying the scope of this thesis.

Following the introduced concept, the implementation consists of three parts: a) table editing part, b) calling the verifier transporting the verification target and the properties, and c) the result representation. Each will be described in the following sections with respect to the implemented features and its utility in the presented approach.

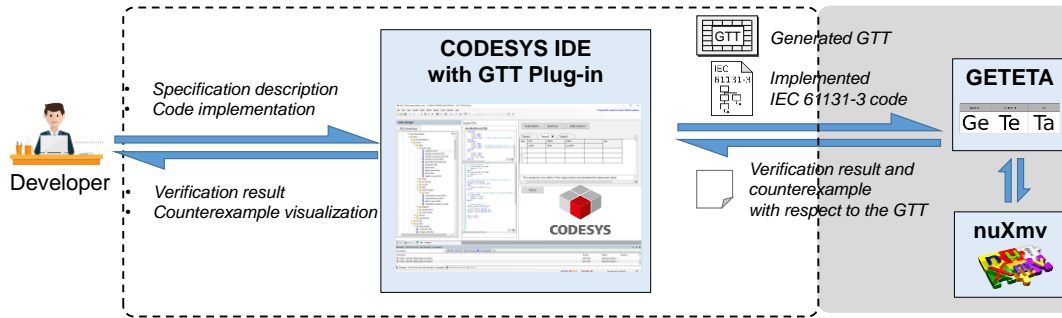


Figure 23: Prototypical toolchain of the formal verification using GTT (scope of the thesis is specified by the dashed line; the gray area is discussed in [Weig21])

6.1. Overview

The tool, named table-based verification manager (TBV manager), is to be added within an existing CODESYS project as an add-on object. When the object is loaded, it shows the object plane consisting of three different parts (Figure 24). The whole code tree within the project is shown in the code directory view. There is a code plane to show the selected code from the code directory as the target for the verification. Corresponding GTTs are to be edited in the GTT editor plane to load and manipulate the properties in GTT. Once the code and the specification in GTT is ready, user could click the verify button to execute the verification by calling the backend model checker inducing the target code and the property. When the verification fails, it generates an additional window to represent a counterexample. The development of the tool was done in Microsoft Visual C# within the Microsoft .NET framework 4.8.

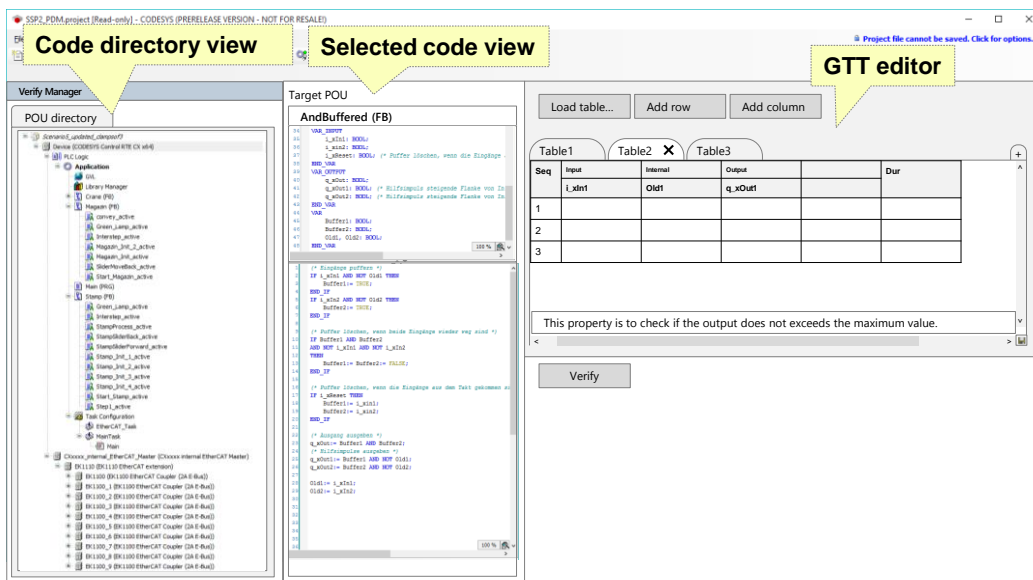
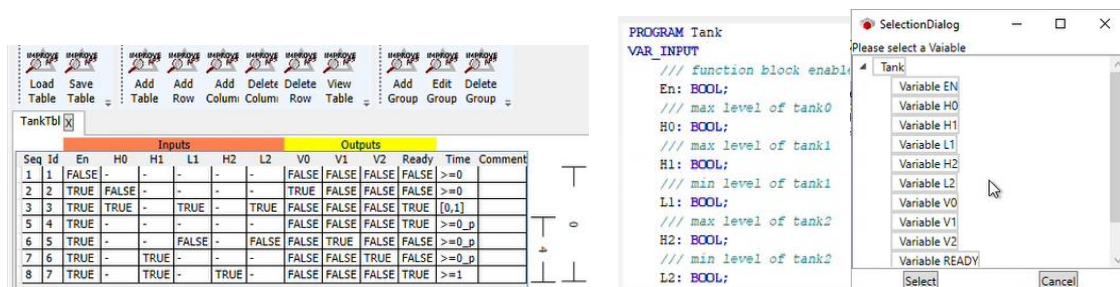


Figure 24: Add-on overview

6.2. Generating and editing the generalized test tables

The user selects the target code to verify from the code directory view. CODESYS projects can be accessed through the *POUObject* package, provided by CODESYS, to support further extension add-on development since the project is not directly accessed to get the software code as text (cf. [Code20c]). Using the package library, the software code is recognized as IEC 61131-3 code instance and retrieved as a text object.

The table editor section (Figure 25 – a) shows the basic table structure of input and output at the beginning. This section was built up based on the *DataGrid* class provided by the Windows Presentation Foundation (WPF) subsystem and usable in .NET framework (cf. [Micr18]). The grid and controlling buttons consist of a separate window section for the GTT editor. As the user of the editor wants, each input and output of the POU is automatically extracted (from the target POU) and shown as possible options to be added as columns when *Add Column* menu is clicked so that the table contents are consistent with the defined variables. The variables are also accessed through the CODESYS project POU object and retrieved as done in the project tree. Depending on the type of variable (input or output), the selected variable column is generated either input section or output section (Figure 25 – b). The sequence could be added by selecting *Add Row* menu and selecting *Add Group* in case of grouping several sequences. For a specific row or group, the duration value is inserted in *Time* column to designate the allowed time information. The grouping aggregation symbol appears outside of the table on the right not to consume too much space for the duration section. Additionally, the *Comment* cell appears for each row for remarks of additional information or an important description regarding the row. Rows, columns, and the groups can be deleted with the corresponding buttons. Each cell value could be directly editable and then saved by clicking *Save Table* menu. GTT can also be loaded from a file by *Load Table* menu. This allows to reuse the existing properties as they are or after revising them. The loaded file is parsed following the defined grammar (cf. [Weig19]) through the generated parser from ANTLR4 [Antl20].



(a)

(b)

Figure 25: Editing GTT – (a) GTT editor plane, and (b) guided variable selection

6.3. Connecting to the verifier

Once the program code part is ready to be checked regarding its conformity to the given specification in GTT, TBV manager object is to communicate with the external model checking by transferring the given code part and the properties. The ultimate model checker is nuXmv and this is called within GETETA [Weig19] which is in charge of the symbolic model generation of the code and the properties in GTT. To deliver the objects to be processed in GETETA (i.e., program code in IEC 61131-3 and the property in GTT) from TBV manager, the selected POU and the property is saved as a form of text file in a specific directory so that the GETETA can read it. Once the *Verify* button is clicked, the corresponding files are stored in the specific directory and GETETA is called in the command line. Executing GETETA is done through the *System.Diagnostics.Process* class to call it as if it is called in the command line interface. After it starts, it waits for the termination of the backend program (i.e., GETETA for here) execution.

6.4. Verification result representation

The verification result is either conformity proved, in case of the program code satisfies the given property specification, or failed, in case of any violation of the program code execution result (or some situation expected as execution results) to the property observed.

In the case of conformed, the program code could be said as well-implemented as intended so it could be deployed if necessary. However, in the violation case, the program code part of the program execution flow which causes the violation should be captured and revised. Since the model checker provides counterexamples, which is the execution traces up to the violation point, users can follow the input/output traces step by step along with the program code. As presented in Section 5.4.1, the value tracking provides the chances of debugging. This could also be done manually; however, it is a tedious and vulnerable (to mistakes by hand) task when the signal trace is long, or many variables are involved in the trace. Therefore, an efficient way to represent the counterexample to support correcting the code is provided in this tool. In TBV manager, the log file generated by GeTeTa is read if it is proven as not conforming. From the log file, the execution traces of input and output values are extracted. This information extraction is also done by the parser, specifically created for GeTeTa output log file. The extracted trace information is presented in the timing diagram (Figure 26). For this timing diagram implementation, DataGrid was basically used with an appropriate presentation for the timing diagram (cf. grid in table representation in Section 6.2). Each variable has its own lane to present the value sequences which is presented horizontally with the unit of the cycle. Thus, each tuple of input and output values satisfies at least one table row in GTTs. From there on, the following sequences are also satisfying the following

rows in the GTT. This means that each sequence step can be mapped to the table and the log includes the information for each value set about on which sequence the model checker decides that the property is not satisfied by the execution. Therefore, it is implemented to allow select a specific set of sequence and then the corresponding table row is also highlighted. The end of the sequence would be the point of violation. Therefore, the user can backtrack the value sequences from the violation point if the origin of that violation has occurred in the previous cycle(s).

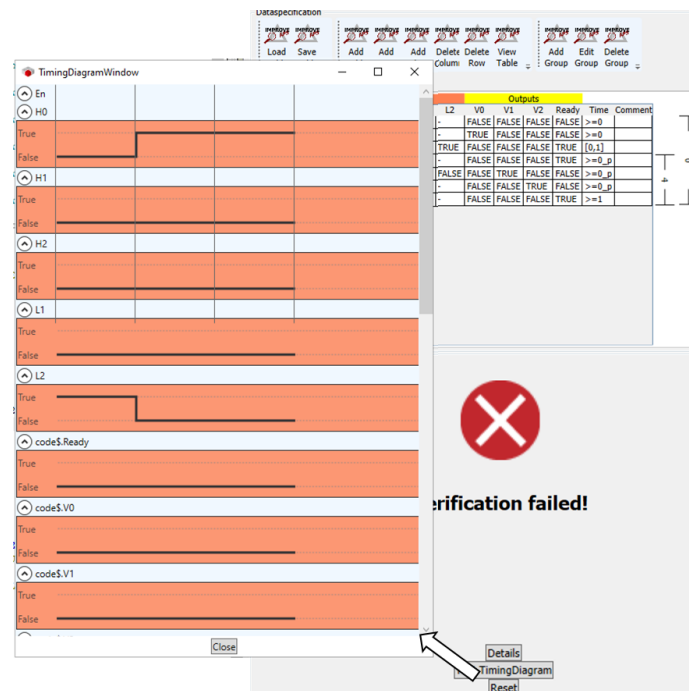


Figure 26: Displaying counterexample in case of verification fail

7. Evaluation of the approach

The presented approach shall improve the efficiency of the quality control activities within the aPS control software engineering. This chapter is to present the evaluation of the proposed specification language, i.e., GTT, with respect to the requirements (cf. Table 7). Three types of evaluation experiments are conducted and presented. First, its applicability and feasibility on the aPS system and its engineering activities is investigated using a community demonstrator in Section 7.1, mainly focusing on the requirements regarding the characteristics of technical systems and

Table 7: Requirements and the corresponding evaluation

ID	Brief description	Section and type (<i>a,e,i</i>)* of evaluation
R-T1	Presenting the relationship between input and output.	7.1.1 (<i>a</i> – single component module), 7.1.2 (<i>a</i> – multi-component module)
R-T2	Presenting the state changes including the conditions and timing constraints	7.1.1 (<i>a</i> – single component module), 7.1.2 (<i>a</i> – multi-component module)
R-T3	Abstracted value range	7.1.1 (<i>a</i> – single component module), 7.1.2 (<i>a</i> – multi-component module)
R-T4	Supporting discrete event processes	7.1.1 (<i>a</i> – single component module), 7.1.2 (<i>a</i> – multi-component module)
R-T5	Supporting cycling execution representation.	7.1.1 (<i>a</i> – single component module), 7.1.2 (<i>a</i> – multi-component module)
R-T6	Compatible to the software in IEC 61131-3	7.1.1 (<i>a</i> – single component module), 7.1.2 (<i>a</i> – multi-component module)
R-E1	Formal specification language shall be developed.	7.1.1-7.1.4 (<i>a</i>) 7.3 (<i>i</i> – expert interview)
R-E2	The test cases shall be instantiated from the specification.	7.1.1 (<i>a</i> – single component module), 7.1.2 (<i>a</i> – multi-component module)
R-E3	Supporting verification based debugging	6 (Implementation)
R-E4	Supporting monitoring block generation	7.1.4 (<i>a</i> – monitoring block generation)
R-E5	Supporting change validations	7.1.3 (<i>a</i> – specification generation)
R-E6	Supporting automation engineers documentation and exchanging activities	7.1.1-7.1.4 (<i>a</i>) 7.3 (<i>i</i> – expert interview)
R-U1	Tool supporting	6 (Implementation)
R-U2	Understanding specifications	7.2 (<i>e</i> – questionnaire), 7.3 (<i>i</i> – expert interview)
R-U3	Creating specifications	7.2 (<i>e</i> – questionnaire)
R-U4	Learning specifications	7.2 (<i>e</i> – questionnaire)
R-U5	Scalability of specifications	7.2 (<i>e</i> – questionnaire), 7.3 (<i>i</i> – expert interview)
R-U6	Satisfaction with the specification approach	7.2 (<i>e</i> – questionnaire), 7.3 (<i>i</i> – expert interview)

**a*: application case presentation (demonstrator), *e*: empirical study, *i*: industry feedback

their engineering processes. Second, since the major motivation of the approach is to provide the user-friendly formal specification language, usability survey based on the user perception was conducted among the students from the mechanical engineering department, presented in Section 7.2. Third, interviews with the experts from the industry have been executed and presented in Section 7.3 to justify the practical applicability.

7.1. Feasibility analysis through application case studies

In this section, the case studies provide the application of the GTT using community demonstrator (including its components) scenarios to illustrate the applicability and feasibility of formal specification in GTT. Considering the engineering process of the aPS control software, four different cases were selected to demonstrate the corresponding engineering activities. The use cases are selected considering different level of ISA-88 [IEC97], i.e., control module and equipment module. The first use case (UC1) is a function block to control a pneumatic cylinder as a single component module (or control module). During or after developing a module library function block (*module developer*), formal specification is to be existed to verify the behavior accordingly. The second use case (UC2) is a sorting module consisting of component modules (or equipment modules). This is assumed to be implemented by *application engineers*, composing existing library modules. The third use case (UC3) demonstrates how the presented approach can support the change vali-

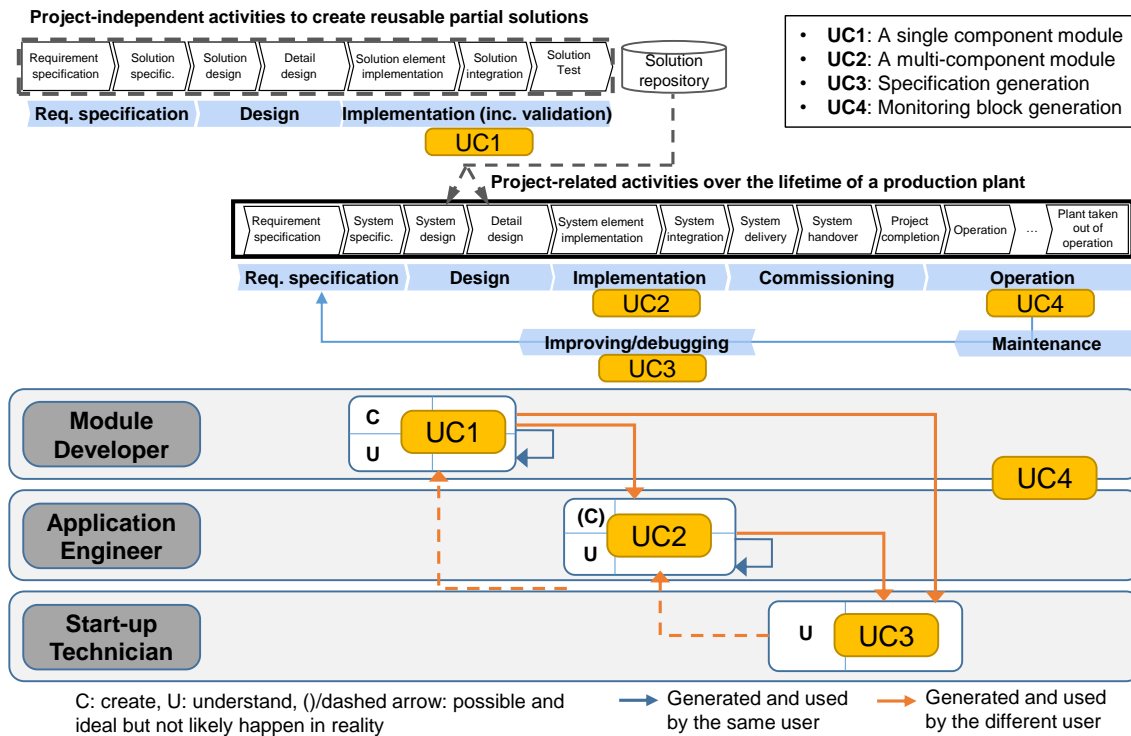


Figure 27: Use cases mapped on the engineering processes

dation to infer the specification. While the current software is running, a change is to be implemented due to the change request of the functionality of the existing (being already implemented) module part in UC2. The fourth use case (UC4) shows how monitoring a function block could be generated from the specification in GTT by the presented approach. Overall use cases will envision how the information exchange and transformation could be supported using and through the specification, created in the presented language.

The demonstrator that is used to present along the following section is Pick & Place Unit (PPU) [VLFF14] and its extended version (xPPU) [VOBS18], a lab-size manufacturing plant demonstrator to benchmark evolution scenarios for aPS with a recent extension of that demonstrator for its functionality and safety features (Figure 28). It has been established within the DFG priority programme SPP 1593 in Germany as a common case study for evolution in plant and machine automation. Although the xPPU is quite simple, it realizes the basic functionalities representative in intralogistics systems as identified by Spindler et al. [SAVF17]. The xPPU consists of four composite modules basically: a stack to load work pieces initially for the whole operation, a stamp to demonstrate manipulations conducted on the work pieces, a sorting module to store the work pieces according to its given mission, and a crane as a transporter to transfer work pieces between the other modules. A wide range of evolution scenarios have been defined and implemented for the (x)PPU with different motivations (e.g., changing requirements, fixing failures, and unanticipated situations on-site).

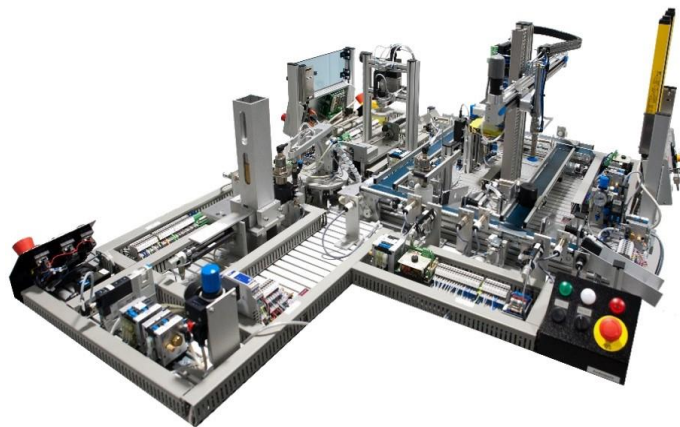


Figure 28: Community demonstrator: extended Pick&Place Unit

7.1.1. Behavior description of a single component – in the viewpoint of module developer (use case 1)

In industrial use, pneumatic controls are common, especially for a fixed distance reciprocation of objects [BuVy17]. Pneumatic cylinders cause motion by extending and retracting their pistons

using pneumatic controls. Here, a single-acting cylinder (also known as a monostable cylinder) and its behavior description in GTT is introduced.

A single-acting cylinder includes one pressure port, to control the extension, and often has a spring, to make the piston return to the base position using the elasticity (Figure 29). The piston position is indicated by binary sensors at each end (e.g., with limit switches or inductive sensors) to indicate fully extended or retracted. This component has been used within the xPPU in stack (to move out one work piece from the magazine), stamp (for stamping on the work piece), and sorting module (to separate a work piece from the conveyor to the storing ramps).

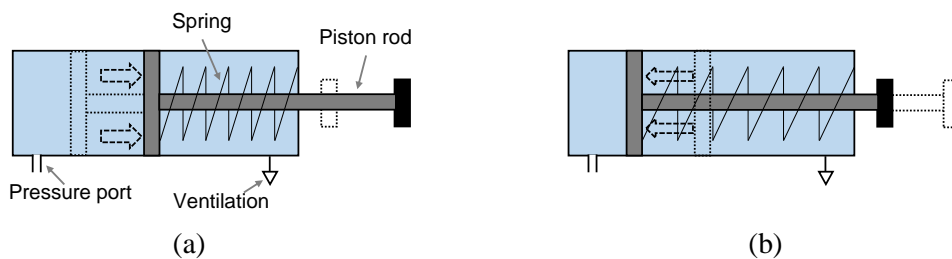


Figure 29: Single acting cylinder: (a) extension and (b) retraction

The module is to be developed as a library, to have a unified code for the efficiency of development, maintenance and further updates (i.e., changes), implementing the extending and retracting behaviors with timing constraints of 500ms for extending and 200 ms for retracting. If it is not extended in 500 ms for the extending behavior, it should force to give the retracting signal for 200 ms. If it is not retracted after that, it should give the alarm. If the cylinder is not retracted in 200 ms for the retracting signal, it should give the alarm right away.

The extending behavior is to be described in GTT (Figure 30). The extended and retracted position is represented with the input variable E and R , respectively. The extending signal and the alarm is represented with the output actuating variable Ext and $Alarm$. When it is to be extended (e.g., called through $extend()$), the output signal is to be activated and the piston shall reach the extended position within 500ms (Figure 30 – a, #1), retaining the extending signal onwards (Figure 30 – b, #1). In case of abnormal extending behavior (Figure 30 – b), which means that the piston is indicated as “not extended” (i.e., $E = \text{False}$) for 500 ms (Figure 30 – b, #1), the cylinder is forced to retract for 200 ms (Figure 30 – b, #2). After that, depending on the piston position, a value for the variable $Alarm$ is decided: $Alarm$ to be true if it is still not retracted (i.e., $E = \text{False}$), otherwise false (Figure 30 – b, #3). In a similar way, retracting behavior could also be represented in GTT (Figure 30 – c). The abnormal retracting behavior could be described as simpler than faulty extending (Fig 7-1d). When the piston is indicated as “not retracted” for 200 ms even after the retracting signaling (Figure 30 – d, #1), it just have to alarm (Figure 30 – d, #2).

Seq.	Input		Output		Duration
	E	R	Ext	Alarm	
1	F	–	T	F	<500msec
2	T	F	T	F	–

(a)

Seq.	Input		Output		Duration
	E	R	Ext	Alarm	
1	F	–	T	F	500 ms
2	–	–	F	F	200 ms
3	–	a	F	!a	>=1

(b)

Seq.	Input		Output		Duration
	E	R	Ext	Alarm	
1	–	F	F	F	<200msec
2	F	T	F	F	–

(c)

Seq.	Input		Output		Duration
	E	R	Ext	Alarm	
1	–	F	F	F	200msec
2	–	–	F	T	–

(d)

Figure 30: Single acting cylinder behavior description considering the timing constraint and corresponding alarm signal: (a) normal extension behavior within 500 ms, (b) abnormal extension behavior with the piston not reaching the expended position within 500 ms, (c) normal retraction behavior within 200 ms, and (d) abnormal retraction behavior with the piston not reaching the retracted position within 200 ms

Feasibility of using GTT for the specification of single component modules has been proven. As it is assumed that the module developers are skilled programmers, the usability is evaluated through the empirical (in Section 7.2) study and through expert interview (cf. Section 7.3). It is expected that all the modules would be specified in the future with this stepwise manner to achieve formally specified module over the time.

7.1.2. Behavior description of a multi-component module – in the viewpoint of application engineer (use case 2)

The target scenario in this case study is Sc10d of xPPU scenarios, which was developed to demonstrate rather small step evolution compared to the major scenario change. The target module of the scenario consists of pusher and conveyer modules and the scenario is to evolve to Sc11 in the next section (Section 7.1.3) to demonstrate a small change case.

In Sc10d, the main focus is put on the sorting module (Figure 31). Work pieces are transferred to the sorting module at the end of the processes to be stored in the intended storage ramps depending on types. Once the work piece arrives at the initial point of the conveyor, conveyor starts to move. When the work piece is detected in the optical sensor area, it stops shortly to detect the work piece type. And then it continues to move for a specific time, depending on the target ramp: Ramp1 or Ramp2. If the detected type is black plastic, it will be pushed out by Pusher1. Other work piece types (i.e., white plastic and metal) are assorted in Ramp2 pushed out by Pusher2.

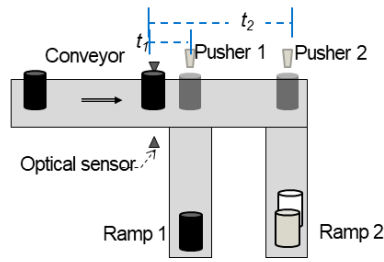


Figure 31: Sorting module: Black WPs are sorted in Ramp1 and the others in Ramp2

This sorting module is multi-component module and each subcomponent have its own functional implementation most likely as a library function, e.g., pneumatic cylinder module as seen in the previous use case. Thus, the application engineer has to understand each module's functionality to utilize them. In this use case, the application engineer is supposed to build up this sorting module. After deciding to put a cylinder component, he/she will have to know how the selected library function block behaves exactly, which can be done by understanding the specification generated by a module developer. Being aware of the module behavior, the application engineer organizes and rearranges components with some glue code. Also, the behavior of this module is also supposed to be described by this application engineers both to apply formal verification and to document the implemented module (cf. R-E6). An excerpt of the Pusher2 behavior in GTT is illustrated in Figure 32. If the target is set to Ramp2 (#12), Pusher2 will be activated (#17).

Seq.	Input						Output			Dur. (ms)	Remarks
	WPsorted	Motor Stopped	R1	R2	P2E	P2R	Motor. Stop	P2.Ex	WPsort		
(omitted)											
11	-	-	!(R1∨R2)	!(R1∨R2)	-	-	F	F	F	[0,*]	OnConv(w)
12	-	-	F	T	-	-	F	F	F	10	OffConv2
13	-	-	-	-	-	-	F	F	F	1730	OffConv2(t)
14	-	F	-	-	-	-	T	F	F	[10,*]	OffConv2(w)
15	-	T	-	-	-	-	T	T	F	10	ExtP2
16	-	-	-	-	F	-	T	T	F	[0,*]	ExtP2(w)
17	-	-	-	-	T	-	T	F	F	10	RetP2
18	-	-	-	-	-	F	T	F	F	[0,*]	RetP2(w)
19	-	T	-	-	-	T	T	F	T	10	SetWPsort
20	F	-	-	-	-	-	T	F	-	[0,*]	SetWPsort(w)

*R1: "WP sorting in R1", R2: "WPsorting in R2", P2E: "Pusher2 is extended", P2R: "Pusher2 is retracted", P2.Ex: "Extend the Puser2"

Figure 32: Excerpt of GTT for the sorting module behavior

As seen, the GTT is applicable to describe multi-component modules. Whether it is easy or efficient to apply by application engineers will be discussed in Section 7.2. It is expected that especially for very critical part of such applications (e.g., ones in the market segment of medical related systems) or part of applications that will most probably frequently used, application engineers would be willing to document it as in GTTs and also even it should be encouraged cost-wise; and, consequently, the percentage of specified applications on this level will increase.

7.1.3. Obtaining changed specification based on the preliminary specification generated (use case 3)

In this use case, it is assumed that a change request arrives on the existing implementation of Sc10d (introduced in Section 7.1.2). Instead of the behavior of sorting out the black work pieces only, it is additionally required to sort out white ones to be Ramp2 and the metal ones to be additional storage, Ramp 3, at the end of the conveyor, meaning that black, white, and metal are all separated (Figure 33). In this scenario, an additional inductive sensor is to be attached to distinguish metal WP. In the control software, the assorting function needs to be revised to make the metal WPs gathered in Ramp3.

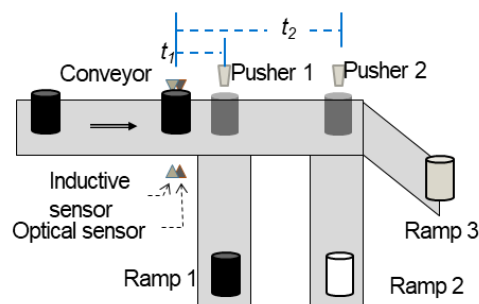


Figure 33: Sorting module (evolved from Figure 33: Black WPs are sorted in Ramp1 and the others in Ramp2 [CWUB18b])

When the application engineer is assigned such a task, he/she would revise the control code as requested, specifically regarding the control of Pusher2 – not to push all the work pieces. After that, he is supposed to validate the behavior and to document the description (cf. R-E5, R-E6). When it is planned to be verified using formal verification to save the testing time with already running hardware, its formal specification is necessary (e.g., in GTT), which would be more highlighted for its necessity by regulations in case of safety-related industry sectors, e.g., medical devices. If there has been already existing a formal specification regarding the behavior of Pusher2, it could be used by the engineer to obtain the new behavior as a base. If not, however, it has to be created manually from scratch. In this case, the preliminary specification can be generated by converting existing code using the approach introduced in Section 5.4.1. (One remark here that the code implementation activities and specification generation activities are to be independent from each other so that the specification does not reflect what is already implemented but what is intended to be implemented. Recall Figure 18: code revision and partial model generation are separately done and then used in the verification).

The language used for this implementation is assumed as IEC 61131-3 SFC and the earlier code is implemented with two different cases (Figure 34): target ramp is Ramp1 ($R1=True$) or target ramp is Ramp2 ($R2 = True$). Since the change is applied to the Pusher2 behavior, the interest of

the engineer would lie in the relevant part of the code (dashed line in Figure 34) and this part needs to be revised by the developer. For the changed behavior description, the base of it could be obtained in the formal language (here in GTT) by converted into GTT and this will be in the form as seen in Figure 32.

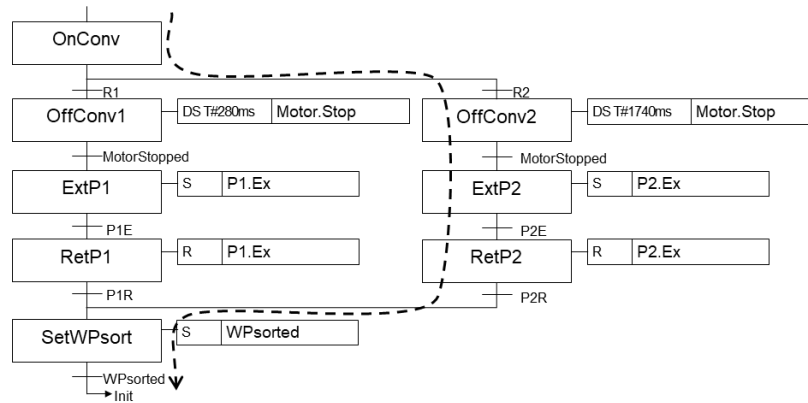


Figure 34: Part of SFC for sorting behavior of conveyor (Sc10d) with a specific behavior branch highlighted with a dashed line [CWUB18b]

After obtaining the GTT base to describe the changed behavior, the engineer would revise the given table to reflect the changed behavior regarding the following points (cf. Figure 35):

- A new input variable, e.g., “R3”, is added equivalently to the “R1” and “R2” which indicates in which ramp the WP needs to be sorted.
- The exit transition condition of the OnConv is revised and also added for the new variable to consider R3. (Seq#11)
- The conditions on which the new functionality is executed are described, e.g., “if R3 is true” in this scenario. (Seq#12)
- Since the conveyor should transport the WP to the Ramp3, the time duration of the conveyor motor execution must be revised to be prolonged properly, e.g., 2990ms. (Seq#13)
- The pushers are nothing to do any more with metal WPs, or Ramp3, the rows related to the pushers are removed from the GTT. (Seq#15-18)
- The condition to decide if the WP is sorted needs to be revised since pusher retraction does not mean the WP is sorted any more, but stopping of the conveyor means instead. (Seq#19)

Seq.	Input					Output			Dur. (ms)	Remarks
	WP sorted	Motor Stopped	R1	R2	R3	Motor. Stop	P2.Ex	WPsort		
<i>(omitted)</i>										
11	-	-	!(R1∨R2∨R3)	!(R1∨R2∨R3)	!(R1∨R2∨R3)	F			[0,*]	OnConv(w)
12			F	F	T	F			10	OffConv2
13			-	-	-	F			2990	OffConv2(t)
14		F				T			[10,*]	OffConv2(w)
<i>(Pusher related behavior is removed)</i>										
15		T						T	10	SetWPsort
16	F	-			-				[0,*]	SetWPsort(w)

*P2E, P2R omitted

Figure 35: The GTT for the revised functionality achieved by editing the automatically generated one. Revised part is highlighted in dark gray [CWUB18b]

It has been shown that specification base in GTT could be generated from IEC 61131-3 SFC software code. The automation engineers, especially for the ones who handles changes and are responsible for validation of the change (many times start-up technicians during the commissioning phase, sometimes application engineers for adaptation during the application generation, and also module developers to generate variants or versions of the library function), benefit from the base specification generation out of the existing software code by not preparing the specification from scratch. The obtained GTT will be used as a formal specification to verify the intended revision of the code, as well as for the documentation purpose. The feedback from the industry experts regarding the usefulness of the specification generation will be discussed in Section 7.3.

7.1.4. Monitoring of the target function block – automated generation of the monitoring block (use case 4)

In this use case, it is assumed that the cylinder behavior is supposed to be monitored. Since the one introduced in Section 7.1.1 includes the monitoring function within its module implementation (thus, additional monitoring block is not required for it), a cylinder without monitoring functionality is introduced here (cf. R-E4). Furthermore, it is an extended version of the previous cylinder, i.e., a double-acting cylinder, to add some more complexity.

A double-acting cylinder (also known as a bistable cylinder) extends and retracts using hydraulic power having two pressure valves, one on each end (Figure 36), to provide multiple stable positions controlled. This component has been used within the xPPU in the stamp for the work piece

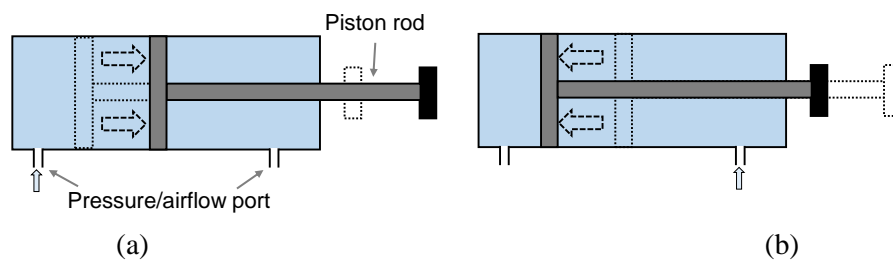


Figure 36: Double acting cylinder: (a) extension and (b) retraction

slider as it has two active positions (retracted: under the stamp, extended: under the crane gripper). In this example, piston position is indicated by an analog sensor for more precise position detection.

The behavior of the module is similar to the single-acting cylinder, i.e., extending and retracting. The extension behavior is described here: when the extend function (*extend()*) is called with the piston positioned at the base position, the piston should leave within 10 ms and then arrive Position 2 within 500 ms. Specifically, different behavior is considered: ‘extending after resetting’. When the extension function is called with the piston positioned not at the base position but in between the base position and working position, the piston should be fully retracted and the extension behaviour should start. The behavior description in GTT corresponding to this resetting added extension is given in Figure 37 – a (#1-3) is the resetting behavior and #4-6 is the extension behavior. Retracting is done simply: once to be retracted, valve at the working position is open to make the piston move to the based position (Figure 37 – b). During this behavior description in GTT, a typical IEC 61131-3 library function SEL(x,a,b) is used, meaning “If x is true, then a. Otherwise, b.”

From the specified GTTs, a monitoring function block can be generated for this cylinder module to indicate whether it is working as intended, incorrect (WARNING), or unexpected (UNKNOWN) using the approach introduced in Section 5.4.3. Besides the optional monitoring enabling signal, the input set of the monitoring function bock includes all the input and output variables of the function block, which are TarInAPos, TarInExtend, TarOutVal1, TarOutVal2, TarOutPos (Figure 38).

Seq.	Input		Output			Duration
	A_Pos	Extend	Val1	Val2	Pos	
1	[3,97]	TRUE	FALSE	FALSE	NONE	10ms
2	<=A_Pos[-1], >=3	TRUE	FALSE	TRUE	NONE	<= 500ms
3	<3	TRUE	FALSE	FALSE	POS1	10ms
4	>=A_Pos[-1], <3	TRUE	TRUE	FALSE	POS1	<=10ms
5	>=A_Pos[-1], >=3, <=97	TRUE	TRUE	FALSE	NONE	<=490 ms
6	>97	TRUE	FALSE	FALSE	POS2	-

(a)

Seq.	Input		Output			Duration
	A_Pos	Extend	Val1	Val2	Pos	
1	-	FALSE	FALSE	FALSE	=SEL(A_Pos<3, POS1, SEL(A_Pos>97, POS2, NONE))	10ms
2	>=A[-1], >=3	FALSE	FALSE	TRUE	=SEL(A_Pos<3, POS1, SEL(A_Pos>97, POS2, NONE))	<=490 ms
3	<3	FALSE	FALSE	FALSE	POS1	-

(b)

Figure 37: Extension behavior description with the resetting function (a) extending after resetting and (b) retracting

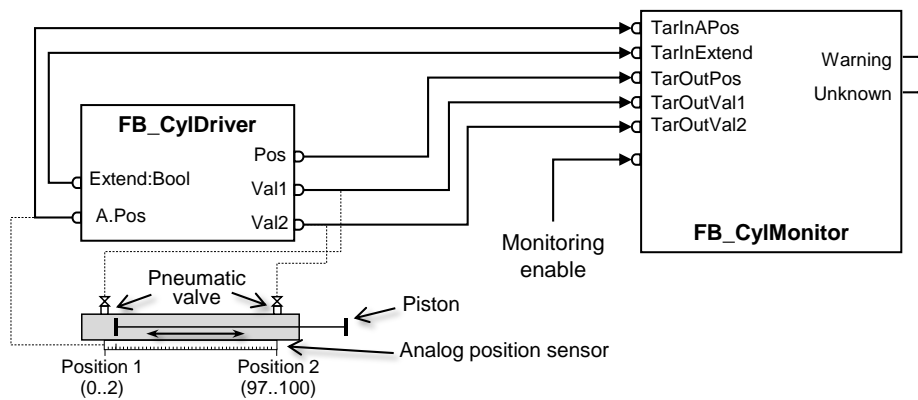


Figure 38: A cylinder driver function block and its monitoring function block [CUVW17] (reproduced)

Note that the generated monitoring function block (Figure 39) is assumed to be re-enabled after a warning or unknown signal is resolved, meaning the monitoring would be active only when it is just enabled or indication signals (i.e., warning and unknown) is not active (line 23). When it is just re-enabled, all the GTT checking information as well as indication signals are initialized (line 27-30). After that, each GTT is checked regarding the corresponding input, output, and its duration; duration is controlled by timers. If the expected input is entered, the output is checked and the warning indication is decided (line 32-66). As two GTTs exist, each GTT is checked and then it is decided whether the state corresponds to the GTT, and on which row if so. Any violation (single or multiple) will raise a warning signal (variable W). If the input is not as expected, the unknown signal will be activated since the case is not handled within the GTT; if all GTTs are not handling such input/output state, the monitoring bloc raises an unknown signal (variable UNK , cf. line 68-70).

Feasibility of monitoring block generation has been shown that the code of the monitoring block could be systematically generated also for multiple GTTs. Still, the performance for scalability of the monitoring block depending on the number of GTTs and the number of concurrently active rows over GTTs (with corresponding timers), should be further researched as well as monitoring the non-deterministic states (rows).

<pre> 00: TYPE E_Warn: (OK,WARNING,UNKNOWN); END_TYPE 01: FUNCTION_BLOCK diagFunction 02: VAR_INPUT 03: TarInAPos : USINT; TarInExtend: BOOL; 04: TarOutVal1: BOOL; TarOutVal2 : BOOL; TarOutPOS: BOOL; EN : BOOL; 05: END_VAR 06: VAR_OUTPUT 07: W : BOOL; 08: UNK : BOOL; 09: END_VAR 10: VAR 11: state1: INT := -1; counter1: INT := 0; a1: INT; 12: state2: INT := -1; counter2: INT := 0; a2: INT; 13: OldEN: BOOL; 14: ActRow: ARRAY [1..2] OF INT:= [2(0)];//which row is targeted to check for each GTT 15: CntRow: ARRAY [1..2] OF INT:= [2(0)];//which row is targeted to check for each GTT 16: TimerRow: ARRAY [1..2] OF Timer;//which row is targeted to check for each GTT 17: t : TON; 18: END_VAR </pre>	<pre> 20: /*It has been enabled but W = TRUE from the previous cycle*/ 21: 22: IF EN = TRUE THEN 23: IF (W = TRUE OR UNK = TRUE)AND OldEN = TRUE THEN 24: /*warning to be handled with disabling monitoring and to be enabled again*/ 25: ELSE 26: IF OldEN = FALSE THEN /*initialization necessary*/ 27: W = FALSE; 28: UNK = FALSE; 29: ActRow[1] = 1; 30: ActRow[2] = 1; 31: END_IF 32: /*check GTT1*/ 33: CASE ActRow[1] OF 34: 1: /*Seq1 of the Extension behavior GTT1*/ 35: IF (TarInAPos>=3) AND (TarInAPos<=97) AND (TarInExtend=TRUE) OR NOT t.Q THEN 36: /*input condition of Seq1 in GTT1*/ 37: t(IN:=TRUE, PT:=T#10MS) 38: IF NOT (TarOutVal1 = FALSE AND TarOutVal2 = FALSE AND TarOutPOS = NONE) THEN 39: ActRow[1]=0; 40: W = TRUE; 41: END_IF 42: ELSIF t.Q THEN 43: ActRow[1]=2; 44: ELSE 45: UNK = TRUE; 46: END_IF 47: ... 48: END_CASE 49: /*check GTT2*/ 50: CASE ActRow[2] OF 51: 1: 52: IF TRUE OR NOT t.Q THEN 53: /*input condition of Seq1 in GTT2*/ 54: t(IN:=TRUE, PT:=T#10MS) 55: IF NOT (TarOutVal1 = FALSE AND TarOutVal2 = FALSE AND 56: SEL(TarInAPos<3, POS1, SEL(TarInAPos>97,POS2, NONE)) THEN 57: ActRow[1]=0; 58: W = TRUE; 59: END_IF 60: ELSIF t.Q THEN 61: ActRow[2]=2; 62: ELSE 63: UNK = TRUE; 64: END_IF 65: ... 66: END_CASE 67: END_IF 68: IF UNK = TRUE AND (ActRow[1] + ActRow[2] > 0) THEN /* any GTT is active */ 69: UNK = FALSE; 70: END_IF 71: END_IF 72: OldEN = EN; 73: END_FUNCTION_BLOCK </pre>
--	---

Figure 39: An excerpt of the automatically generated monitoring block for the cylinder behaviors in Figure 38

7.2. Empirical study of the usability evaluation

To observe detailed and more objective behavior aspects, empirical study was executed as a user evaluation together with the industry expert interviews (cf. Section 7.3). The user evaluation experiments were conducted to understand effectiveness and user satisfaction. *Effectiveness* means the possibility of accomplishing a successful task and that a user should be able to comprehend and create artifacts in the (modeling) language to an acceptable degree as summarized by Schalles et al. in [Scha13] in this regard. *User satisfaction* means the degree to which user needs are satisfied in the usage [ISO11] (it is defined similarly in different works of literature such as [Beva95, Scha13]). Over the 2019 summer semester and the 2019-2020 winter semester, three experiments were conducted with master and bachelor students of the mechanical engineering department at the Technical University of Munich. These students were regarded as the potential users of GTTs because they are future module developers and also application engineers [Voge14]. They participated in the training, exercises, and a user perception questionnaire survey over three experiments for the analysis of: (i) how much effectiveness is achieved compared to conventional language and (ii) how much satisfaction the users perceive during usage of the language. Additionally, it was also planned to assess feedback to elaborate the developed language coming up with users' perspectives. The experiments were conducted in a paper-based manner since the tool was still in development and the details of the experiment are presented in the following sections regarding hypotheses (7.2.1), experiment plans (7.2.2) and results (7.2.3 and 7.2.4). This evaluation was preliminarily published in [CVWU21].





7.2.1. Experimental hypotheses

Considering major usage activities of the specification language, as applied in [ZeVe07] and described in [Lude89], effectiveness could be defined as the ability to understand, create, and learn perspectives. Looking at each of these chronologically, enabling to create is important to represent the intention of the requirement (properties) in the viewpoints of the automation engineers, i.e., module developers for the library module, application developers for the compound modules (cf. Figure 7), of the automation system software, as grounds for verification (R-U2b). In the experiment with master students, module developers are assumed and the experiment with bachelor students, application engineers, are assumed and targeted since module developers are more skilled programmers and, thus, more familiar with such a behavior specification. Also, understandability is important for automation engineers to figure out whether the existing specifications depict the property correctly or how the specifications diverge from the intention (R-U2a). For the proper use of the language, the user must understand it correctly in regard to its syntaxes and semantics.

To measure usability in regard to these aspects, the evaluation was done by comparing the proposed approach with another specification language (comparative experiment). The experiment factor was differences of the languages requested to the participants to handle. As a control factor, another formal language (PN) was selected to see the superiority or inferiority of GTT, compared to the conventional language objectively. More precisely, Condition/Event (C/E) type PN [DaAl92] were selected for two reasons: First, students had been taught with this language previously in another mandatory lecture; and C/E PN are the simplest type to avoid unnecessary disadvantages in the control language. The language should be able to deliver the originally intended semantics to the user (who is reading the specification) through the understanding activity. This can be assessed by observing whether participants comprehend the intention of the specified descriptions in the language (H1.1).

In reverse, the language should be able to be used to describe intended behavior appropriately by the user (who is creating the specification). This can be assessed by observing whether participants use defined notations of the language properly to describe the given behavior (H1.2).

Table 8: C/E type PN notations and comparable elements in GTTs

Graphical symbol	Description	Interpretation	Comparables in GTTs
	Place (state)	Status where objects await processing or conditions that objects are in	Rows
	Transition (event)	State transition	Cascading rows
	Edge	Connection between places and transitions	-
	Initial transition (event)	Initial transition with omitting the common initial place	The first row
<i>Variable names and values</i>	Variables	Variable value changes	Columns (input/output) and cells

If the understanding and creating activities regarding the target language depend on the specific knowledge or experience of the user, the aim of user accessibility is challenged since the successful result of these activities would be affected by some individual knowledge or background levels, not by the accessibility of the language itself, and the domain experts with prior knowledge are not necessarily the language's end users [BAGB14] (H1.3). Additionally, scalability of the language is to be considered for practical and industrial cases. Although it would be systematically possible to describe enlarged behavior, comprehension and creation of the scaled version by users shall be assessed separately (H1.4). The degree of user satisfaction is another aspect to be researched in regard to the usability of the language, experimented through the questionnaire. Based

on the system usability scale items [Broo96] in consideration of the major activities of language usage, users must have confidence regarding comprehension and creation after some experience of using it (H2.1, H2.2). From the same point of view, the degree of experience for learning should not be regarded as very large (H2.3). In addition, the degree of satisfaction would be related to the users' will to utilize it as a solution or, more specifically, as a form to achieve the solution (H2.4, H2.5). The hypotheses regarding each research question are summarized in Table 9.

Table 9: Hypotheses and proving methods

Category	Hypothesis	Proof method
Effectiveness	H1.1: A user can understand a system behavior by the specifications in GTTs similar to or better than by the conventional language (PN).	Score comparison of the understanding specification task in GTTs and PN
	H1.2: A user can create a specification regarding a system behavior in GTTs similar to or better than in the conventional language (PN).	Score comparison of the creating specification task in GTTs and PN
	H1.3: Understanding and creating specifications in GTTs is less related to the background knowledge of the user regarding software engineering than to the conventional language (PN).	Correlation study between the understanding task score and the personal grade
	H1.4: The score of understanding task is less sensitive to the scale of the specification in the target language than the conventional language (PN).	Correlation study between the specification complexity and the score comparison of the understanding task
User satisfaction	H2.1: Users have confidence in understanding a specification in GTTs.	Questionnaire
	H2.2: Users have confidence in creating a specification in GTTs.	Questionnaire
	H2.3: Users think the language, i.e., GTTs, is reasonably learnable .	Questionnaire
	H2.4: Users are satisfied with the language, i.e., GTTs as a behavior specification representation method in general.	Questionnaire
	H2.5: Users would like to use the language, i.e., GTTs, as a behavior specification representation method in the future.	Questionnaire

7.2.2. Experiment planning

Regarding the three experiments, the following sections describe each of the experiment designs and participants. All of the program codes given to students during the experiments were in IEC 61131-3 ST and C/E type PN were selected as the control language in the comparison experiment, as commented earlier.

7.2.2.1 Comparing the effectiveness of GTT and PN concerning understanding and creating with separated participant groups (Exp-1)

The first experiment (Exp-1) was planned to conduct the effectiveness comparison of GTT and PN as a measurement of the appropriate level of tasks. The aspects of the effectiveness targeted here were mainly understandability and creatability, as well as the effect of the different scales to those aspects. Thus, participants were divided into two groups and assigned with the understanding and creating tasks in each language.

Exp-1 consisted of one hour tutorial of GTTs, given to the participants in the form of a lecture in the classroom allowing them to ask questions and including a review of examples. The corresponding lecture is Industrial Software Development for Engineers 2 (in German: Industrielle Softwareentwicklung für Ingenieure 2), a master course lecture in the mechanical engineering department. The lecture has well prepared and the contents are mature enough to be run more than five years. As they are master students, they were assumed to be future module developers with comparably mature programming skills. Before going to the tasks, the participants were asked to answer a questionnaire designed to collect data for their profile, including demographic data (Appendix A.1). Following paper-based exercise tasks had to be answered during another hour, one week after the tutorial, due to the timing constraint. Three tasks were prepared regarding: i) to understand the behavior in the specification (GTT or PN) and correct the given code in a comparably smaller scale problem, ii) same task with i) but in a comparably greater scale problem, and iii) to create the formal specification after reading the given natural language description (cf. Appendix A.2). The participants were divided into two groups and provided with the behavior described in GTT or PN. Basic notations are given to the participants for both languages (cf. Appendix A.3). Each group received the specification of each language alternately in two understanding tasks so that they would not be biased by the learning effect of the tasks.

In task1 (simpler understanding task, 15 min), a brief description in the natural language and a formal specification (in GTT for Group1 and in PN for Group2) were provided as information of the target system block. Participants were requested to understand the given information and to then find errors in the program code. Task2 (scaled understanding task, 15 min) was framed the same way, but with another system block and the formal specification was assigned the opposite way: in PN for Group1 and in GTT for Group2 to avoid the learning effect over task1 and task2. The assigned time was set the same as with task1 despite the increased scale, since the number of erratic parts was reduced. For task3 (a creating task, 30 min), a precise description in the natural language and the corresponding program code were provided as information about the target system block. Participants were requested to understand the given information and to then create a formal specification in both GTT and PN. More time was assigned, assuming that creating requires

more time than understanding since it requires more processes such as notation diversity, syntax check, and structuring [ShLo88].

7.2.2.2 Comparing the effectiveness of GTT and PN concerning understanding and creating by the same participants (Exp-2)

This experiment (Exp-2) was designed based on the analysis of Exp-1 and conducted as part of the class exam with 34 master course students of mechanical engineering department. The students are fond of software engineering, but basically with mechanical background, interested in applications. Learning from the Exp-1 and its result, which will be discussed in detail in the result discussion section, as well as the complexity level between GTTs and the control language (PN), were to be balanced more precisely and correctly. Since the participants had to handle both GTTs and PN as specification languages for each task, the target behavior of each should not be the same due to the learning effect. In other words, one might recognize behavior with the easier language for him/herself and try to apply it to the other if the same system is given in two languages. Instead, the system for each language was different, but the complexity of the behavior described in the language was controlled to be at the same level. For this, each had the same table size when described in GTTs and the same number of places and transitions in PN.

Two tasks were provided in Exp-2 with subtasks for each. In the first task, a brief description in the natural language and a formal specification in PN were provided. The first subtasks are regarding the understanding of the specification consisting of two questions of understanding overall behavior and tracking some signal pair changes. Different from Exp-1, program code was not involved in this task on the one hand to simplify the task focusing on the specification, and on the other hand to remove any bias due to program comprehension abilities. Since the amount of information and the number of questions were reduced, the assigned time was also decreased to 6 minutes. The second subtasks were in regard to creating the specification, which was partially based on a given specification. The creating of repeated behavior was the special focus in this creation experiment. The partial behavior was thus indicated as a repeated part on the given specification and participants were asked to generate the change on the sheet. In PN, two options were given: to implement the change with a counting variable (easy), and to implement the change by adding places and corresponding edges (hard) to see which level the GTT is comparable to. The next task was framed the same way as with the first one with PN, but targeting a different system with the formal specification in GTT. Program code was also left out to remove the bias factor, and the amount of writing was slashed to focus on the specific notation (for repetition description) as well as to harmonize with the other exam questions in the viewpoint of the difficulty level.

Materials used for Exp-2 is described in Appendix B.

7.2.2.3 Evaluating subjective user satisfaction (Exp-3)

This experiment (Exp-3) was designed and planned to be conducted at the bachelor student level (5th semester) in perspective of behavior complexity and the range of notation. While the master students could be regarded as module developers who are able to and willing to deal with this type of formal specification, the bachelor students are comparable to the application engineers. Based on this assumption, the subjects were decided to be master students in the previous experiments and then extended down to the bachelor level in this experiment to conform to the appropriateness in an immature background level.

These are downsized and reduced to consider the level of the participants as well as to focus on the effectivity of some specific notations of GTT. The ultimate goal of this experiment compared to the previous experiments is to achieve a qualitative assessment from the participants regarding their subjective perceptions. The experiment was conducted through the lecture in a classroom, consisting of 40 minutes of tutorial and exercise tasks asked of participants to answer for 35 minutes, including a qualitative questionnaire.

The exercises consisted of an understanding task and a creating task like the other experiments: answering the question regarding behavior with respect to the brief natural language description and a specification in GTT and creating the specification based on the partially given specification in GTT. After that, the participants were asked to answer the evaluation questionnaire, which is a tailored version of the System Usability Scale (SUS) [Broo96] regarding understanding, creating, and general impression. The questionnaire was intended to rate these aspects as perceptions in a Likert scale of one to five. Answer options consisted of five levels from the least to the most, namely, *strongly disagree*, *disagree*, *uncertain*, *agree*, and *strongly agree*. To avoid unintentional factors, the statements were prepared in a mixed tone (of positives and negatives) by forcing the respondent to read each statement and make an effort to think about it (cf. [Davi89, WoRB03]).

Materials used for Exp-3 is described in Appendix C and the overall experiment plan is summarized in Table 10.

7.2.3. Participants profile

In the experiments, the participants participated in training and were asked to perform the exercise tasks and answer the subjective assessment questionnaire depending on the experiment. Additional questionnaires were given to the participants to collect their profiling information.

Table 10: Summary of the tasks performed in the experiments

Experiment	Task description	Time (min)	Difficulty level	Remarks	
Comparing effectiveness	<ul style="list-style-type: none"> ▪ Target group: Master students 				
	Exp-1 (N = 9) *for Group1 and Group2 each	1. Understanding Spec. (1) in GTT/PN To read the specification and find violation in the given shorter IEC 61131-3 code: checking branch condition and missing implementation	15	Base 1	<ul style="list-style-type: none"> ▪ Given specification: Group1 in GTT, Group2 in PN
		2. Understanding the specification (2) in GTT/PN– scaled To read the scaled specification and find violation in the given longer IEC 61131-3 code: checking branch condition	15	> Base 1	<ul style="list-style-type: none"> ▪ Given specification: Group1 in PN, Group2 in GTT (switched specification language) ▪ Scaled size specification and program code targeting more difficult level
		3. Creating the specification To generate the specification regarding the natural language description and corresponding code in IEC 61131-3	30	Base 2	<ul style="list-style-type: none"> ▪ Group1 in GTT → PN, Group2 in PN → GTT - Conversed creating order of the language ▪ More time assigned on creating
	Exp-2 (N = 34)	<ul style="list-style-type: none"> ▪ Target group: Master students 			
		1. Understanding the specification in PN To read the specification with respect to the overall behavior understanding and specific variable pair traces	6	< Base 1 (Base 3)	<ul style="list-style-type: none"> ▪ Tasks are simplified by removing the code inspection part and balanced complexity-wise regarding the target system in GTT and PN questions ▪ The order of solving tasks and time limit is not controlled - Marked on the left column is the expected time duration for each task - Participants would assign as they wish within the overall allowed time duration (90 min)
		2. Creating the specification in PN To implement partial repetition on the given PN (i) using a variable and (ii) adding more places	6	< Base 2 (Base 4)	
		3. Understanding the specification in GTTs To read the specification with respect to the overall behavior understanding and specific variable pair traces	6	= Base 3	
		4. Creating the specification in GTTs To implement partial repetition on the given GTT table	2	= Base 4	
Subjective satisfaction	<ul style="list-style-type: none"> ▪ Target group: Bachelor students 				
	Exp-3 (N = 73)	1. Understanding the specification in GTTs To read the specification with respect to the overall behavior understanding and specific variable pair traces	15	= Base 3	<ul style="list-style-type: none"> ▪ Similar tasks to Exp-2 with more time considering the level of participants ▪ Reduced scope in creating task compared to Exp-1; but most basic notations are included
		2. Creating the specification in GTTs To implement partial repetition	10	< Base 1 & > Base 4	
	3. Qualitative questionnaire		10	N/A	-

One of the reasons for the conduct of these evaluations with university students is that they would be at a similar level to junior engineers [Usbu20] in the field as major future users of the language, i.e., GTTs and that they are likely to use such types of languages in (future) practices. Therefore, bachelor students represent immature engineers with some basic theoretical knowledge about engineering and embedded system implementation, and master students are more mature in automation software engineering inclusive of specifications.

In Exp-1, although the number of individuals attending was 29, 18 of them were valid participants (9 for each group) who had the knowledge about the PN and had attended the training session one week before the evaluation (i.e., there was a timing difference between the training and evaluation due to the timing constraint of the lecture as explained in the experiment planning). The overall profile is also effective for Exp-2 in which participants from the same group (i.e., students from the same lecture within the same semester) attended. In Exp-2, there were 34 participants. Although it was not revealed clearly who had attended the tutorial due to the anonymity of the experiments, the participants could be regarded as fully trained since it was an exam that matters to their degree results. The participants were master course students with mostly a mechanical engineering major with an average grade of 2.26 (σ : 0.50) in the German grade mark scale with 1 as the highest and 5 as the lowest grade (Figure 40 – a). More than 80% of the participants had some experience in the industry in the form of internships and working. The work duration among the experienced ones spans from 2 to 8 months for an internship and from 6 to 48 months for part-time or full-time work (Figure 40 – b,c). The participants evaluated their programming skills as intermediate level (mean: 3.17 and σ : 0.17 in a Likert scale with 1 as the lowest and 5 as the highest) in a self-assessment.

The participants of Exp-3 were 73 bachelor students. Assuming that the students did not have influential industry experiences since the lecture was taken in the fifth semester of the bachelor course, only the grade information of the participants was gathered (Figure 40 – d).

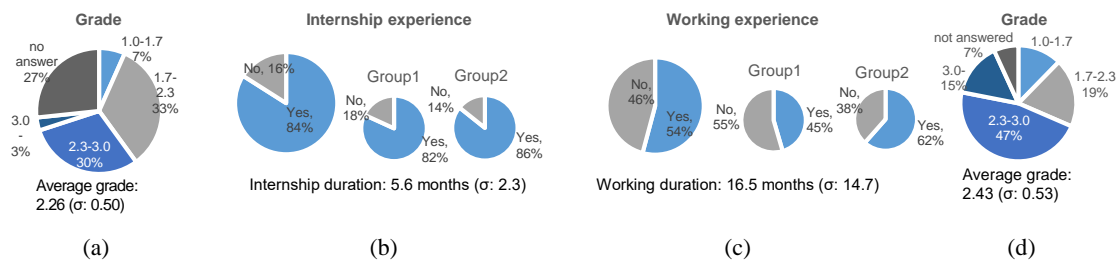


Figure 40: Profiles of the participants: (a)-(c) for Exp-1 and (d) for Exp-3 [CVWU21]

7.2.4. Experiment results

The following sections describe each of the experiment results collected in the experiments are presented with descriptive and inference statistics.

7.2.4.1 Understanding and creating with separated participant groups (Exp-1)

First, in the understanding tasks (i.e., task1 and task2) of Exp-1, the participants are asked to find faults within the program code with respect to the given specification in GTT or PN (Figure 41 – a,b). In the first task, GTT specification (Group1) showed better scores than PN one (Group2). Conversely, PN specification (Group1) showed a better result than GTT one (Group2). Although the overall point was higher in task2 for both languages, direct comparison between task1 and task2 would not be meaningful because of differences in the type of system behavior, the size of specifications, and the length of the given code. Although the second task was targeted to evaluate the scaled case (i.e., more difficult), the PN specification scale was not managed well while GTT specifications and the given program size were scaled. Therefore, it is hard to say which one language is dominant to the other regarding understandability (H1.1: understanding GTT similar to or better than PN).

For the creating task, the participants were asked to create specifications in both languages for one target system. The overall mean value of the results was very similar in GTT and PN (Figure 41 – c), and thus H1.2 (creating GTT similar to or better than PN) is true. However, the results of each group are very interesting. For Group1, the specification creation task of GTT was given before PN, which is in reverse order in Group2. For each group, the preceding one (i.e., GTT for Group1 and PN for Group2) got a lower point. They probably learned behavior during the first specification creation task and could then represent the second specification creation better, which is promising for practical use (applied to both cases).

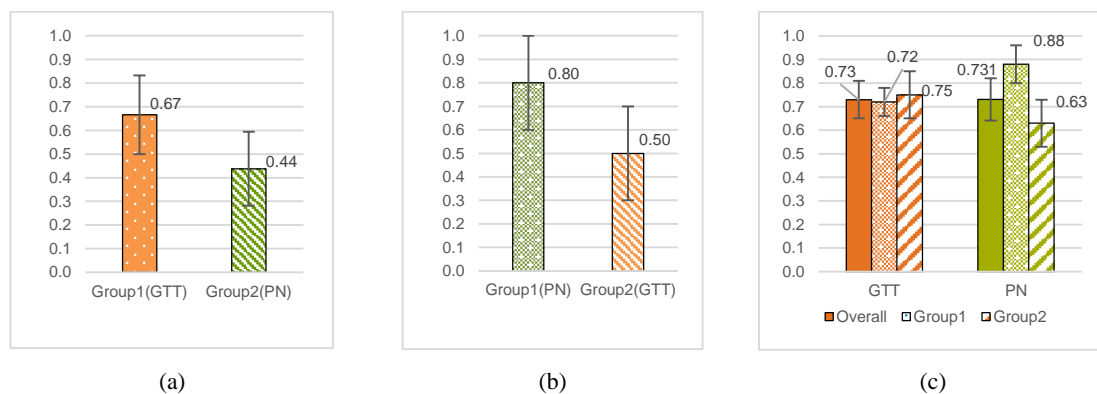


Figure 41: Exercise score comparison of Exp-1 (normalized mean value with error bars of the standard error): (a) – task1 (understanding), (b) – task2 (understanding, scaled), and (c) task3 (creating) [CVWU21]

One additional observation of tasks is the sensitivity to the specification scale (cf. Table 11). Since the specification size grew around 2.3 times in GTT (from 3 to 7 rows), a degradation was expected and observed as 25% degraded in the scaled size (7 rows) compared to the smaller one (3 rows). In PN, since the specification size grew 2.3 times (from 8 edges to 18 edges), a 37% degradation was observed. Although it is a simple comparison, not considering the difference in the number of inputs/outputs that affect the GTT size, the sensitivity of the language to the growth of the specification seems less in GTT than in PN (H1.4: less sensitivity of the understanding ability to the problem scale).

Table 11: Task and specification complexity comparison in Exp-1*

a. GTT					
Group	Task	I/O	Specification size	Program size	Score
Group1	Task1	2 IN / 1 OUT	3 rows	18 lines	0.67
Group2	Task2	6 IN / 4 OUT	7 rows	53 lines	0.50

b. PN					
Group	Task	I/O	Specification size	Program size	Score
Group1	Task2	6 IN / 4 OUT	5 places, 8 edges	53 lines	0.80
Group2	Task1	2 IN / 1 OUT	5 places, 18 edges	18 lines	0.44

*The number of rows for GTT, and the number of places and edges for PN as specification size

The result was also analyzed with respect to the background knowledge of the participants. Based on the participant’s profile, grade, working and internship experience could be regarded as criteria for the background level. The trend line of GTT showed a more gentle slope than the one of PN (Figure 42 – a) even though it takes the position lower than PN. Here, the creation task result was considered for comparison since the effective comparison is possible between GTT and PN with the valid number of samples who answered the task and shared the grade (6 for GTT and 8 for PN). The comparison of the mean value of creation results also showed that the internship experience is less influential in GTT than PN (Figure 42 – b). The comparison for working experiences showed no difference between GTT and PN (Figure 42 – c). Therefore, H1.3 (understanding/creating ability related to background less in GTT than PN) is concluded as true.

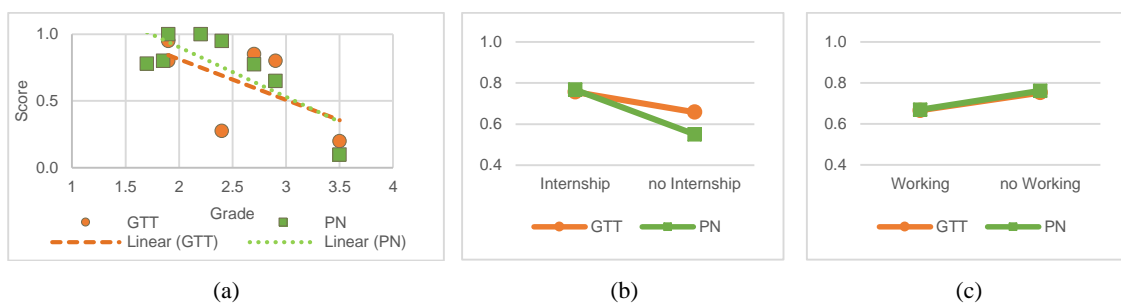


Figure 42: Correlation (based on Exp-1) of (a) score vs. grade, (b) score vs. internship experience, and (c) score vs. working experience [CVWU21]

Through this first experiment, some bias points were found. Some of the participants turned not to get used to the IEC 61131-3 language, different from the expectation. Although the implementation was comparably simple, this biased the result. Also, the time difference between training and evaluation affected the controlling of attendance of the participants. Splitting the training session and evaluation, and non-compulsive evaluation resulted in attendance with a smaller number of valid participants. In addition, the fact that each group was given different tasks made the comparison harder, especially due to the small number of subjects. Task1 and task2 were planned to see the effect of the scaled problem; however, the result could not be compared directly due to the low number of participants, which led to dependent results in the participants' profiles.

7.2.4.2 Understanding and creating in different language by the same participant (Exp-2)

In this experiment, each participant had to solve two tasks (GTT and PN specification). The tasks are balanced validated by transforming each specification from one language to the other. Scores are presented as normalized to ease the comparison. Although measuring understanding and creating effectiveness had been targeted in Exp-1, the number of valid participants was not sufficient to justify the result of the experiment. So it was measured in Exp-2 again with the higher number of participants (N=73) to strengthen the experiment for H1.1, H1.2 (Understanding and creating ability comparison) and H1.3 (understanding/creating ability related to background).

There were two points targeted in the understanding task: one was understanding overall behavior (allotted 8 points) and the other was tracking of variable value set changes (allotted 3 points). For both, GTT showed a statistically meaningful improvement (5.1% and 7.7% respectively) and 5% for overall score in comparison, supporting H1.1 (understanding GTT similar to or better than PN) to be true (Figure 43 – a).

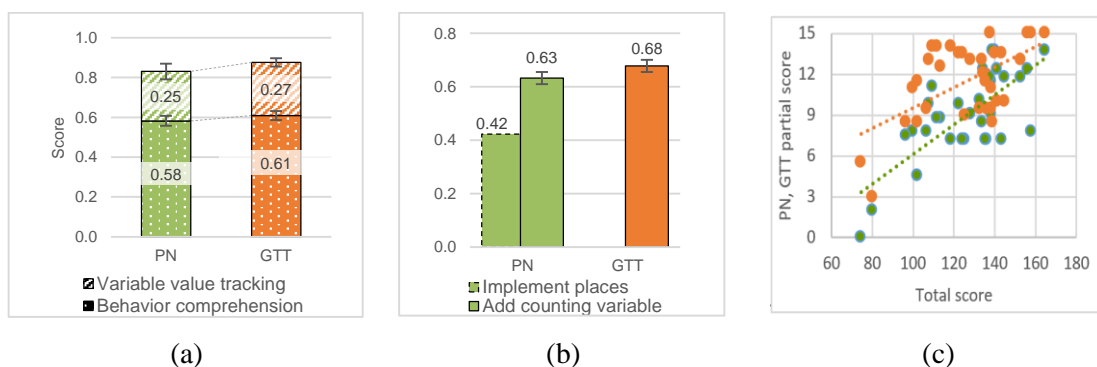


Figure 43: Score comparison of GTT and PN from Exp-2(a) Understanding task, (b) Creating task, and (c) Correlation of score to the total score [CVWU21]

Creating specification task was focused on representing the repetition of certain behavior. The block repetition notation of GTT was to be compared with that of PN. In the case of PN, two options were given: using a counter variable (easy) and adding places and corresponding edges (hard). It is hypothesized that participants would get higher scores by using a counting variable than by adding more PN notation elements (i.e., places and transitions) and thus was proved to be true as seen in the scores received by the students (Figure 43 – b). Comparing GTT to PN, students obviously got more points with the repetition block notation in GTT than by changing the net structure and slightly more than using the variable. Therefore, H1.2 (creating GTT similar to or better than PN) is true.

The score of PN and GTT was related to the overall score, assuming that the total score indicates their background knowledge regarding software engineering. If one specification language required higher knowledge, the slope would be proportional to the overall score. As seen in Figure 43 – c, GTT shows a more gradual slope than PN, which means that GTT has less correlation with the knowledge or background regarding software engineering. This supports H1.3 (understanding/creating ability related to background less in GTT than PN) to be true.

7.2.4.3 Subjective user satisfaction (Exp-3)

The focus of the third evaluation concerned the subjective perception of GTT. After the tutorial, participants were asked to solve exercise tasks and then to answer the questionnaire. The exercises were similar to the understanding task of Exp-2 but were simplified, the choice question considering the different student levels. The score of the understanding task was 0.84 (normalized mean with standard deviation $\sigma = 1.53$ and skewness $\gamma_1 = -1.4$), which is slightly less than Exp-2 but could be an indication that the participants were following the concept successfully. The creating task was also adapted to the level by providing the table and filling the blanks. The mean score of the creating task was 0.78 ($\sigma = 3.57$, $\gamma_1 = -1.2$).

The scores were to be checked if they were related to the overall grade of the participants assuming that the higher grade implies a higher knowledge level of system engineering. For this experiment, the overall grades of the students up to that point were collected through the questionnaire. Consistent to the results of the previous experiments, the result of this experiment also supports the uncorrelated relationship between the score of the exercises and the overall grade (Figure 44). This partially supports H1.3 (understanding/creating ability related to background less in GTT than PN) since the result could not be compared to the PN cases, which were not evaluated during the Exp-3. At least, the result showed that irrelevancy trend of the understanding and creating scores over the grades.

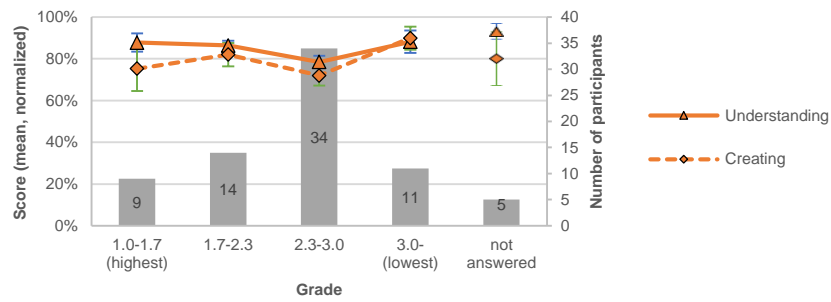


Figure 44: Correlation of the GTT exercise score and the overall grade (Bachelor course) from Exp-3 [CVWU21]

For qualitative questions, each choice is rated as 1 – most negative (i.e., strongly disagree), 2 – negative (i.e., disagree), 3 – neutral, 4 – positive (agree), 5 – most positive (i.e., strongly agree). The majority of the participants (81%) accept the language as understandable concept for them (Figure 45 – a). Although the rate of comprehending the given task is lower than the understanding rate of the concept, more than half (58%) agreed that comprehending the specification in GTT is easy. For the remembering syntaxes, although the rating is spread somewhat more than with the others, the mean value is still indicating less than three (since it is the negative question), and the positive answer rate (44%) is higher than the negative (25%). Therefore, the conclusion is still positive. Overall, as seen in the graph of the blue area and in the skewness values, understandability can be deemed as positive. Thus, H2.1 (confidence in understanding) is true.

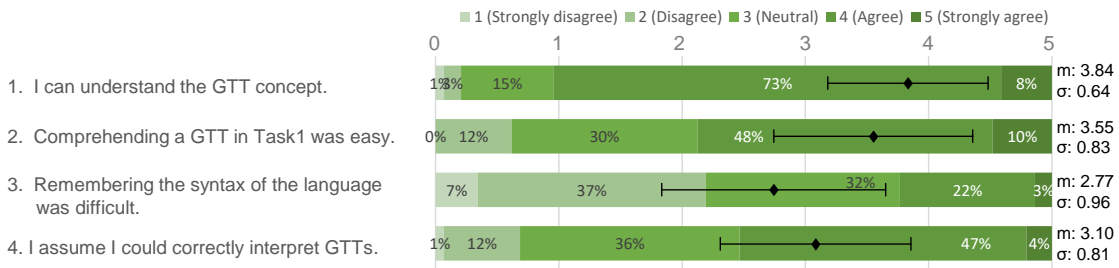
For creating (Figure 45 – b), more than half of the participants (56%) answered that they could apply GTT concepts to creating specifications. The result is lower than the understanding result, but it can still be considered as positive feedback. The participants answered mostly neutral to the question that the given exercise tasks were not easy to create specifications (mean = 3.03), although they agreed that they could mostly understand the given task (39% positive, 31% neutral). The collected answers with difficulty points were mostly in writing Boolean expressions (15 out of 27 collected answers). Nevertheless, the participants agreed that they could create a specification in GTTs overall. Therefore, H2.2 (confidence in creating) is true.

Almost 65% of participants answered positive to usability of the GTT being easy (Figure 45 – c, mean = 3.63, $\sigma = 0.63$, $\gamma_1 = -0.73$). This is consistent with the result of the question regarding the perceived complexity in number 4, which was answered positive by 81% (strongly disagree and disagree) of participants. So, in general, the satisfaction rate seems to be high, which leads to H2.4 (satisfactory method as a behavior specification representation) to be true. For learning, 77% of participants answered positive to the question that an application engineer can learn GTT in a reasonable time (a day or less) – strongest positive (mean = 3.89, $\sigma = 0.67$, $\gamma_1 = -0.2$), supporting

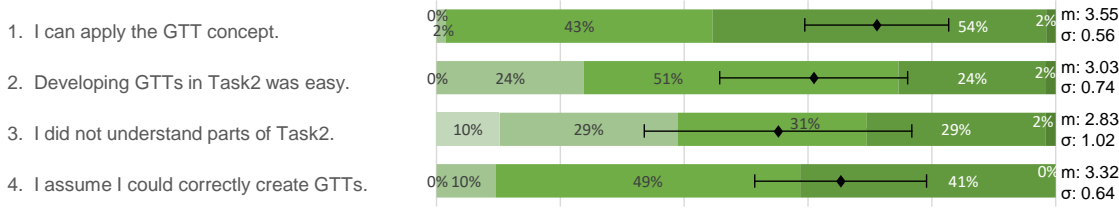
H2.3 (learnable) to be true. However, the answer to the question regarding future usage is comparably lower than to the other questions although the result is still positive with the mean of 3.33 ($\sigma = 0.62$) with γ_1 as 0.05. This supports H2.5 (willing to use in the future) to be true.

7.2.5. User evaluation summary

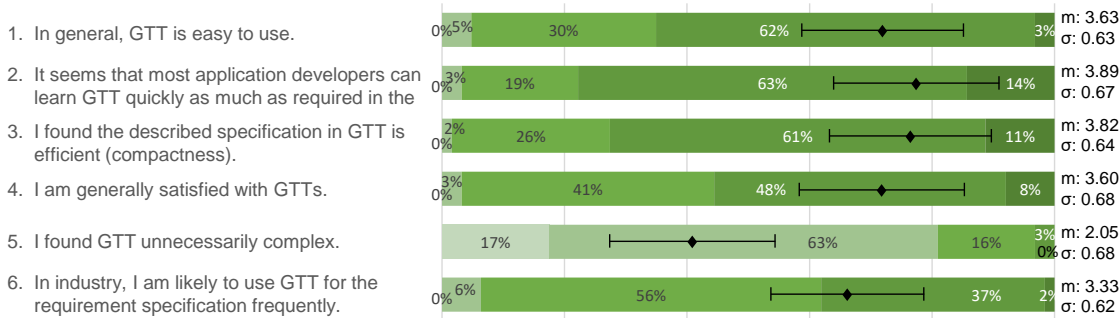
The usability of GTT was investigated through user evaluation and its analysis with. It was hypothesized that users can understand the system behavior through GTT specification similar to or better than another conventional language (H1.1). In Exp-1 and Exp-2, the control language was chosen as PN. Although it was not clearly revealed in Exp-1, GTT showed better results than PN regarding understanding behaviors and tracking variable value changes in Exp-2. Therefore, the



(a)



(b)



(c)

Figure 45: Qualitative evaluation result in Exp-3. (a) Understanding, (b) Creating, and (c) overall. Reproduced from [CVWU21]

result could be considered as valid. For creating the specification (H1.2), Exp-1 showed similar results in both languages and Exp-2 showed the superiority of GTT with respect to representing the repetition of some specific block. However, Exp-1 did not have a large enough number of participants that were not biased by differences of each, and Exp-2 focused on a specific notation only for the creation task. Therefore, the validity level could be said to be weak, and the hypothesis should be investigated with a wider range of notation usages to support the hypothesis more strongly. Also, the correlation between usability and background of the user was examined (H1.3). Although the examined result shows as hypothesized, it is under the assumption of the implying relationship between the background knowledge levels and the academic grades or internship experiences. Thus, the validity of this relationship should be clarified to fully validate the correlation analysis. Regarding the sensitivity of the specification scale (H1.4), it was concluded as true comparing the results from the tasks; however, the direct comparison between the groups in Exp-1 was not entirely sound. This is because on the one hand, the results are from different groups, and on the other hand, the groups lack representativeness due to the limited number of participants. Also, the scale of the specifications and of the established problem, including the described system, was not well enough managed to be able to verify the hypothesis. Thus, the result cannot be regarded as valid. On remark for Exp-1 and Exp-2 is that the evaluation was done in the paper-based manner. It was an advantage as the participants could freely express as they want without being affected by the maturity of the tool. It is a threat as field engineers will only have to accept a notation given in an appropriate tool.

Speculating about the subjective perception levels of the language users, a positive user perception level for understanding and creating activities was hypothesized (H2.1, H2.2). As the result shows, a high rate of positive feedback on both (higher in understanding) was observed. Users also regarded the language to be reasonably learnable (H2.3). However, for this point, additional evaluation regarding learning would be necessary to measure the precise level of learnability. While the overall user perception regarding its complexity and satisfaction was positive (H2.4), it was also found that almost half of the participants were still wavering over whether to use it in practice in spite of positive feedback on the average (also observable as $\gamma_1 = -0.20$ – the lowest among all the questions). Therefore, performing successive work to figure out the main reason for this phenomenon and the corresponding adaptation of the language would be necessary. In addition, to make the results of the qualitative assessment stronger, a further survey comparing other formal specification languages with a promising number of participants should be conducted.

Table 12: Summary of the hypotheses evaluation

Related hypothesis	Experiment	Result (validity)
H1.1: A user can understand a system behavior through the specification in GTTs similar to or better than through the conventional language (PN).	Exp-1, Exp-2	True (valid)
H1.2: A user can create a specification regarding a system behavior in GTTs similar to or better than in the conventional language (PN).	Exp-1, Exp-2	True (weakly valid)
H1.3: Understanding and creating specifications in GTTs is less related to the background knowledge of the user regarding software engineering than conventional language (PN).	Exp-1, Exp-2, Exp-3	True (weak valid)
H1.4: The score of understanding task is less sensitive to the scale of the specification in the target language than the conventional language (PN).	Exp-1, Exp-2	True (not valid)
H2.1: Users have confidence in understanding a specification in GTTs.	Exp-3	True (valid)
H2.2: Users have confidence in creating a specification in GTTs.	Exp-3	True (valid)
H2.3: Users think the language, i.e., GTTs, is reasonably learnable.	Exp-3	True (valid)
H2.4: Users are satisfied with the language, i.e., GTTs, as a behavior specification representation method in general.	Exp-3	True (valid)
H2.5: Users would like to use the language, i.e., GTTs, as a behavior specification representation method in the future.	Exp-3	True (weakly valid)

7.3. Expert evaluation from industry

Earlier, the applicability evaluation has shown the feasibility of the presented approach to be used for realistic use cases. A usability evaluation has been also conducted with the potential users to measure how much the presented approach is acceptable for them. To strengthen the usability measurement, direct feedback from the current industrial field is necessary. Therefore, the expert evaluation has been conducted with the field engineers and managers. Since it is not suitable to gather that many personnel in companies as done in the empirical study, scattered interviews within workshops and meetings were conducted instead. All experts have a deep understanding of IEC 61131-3 code development project. No questionnaire sheets were used but rather the topic was discussed in the informal conversation after giving them a presentation (15min) of the approach. And then, the current quality management processes of the companies were discussed followed by the impression and feedback of the approach. This section summarizes the feedback obtained from these interviews.

The first workshop was with experts from five different companies of machine & plant manufacturing companies and IEC 61131-3 compliant engineering environment provider. Some positive feedback was delivered also including some concerns about scalability to be applied in practices. Each comment was stated from different participants.

“For some cases, we apply test-driven development and debugging. The way of describing the sequence seems interesting and could be applied there.” (cf. R-E2)

“The table form is consistent and intuitive. The approach could be improved with some adaptations like handling the size – columns for the relevant input and the rows for the sequences. Additionally, the function dependency representation will help to be applicable in a scaled system.” (cf. R-U2 – R-U6)

“It would be beneficial in a sense that the library functions could be verified easily since they are used very often and more times. Formal verification is still hard to apply to prepare the specification of each.” (cf. R-E1)

“We have tried generating specification in Petri nets but did not work out well. If the specification in tables could be obtained inexpensively, we would like to try.” (cf. R-E5)

There was another interview with the department leader of the software standards in a machine & plant manufacturing company. This company is active in the field of medical manufacturing and packaging automation. Since the medical-related systems are under the effect of legal regulations, the implementation history and quality assurance activities have to be carefully managed.

“Although it might require some adjustment, the concept is good. It seems to be worthy to check the applicability to the real industry cases.”

In a joint seminar with a manufacturer of industrial automation devices, the concept of the approach as well as a short demonstration of the use case was presented. After that, some feedback from the automation engineer’s point of view was collected. The opinion was more regarding the documentation of the software behavior than the formal verification.

“Even if we cannot apply the formal verification directly at once, at least the approach could be applied to the functionality description especially for the changed part description. It would be helpful to document so that the others could read it to understand the behavior for further reuse.” (cf. R-E6)

An additional opinion was collected from a chief software development officer of a machine and plant manufacturing company, covering automotive system components and medical technology products. Highlighting the importance of obeying the regulations, as also published in [VoCh20], the industry expert reported his interest:

“To follow up the regulation like GAMP, it is necessary to document a specific description about the specification, test procedures, and change history to track the all the activities of the change implementation. These points could be helped by the presented approach as it supports the documentation as well as the formal verification of the implemented control software.” (cf. R-E1, R-E5, R-E6)

Conclusively, the gathered feedback was mainly positive (cf. Table 13). Although some concerns were also observed for the feasibility to the industry settings, the basic ideas of the approach were agreed. One of the companies stated above agreed on joint work of adjusting the approach and applying it in the form of the transfer project within the control software development processes for medical related business sector; this reflects that the approach satisfies the necessity of the industry and fits for the needs.

Table 13: Summary of the expert evaluation comments relating to the requirements

Opinion (summerized)	Relevant requirement(s)
Applicable to test-driven development	R-E2
Intuitive format, scalability to be reviewed	R-U2 – R-U6
Generating specification with less effort would be beneficial	R-E5
Applicable for the documentation purpose	R-E6
Pragmatic way to adhere to the legal regulations	R-E1, R-E5, R-E6

8. Assessment of the Fulfillment of the Requirements

The requirements discussed in chapter 3 were envisioned in three points of view: target system, its engineering processes, and usability, and these are evaluated throughout the case studies, empirical studies, and expert interviews. In this chapter, the assessment of the evaluation with respect to the requirement are presented (summarized in Table 14).

As the system characteristics could be prescribed in terms of input and output sequences, the specification language was required to present the relationship of these two sequences (R-T1). The basic structure of GTT allows to present the input (sequences) and corresponding output (sequences) in a structured form; the fulfillment was, thus, clearly shown through the application cases. Since the target system is reactive systems, which has its own state and changing the state into another state depending on the input sequences, each state (accepting input and resulting output) and its transition condition should be described in the specification (R-T2). Specific values of the system component control signals and discrete event processes correspondingly were targeted (R-T4). Rows, which are regarded as a state (each) in a system, in GTT could satisfy these characteristics as also shown the application cases. However, only a specific path of the control can be described within a GTT due to the nature of the table. Although the behavior repetition was achievable through the repetition symbol, diverging/converging the execution control (branching) is not capable in GTT at the moment. A state in a system is not defined by or limited to a specific input/output value only but allows a range of the values with the constraint expressions (R-T3). Allowing generalizing expressions in cells could fit this requirement, as shown in all application cases (cf. Section 7.1) and especially the one introduced in Section 7.1.4. Considering the cyclic execution of PLC (R-T5), the value set in a row should retain during the defined duration in a GTT. Consistent representation of common library functions in IEC 61131-3 was targeted (R-T6); this was shown as satisfied within the case example in Section 7.1.4.

The approach is developed based on the testing practices. Although testing could provide the chances to control the quality, it is not always feasible and cheap to execute the system for testing, which are the characteristics to be covered by formal verification (R-E1). Since the concept is generalizing same sort of test cases, test cases could be derived in reverse by designating specific values in the existing GTT (R-E2). Initiated from one of the motivations of applying formal verification on aPS engineering processes, GTTs as a formal specification in the formal verification procedure is shown in the previously published paper, [CWUB18a], and [Weig21]. The result of

the verification should also be easily used for the debugging in the viewpoint of automation engineers (R-E3); the way of connecting counterexample and the corresponding GTT was presented in Section 5.4.1. The capability of systematic data handling is a requirement to be implemented in a tool and also to be integrated with a model checking, not managed solely in a manual manner (R-E5). This was proven by the implemented tool integrated into an IEC 61131-3 IDE, and this also supports the debugging support requirement (R-E3). Transformation cases from and into GTT (shown in Section 5.4.2 and 5.4.3) also showed that they support the systematic data handling requirement (R-E5). One of the major requirements and the motivation of the presented approach is the characteristic of the control software change in aPS lifecycle: it changes very often in a small scale. That is, the software goes through a validation process very often accordingly. Since testing could not always be the best solution due to its dynamic execution manner requiring a physical system and quite some time, regression verification was focused on completing it, and the changed part description was required accordingly (R-E5). To fulfill this requirement, an approach to achieve preliminary specification from the existing code was presented (Section 5.4.2) with a use case to show its feasibility (Section 7.1.3). Still, specification inferencing from the existing code is limited to a specific language (SFC) and needs to be further researched into the other languages. Strengthening quality management through the presented concept supporting change management and monitoring (R-E4, R-E6) was approached as its extended utilization to strengthen quality management. R-E6 also targets the specification as a tool for communication and data exchange between automation engineers, and this was demonstrated over the presented use cases (Section 7.1).

Promoting formal verification requires usable formal specification language (R-U2); this is required to be supported by a corresponding tool for efficient engineering processes (R-U1). Prototypical tool implementation (cf. chapter 6) proves the feasibility of the tooling. Usability aspects were approached in the empirical study (Section 7.2) through the experiment with the mechanical department students who were regarded as potential automation engineers. Understanding, creating ability, and the comparison regarding the scalability was assessed by being compared with PN (R-U2, R-U3, R-U5). Some weak point of the experiment in Exp-1 were observed, including a smaller number of participants. Although the understanding and creating performance was shown as satisfactory, partial notations were targeted in creating part. The subjective aspects were assessed through the questionnaire (R-U4, R-U6). Mostly positive results were observed; however, there still are some open points to decide the validity of the creating aspect. In the workshops and interview, opinions from the industry experts were gathered and this also turned out to be positive for its usage although some concerns about the scaling were observed as not entirely resolved. As

mentioned, a joint project is being prepared with one of the interviewee to apply presented approach to their engineering process.

Table 14: Summary of the evaluation of the presented approach with respect to the requirements

Requirement	Tool supporting (Section 6)	Feasibility analysis (Section 7.1)	Empirical study (Section 7.2)	Expert evaluation (Section 7.3)	Overall evaluation
R-T1. Presenting the relationship between input and output		+ (UC1, UC2)			+
R-T2. Presenting the state changes		+ (UC1, UC2)			+
R-T3. Abstracted value range for wider coverage		+ (UC1, UC2)			+
R-T4. Supporting discrete event processes		+ (UC1, UC2)			+
R-T5. Supporting cycling execution representation		+ (UC1, UC2)			+
R-T6. Compatible to the software in IEC 61131-3		+ (UC1, UC2)			+
R-E1. Usable over engineering processes		+ (UC1 - UC4)		+	+
R-E2. Test case instantiation from the specification		+ (UC1, UC2)			+
R-E3. Supporting debugging of the implementation	+				+
R-E4. Supporting monitoring block generation		+ (UC4)			+
R-E5. Supporting change validations		+ (UC3)			+
R-E6. Supporting documentation and information exchange		+ (UC1 - UC4)		+	+
R-U1. Tool supporting	+				+
R-U2. Understanding specifications			+ (Exp-1 - Exp-3)	+	+
R-U3. Creating specifications			0 (Exp-1 - Exp-3)		0
R-U4. Learning specifications			+ (Exp-1 - Exp-3)	+	+
R-U5. Scalability of specifications			0 (Exp-1)	0	0
R-U6. Satisfaction with the specification approach			+ (Exp-3)	+	+

*+: Fully satisfied, 0: Partially satisfied, -: not satisfied, empty cell: not available

9. Conclusion and Outlook

Software specification itself takes a role of a language over the lifecycle of the software in the sense that various stakeholders express their requirements and understand others' intentions. Moreover, the importance of the correct software specification gets highlighted since engineering activities including design, implementation, validation, and deployment, are grounded on it and, thus, correct and efficient generation of artifacts throughout those activities could be affected by the specification. That is, well-structured specification could support not only describing the requirement for further understanding of the corresponding engineers but also being used systematically during engineering processes to improve the quality of the implemented control software.

In this thesis, the requirements of a control software specification language for aPS to be used to improve control software quality efficiently were presented. Completing testing, which is useful for typical or expected faults and also requires the execution of the artifact consuming quite some time, the advantage of formal verification was claimed, especially for aPS change cases that occurs frequently, are typically small, but require effort to validate. As a mandatory element of the formal verification, the requirements for the specification were analyzed to maximize the utilization as a well-formed document for a description of the implementation characteristics and for utilization in another artifact form, focusing on the fact that the specification could be used by and exchanged between different types of engineers (i.e., module developers, application engineers, and commissioners). To fulfill the requirements, a concept of the table-based formal specification language was presented as well as its applications. The table form specification is grounded on the industry practices basically and could reflect the characteristics of the aPS technical processes. By putting the variables in the columns and representing the state sequences in the rows, a program execution path can be described in a table. The generalizing concepts enable the table to abstract a number of behaviors (maximally infinite) of the same sort. Referencing the other cells, using IEC 61131-3 libraries, and timing abstraction notations allow for the table to be compact and understandable. Also, this approach aims at the accessibility from the automation engineers, so that they can easily generate and understand the specification. It was also presented that the approach could be applied various stages of the engineering to strengthen the quality management, namely implemented software verification, obtaining specification, and monitoring block generation. The tool to support the presented approach was implemented prototypically by being embedded in one of the major control software IDE. The following evaluation showed the feasibility of the approach to be applied to the existing systems for their specification and further utilization. Four use cases were

introduced as the application cases of the presented approaches : i) description of a single component module, ii) description of a multi-component module, iii) obtaining preliminary change specification from the existing IEC 61131-3 software, and iv) generating the monitoring function block from the specification. The result of the empirical study, conducted with the mechanical engineering department students as potential automation engineers, and the discussion with the industry experts enlightened that the presented approach has potential usefulness in the real industry field. Formal verification can complete the validation cases where testing cannot be entirely and exhaustively done due to the time limitation and the plant execution; the presented approaches, as a result, could promote an early application of regression verification through the incremental step-wise approach from the single component modules up to the complex compound modules. It was proven as promising by the interest of a company to apply the given approaches through a transfer project.

Although the evaluation of feasibility and the applicability have been conducted and the result turned to be positive, it was demonstrative level, and more case studies of the approach with various scales should be conducted. Here, the scale could be considered in various dimensions as experts argued. First, it could be regarding the size of the table. With more number of input/output and sequences, the table scales up horizontally (more columns) and vertically (more rows), respectively. The borderline of the accessibility from the user point of view depending on the table size is to be more precisely measured. This will also require how to manage the abstraction of the unnecessary or omit-able cells with specific effective notations. Second, the scale could be handled in the sense of POU architecture level. Especially in the viewpoint of application engineer, some detailed behavior of modules could be abstracted during the implementation, meaning not knowing the details. In this case, corresponding GTTs for detailed behaviors could be merged after being abstracted to get the specification of the higher level of POU (i.e., a composition of the smaller POU). Correspondingly, the merging techniques of GTTs should be approached in future research. This will allow to generate the specification of an integrated component block easily. The executed user evaluation was mainly with the potential users; however, the approach is to be applied in the real industry engineering workflows to hear the voice of the engineers, to be justified for its utility and usability. This includes the evaluation of the tool itself since it has not been assessed.

Additional evaluation would also be necessary to strengthen the justification of the approach. Regarding learning the specification approach, it would be necessary to measure the precise level of learnability. While the overall user perception regarding its complexity and satisfaction was shown as positive from the conductive evaluation, it was also found that many participants were still wavering over about the usage in practice (in spite of positive feedback on the average, it was

lowest among the results). Therefore, performing successive work to figure out the main reason for this phenomenon and the corresponding adaptation of the language would be necessary. In addition, to make the results of the qualitative assessment stronger, a further survey comparing other formal specification languages with a promising number of participants should be conducted.

For its applications, the generation of the specification base was approached on one of the IEC 61131-3 language, i.e., SFC (cf. use case 3). As the other languages, such as ST or LD, are also prevalently used in the field, specification mining from the software code in those languages (not only graphical but also textual ones) should also be approached. To complete the generated specification for improved accuracy, various skills could be applied and integrated, such as recognizing typical code patterns by the code analysis or learning through the signal traces, to also support extended inferencing ability for nested cases in various type of languages. Domain's typical behavior architectures such as OMAC states [Omac09] and ISA-88 [IEC97] architecture level could also be considered to decide the typical templates or patterns. Furthermore, the method of the monitoring block generation has a room to be improved overcoming the assumptions (cf. use case 4). Within the current approach, rows are not overlapped to consider only the deterministic cases, meaning the only row is active at one time. However, more rows could be active in the real cases non-deterministically. In this case, the monitoring should consider different possible cases at the same time; this will require modification of the generation rules of the monitoring block and also affect the performance of the monitoring. These extensions of the presented methods are to be the direction of further research. In addition, the approach could produce a huge synergy together with testing as an artifact related to the quality assurance activities. It was shown that test cases could be provided by GTT through instantiation. Significant criteria to be designated to generate the most effective test cases out of GTT should also be researched depending on the objectives and testing aspects for the cases where the dynamic analysis is more required and effective.

10. Literature

- [AbRo10] Abellan-Nebot, J.V. and Romero Subirón, F., "A review of machining monitoring systems based on artificial intelligence process models," In: *International Journal of Advanced Manufacturing Technology*, vol. 47, no. 1–4, pp. 237–257, 2010.
- [AHDR18] Aniculaesei, A., Howar, F., Denecke, P. and Rausch, A., "Automated generation of requirements-based test cases for an adaptive cruise control system," In: *2018 IEEE Workshop on Validation, Analysis and Evolution of Software Tests (VST)*, pp. 11–15, 2018.
- [AlDi92] Alur, R. and Dill, D., "The theory of timed automata," In: *Proceedings of REX: Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems)*, pp. 45–73, 1992.
- [Ansy21] ANSYS, I., "SCADE Suite: Integrated Model-Based Design & Development Environment | Ansys," [Online] Available: <https://www.ansys.com/products/embedded-software/ansys-scade-suite>, [Accessed: 2021-01-21].
- [Antl20] ANTLR, "ANTLR," [Online] Available: <https://www.antlr.org/>, [Accessed: 2021-01-21].
- [AuIP07] Autili, M., Inverardi, P. and Pelliccione, P., "Graphical scenarios for specifying temporal properties: an automated approach," In: *Automated Software Engineering*, vol. 14, no. 3, pp. 293–340, 2007.
- [BAGB14] Barišić, A., Amaral, V., Goulão, M. and Barroca, B., "Evaluating the Usability of Domain-Specific Languages," In: *Formal and Practical Aspects of Domain-Specific Languages*. IGI Global, pp. 386–407, 2014.
- [BaKa08] Baier, C. and Katoen, J.P., *Principles Of Model Checking*. The MIT Press, 2008.
- [BBFL13] Berard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L. and Schnoebelen, P., *Systems and software verification: model-checking techniques and tools*. Springer-Verlag Berlin Heidelberg, 2013.
- [BBKL98] Brink, K., Bun, L.J.G., van Katwijk, J., Lutje Spelberg, R.F. and Toetenel, W.J., "Automatic analysis of embedded systems specified in Astral," In: *Proceedings - 31st Hawaii International Conference on System Sciences*, pp. 177–186, 1998.
- [BCUV17] Beckert, B., Cha, S., Ulbrich, M., Vogel-Heuser, B. and Weigl, A., "Generalised Test Tables – a Practical Specification Language for Reactive Systems," In: *Proceedings - 13th International Conference on integrated Formal Methods*, pp. 875–882, 2017.
- [BEHL04] Bauer, N., Engell, S., Huuck, R., Lohmann, S., Lukoschus, B., Remelhe, M. and Stursberg, O., "Verification of PLC Programs Given as Sequential Function Charts," In: *Integration of Software Specification Techniques for Applications in*

- Engineering: Priority Program SoftSpez of the German Research Foundation (DFG), Final Report.* Springer Berlin Heidelberg, pp. 517–540, 2004.
- [Berg82] Berg, H.K., *Formal Methods of Program Verification and Specification.* Prentice Hall, 1982.
- [Beva95] Bevan, N., "Measuring usability as quality of use," In: *Software Quality Journal*, vol. 4, no. 2, pp. 115–130, 1995.
- [BGFF12] Bonato, M., Di Guglielmo, G., Fujita, M., Fummi, F. and Pravadelli, G., "Dynamic property mining for embedded software," In: *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis - CODES+ISSS '12*, p. 187, 2012.
- [Bits01] Bitsch, F., "Safety Patterns – The Key to Formal Specification of Safety Requirements," In: *Proceedings - 20th International Conference on Computer Safety, Reliability, and Security (SAFECOMP)*, pp. 176–189, 2001.
- [BORZ99] du Bousquet, L., Ouabdesselam, F., Richier, J.L. and Zuanon, N., "Lutess: a specification-driven testing environment for synchronous software," In: *Proceedings - 21st international conference on Software engineering (ICSE)*, pp. 267–276, 1999.
- [BoVH05] Bouzon, G., Vyatkin, V. and Hanisch, H.M., "Timing Diagram Specifications In Modular Modeling of Industrial Automation Systems," In: *IFAC Proceedings Volumes*, vol. 38, no. 1, pp. 80–85, 2005.
- [Broo96] Brooke, J., "SUS-A quick and dirty usability scale," In: *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.
- [BrSt01] Broy, M. and Stølen, K., *Specification and Development of Interactive Systems. , Monographs in Computer Science* Springer New York, 2001.
- [BUVW15] Beckert, B., Ulbrich, M., Vogel-Heuser, B. and Weigl, A., "Regression verification for programmable logic controller software," In: *Proceedings - 17th International Conference on Formal Engineering Methods (ICFEM)*, pp. 234–251, 2015.
- [BuVy17] Buzhinsky, I. and Vyatkin, V., "Automatic Inference of Finite-State Plant Models From Traces and Temporal Properties," In: *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1521–1530, 2017.
- [BVFS19] Bougouffa, S., Vogel-Heuser, B., Fischer, J., Schaefer, I. and Li, H., "Visualization of Variability Analysis of Control Software From Industrial Automation Systems," In: *Proceedings - IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp. 3357–3364, 2019.
- [CaEB05] Carew, D., Exton, C. and Buckley, J., "An empirical investigation of the comprehensibility of requirements specifications," In: *Proceedings - International Symposium on Empirical Software Engineering (ISESE)*, pp. 256–265, 2005.

- [CaHN11] Campetelli, A., Holzl, F. and Neubeck, P., "User-friendly model checking integration in model-based development," In: *Proceedings - 24th International Conference on Computer Applications in Industry and Engineering (CAINE)*, pp. 199–204, 2011.
- [CaMa09] Campos, J.C. and Machado, J., "Pattern-based analysis of automated production systems," In: *IFAC Proceedings Volumes*, vol. 13, no. 1, pp. 972–977, 2009.
- [CCDM05] Caffall, D.S., Cook, T., Drusinsky, D., Michael, J.B., Shing, M.T. and Sklavounos, N., 2005, "Formal Specification and Run-time Monitoring within the Ballistic Missile Defense Project," Naval Postgraduate School, California[Online] Available: <http://hdl.handle.net/10945/25583>.
- [CGKT16] Champion, A., Gurfinkel, A., Kahsai, T. and Tinelli, C., "CoCoSpec: A Mode-Aware Contract Language for Reactive Systems," In: *Proceedings - 14th International Conference on Software Engineering and Formal Methods (SEFM)*, pp. 347–366, 2016.
- [CHVB18] Clarke, E.M., Henzinger, T.A., Veith, H. and Bloem, R., *Handbook of model checking*. Springer, Cham, 2018.
- [Code20a] CODESYS GmbH, "Codesys Company - Facts & Figures," [Online] Available: <https://www.codesys.com/company/facts-figures.html>, [Accessed: 2020-12-29].
- [Code20b] CODESYS GmbH, "CODESYS Store International - CODESYS Test Manager," [Online] Available: <https://store.codesys.com/codesys-test-manager.html?store=en>, [Accessed: 2021-02-05].
- [Code20c] CODESYS GmbH, "CODESYS AUTOMATION PLATFORM," [Online] Available: <https://www.codesys.com/products/codesys-engineering/automation-platform.html>, [Accessed: 2021-01-22].
- [CPHP87] Caspi, P., Pilaud, D., Halbwachs, N. and Plaice, J.A., "LUSTRE: a declarative language for real-time programming," In: *Proceedings - 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages (POPL)*, pp. 178–188, 1987.
- [CUVW17] Cha, S., Ulewicz, S., Vogel-Heuser, B., Weigl, A., Ulbrich, M. and Beckert, B., "Generation of Monitoring Functions in Production Automation Using Test Specifications," In: *Proceedings - 15th IEEE International Conference on Industrial Informatics (INDIN)*, pp. 339–344, 2017.
- [CUWB19] Cha, S., Ulbrich, M., Weigl, A., Beckert, B., Land, K. and Vogel-Heuser, B., "On the Preservation of the Trust by Regression Verification of PLC software for Cyber-Physical Systems of Systems," In: *Proceedings - 17th IEEE International Conference on Industrial Informatics (INDIN)*, pp. 413–418, 2019.
- [CVWU21] Cha, S., Vogel-Heuser, B., Weigl, A., Ulbrich, M. and Beckert, B., "Table-based formal specification approaches for control engineers – empirical studies of

- usability," In: *IET Cyber-Physical Systems: Theory & Applications*, 2021. (online-first: <https://doi.org/10.1049/cps2.12017>)
- [CWAC96] Clarke, E.M., Wing, J.M., Alur, R., Clarke, E., Cleaveland, R., Dill, D., Emerson, A., Garland, S. et al., "Formal methods: State of the art and future directions," In: *ACM Computing Surveys*, vol. 28, no. 4, pp. 626–643, 1996.
- [CWUB18a] Cha, S., Weigl, A., Ulbrich, M., Beckert, B. and Vogel-Heuser, B., "Applicability of Generalized Test Tables: A Case Study Using the Manufacturing System Demonstrator xPPU," In: *Automatisierungstechnik*, vol. 66, no. 10, pp. 834–848, 2018.
- [CWUB18b] Cha, S., Weigl, A., Ulbrich, M., Beckert, B. and Vogel-Heuser, B., "Achieving delta description of the control software for an automated production system evolution," In: *Proceedings - 14th IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 1170–1176, 2018.
- [DaAl92] David, R. and Alla, H., *Petri nets and Grafcet: tools for modelling discrete event systems*. Prentice Hall, New York, 1992.
- [DaBM15] Darvas, D., Blanco Viñuela, E. and Majzik, I., "A formal specification method for PLC-based applications," In: *Proceedings - 15th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS)*, pp. 907–910, 2015.
- [DaHa99] Damm, W. and Harel, D., "Lsc's: Breathing Life Into Message Sequence Charts," In: *Proceedings - 3rd International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, pp. 293–311, 1999.
- [DaMB16] Darvas, D., Majzik, I. and Blanco Viñuela, E., "Formal Verification of Safety PLC Based Control Software," In: *Proceedings - 12th International Conference (IFM)*, pp. 508–522, 2016.
- [DaMV15] Darvas, D., Majzik, I. and Viñuela, E.B., "Requirements towards a formal specification language for PLCs," In: *Proceedings - 22nd PhD Mini-Symposium of the Budapest University of Technology and Economics, Department of Measurement and Information Systems*, pp. 18–21, 2015.
- [DaMV17] Darvas, D., MajzikIstván, I. and Viñuela, E.B., "Well-formedness and invariant checking of PLCspecif specifications," In: *Proceedings - 24th PhD Mini-Symposium of the Budapest University of Technology and Economics, Department of Measurement and Information Systems*, pp. 10–13, 2017.
- [Davi89] Davis, F.D., "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology," In: *MIS Quarterly*, vol. 13, no. 3, p. 319, 1989.
- [DaVM16] Darvas, D., Viñuela, E.B. and Majzik, I., "PLC code generation based on a formal specification language," In: *Proceedings - 14th IEEE International Conference on Industrial Informatics (INDIN)*, pp. 389–396, 2016.

- [DDHM02] Deng, X., Dwyer, M.B., Hatcliff, J. and Mizuno, M., "Invariant-based specification, synthesis, and verification of synchronization in concurrent programs," In: *Proceedings - 24th international conference on Software engineering (ICSE)*, pp. 442–452, 2002.
- [DeK102] van Deursen, A. and Klint, P., "Domain-specific language design requires feature descriptions," In: *Journal of Computing and Information Technology*, vol. 10, no. 1, pp. 1–17, 2002.
- [DKKK12] Durdik, Z., Klatt, B., Koziolok, H., Krogmann, K., Stammel, J. and Weiss, R., "Sustainability guidelines for long-living software systems," In: *Proceedings - 28th IEEE International Conference on Software Maintenance (ICSM)*, IEEE, pp. 517–526, 2012.
- [DoHF14] Dokhanchi, A., Hoxha, B. and Fainekos, G., "On-Line Monitoring for Temporal Logic Robustness," In: *Proceedings - 5th International Conference on Runtime Verification (RV)*, pp. 231–246, 2014.
- [DwAC99] Dwyer, M.B., Avrunin, G.S. and Corbett, J.C., "Patterns in property specifications for finite-state verification," In: *Proceedings - 2nd workshop on Formal methods in software practice (FMSP)*, pp. 411–420, 1999.
- [EiFi18] Eisner, C. and Fisman, D., "Functional Specification of Hardware via Temporal Logic," In: *Handbook of Model Checking*. Springer, Cham, pp. 795–829, 2018.
- [EiFP09] Eibensteiner, F., Findenig, R. and Pfaff, M., "SynPSL: Behavioral Synthesis of PSL Assertions," In: ., pp. 69–74, 2009.
- [EmCl82] Emerson, E.A. and Clarke, E.M., "Using branching time temporal logic to synthesize synchronization skeletons," In: *Science of Computer Programming*, vol. 2, no. 3, pp. 241–266, 1982.
- [EmHa86] Emerson, E.A. and Halpern, J.Y., "'Sometimes' and 'not never' revisited," In: *Journal of the ACM*, vol. 33, no. 1, pp. 151–178, 1986.
- [FBSS18] Fischer, J., Bougouffa, S., Schlie, A., Schaefer, I. and Vogel-Heuser, B., "A Qualitative Study of Variability Management of Control Software for Industrial Automation Systems," In: *Proceedings - IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 615–624, 2018.
- [FKWV19] Feldmann, S., Kernschmidt, K., Wimmer, M. and Vogel-Heuser, B., "Managing inter-model inconsistencies in model-based systems engineering: Application in automated production systems engineering," In: *Journal of Systems and Software*, vol. 153, Elsevier Inc., pp. 105–134, 2019.
- [FMFR11] Falcone, Y., Mounier, L., Fernandez, J.C. and Richier, J.L., "Runtime enforcement monitors: composition, synthesis, and enforcement abilities," In: *Formal Methods in System Design*, vol. 38, no. 3, pp. 223–262, 2011.

- [Frey00] Frey, G., "Automatic implementation of Petri net based control algorithms on PLC," In: *Proceedings - 2000 American Control Conference (ACC)*, pp. 2819–2823, 2000.
- [FrLi00] Frey, G. and Litz, L., "Formal methods in PLC programming," In: *Proceedings - IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp. 2431–2436, 2000.
- [GaSu10] Gabel, M. and Su, Z., "Online inference and enforcement of temporal properties," In: *Proceedings - 32nd ACM/IEEE International Conference on Software Engineering (ICSE)*, p. 15, 2010.
- [GhFu09] Gharehbaghi, A.M. and Fujita, M., "Transaction-based debugging of system-on-chips with patterns," In: *Proceedings - IEEE International Conference on Computer Design*, pp. 186–192, 2009.
- [GhKe91] Ghezzi, C. and Kemmerer, R.A., "ASTRAL: An assertion language for specifying realtime systems," In: *Proceedings - 3rd European Software Engineering Conference (ESEC)*, pp. 122–146, 1991.
- [GiDi93] Giua, A. and DiCesare, F., "GRAFCET and Petri Nets in Manufacturing," In: *Intelligent Manufacturing: Programming Environments for CIM*. Springer, London, pp. 153–176, 1993.
- [GoMH16] González, E., Marichal, R. and Hamilton, A., "Ontology-based approach to Basic Grafcet formalization," In: *Journal of the Chinese Institute of Engineers*, vol. 39, no. 8, pp. 946–953, 2016.
- [Gora06] Goranko, V., "Logic in Computer Science: Modelling and Reasoning About Systems," In: *Journal of Logic, Language and Information*, vol. 16, no. 1, pp. 117–120, 2006.
- [GrLa06] Gruhn, V. and Laue, R., "Patterns for Timed Property Specifications," In: *Electronic Notes in Theoretical Computer Science*, vol. 153, no. 2, pp. 117–133, 2006.
- [GüWF08] Güttel, K., Weber, P. and Fay, A., "Automatic generation of PLC code beyond the nominal sequence," In: *Proceedings - IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1277–1284, 2008.
- [Hack18] Hackenberg, G., *Test-driven conceptual design of cyber-physical manufacturing systems*. Dissertation, Technischen Universität München, Munich, Germany. Institut für Informatik, 2018.
- [Halb05] Halbwachs, N., "A synchronous language at work: the story of Lustre," In: *Proceedings - 2nd ACM/IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE)*, pp. 3–12, 2005.
- [HaMM16] Hauzar, D., Marché, C. and Moy, Y., "Counterexamples from proof failures in SPARK," In: *Proceedings - 14th International Conference on Software Engineering and Formal Methods (SEFM)*, pp. 215–233, 2016.

- [Hans15] Hanssen, D.H., *Programmable logic controllers: a practical approach to IEC61131-3 using Codesys*. Wiley, 2015.
- [HBZY19] Huang, Y., Bu, X., Zhu, G., Ye, X., Zhu, X. and Shi, J., "KST: Executable Formal Semantics of IEC 61131-3 Structured Text for Verification," In: *IEEE Access*, vol. 7, pp. 14593–14602, 2019.
- [HCLM14] Hackenberg, G., Campetelli, A., Legat, C., Mund, J., Teufl, S. and Vogel-Heuser, B., "Formal Technical Process Specification and Verification for Automated Production Systems," In: *Proceedings - 8th International Conference on System Analysis and Modeling (SAM)*, pp. 287–303, 2014.
- [HCRP91] Halbwachs, N., Caspi, P., Raymond, P. and Pilaud, D., "The synchronous data flow programming language LUSTRE," In: *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1305–1320, 1991.
- [HLLF14] Haubeck, C., Lamersdorf, W., Ladiges, J., Fay, A., Fuchs, J., Legat, C. and Vogel-Heuser, B., "Interaction of model-driven engineering and signal-based online monitoring of production systems: Towards Requirement-aware evolution," In: *Proceedings - 40th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, pp. 2571–2577, 2014.
- [HLLT06] Hanisch, H.M., Lobov, A., Lastra, J.L.M., Tuokko, R. and Vyatkin, V., "Formal validation of intelligent-automated production systems: towards industrial applications," In: *International Journal of Manufacturing Technology and Management (IJMTM)*, vol. 8, no. 1/2/3, pp. 75–106, 2006.
- [Hoar78] Hoare, C.A.R., "Communicating sequential processes," In: *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, 1978.
- [Holz02] Holzmann, G.J., "The logic of bugs," In: *Proceedings - 10th ACM SIGSOFT symposium on Foundations of software (SIGSOFT/FSE)*, pp. 81–87, 2002.
- [HSFS17] Hildebrandt, C., Scholz, A., Fay, A., Schröder, T., Hadlich, T., Diedrich, C., Dubovy, M., Eck, C. et al., "Semantic Modeling for Collaboration and Cooperation of Systems in the production domain," In: *Proceedings - 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, 2017.
- [HuKi11] Huang, H. and Kirchner, H., "Formal Specification and Verification of Modular Security Policy Based on Colored Petri Nets," In: *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 6, pp. 852–865, 2011.
- [HuRy04] Huth, M. and Ryan, M., *Logic in Computer Science: Modelling and Reasoning About Systems (2nd. Ed.)*. Cambridge University Press, Cambridge, 2004.
- [Huuc05] Huuck, R., "Semantics and Analysis of Instruction List Programs," In: *Electronic Notes in Theoretical Computer Science*, vol. 115, pp. 3–18, 2005.

- [IEC09] IEC, 2009, "IEC 61131-3 Programmable Logic Controllers – Part 3: Programming Languages."
- [IEC13] IEC, 2013, "IEC 60050-351:2013 - International Electrotechnical Vocabulary (IEV) - Part 351: Control technology."
- [IEC97] IEC, 1997, "IEC 61512-1:1997 - Batch control - Part 1: Models and terminology."
- [IEEE10] IEEE, 2010, "IEEE 1850-2010 - Standard for Property Specification Language (PSL)," [Online] Available: <https://ieeexplore.ieee.org/servlet/opac?punumber=5445949>.
- [IEEE14A] IEEE, 2014, "730-2014 IEEE Standard for Software Quality Assurance Processes," [Online] Available: <https://ieeexplore.ieee.org/document/6835311>.
- [IEEE14B] IEEE Computer Society, 2014, "Guide to the Software Engineering - Body of Knowledge, Version 3.0," [Online] Available: <http://www.swebok.org>.
- [ISO10] ISO, 2010, "ISO/IEC/IEEE 24765:2010 Systems and Software Engineering—Vocabulary," [Online] Available: <https://www.iso.org/standard/50518.html>.
- [ISO11] ISO, 2011, "ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models," [Online] Available: <https://www.iso.org/standard/35733.html>.
- [ISO16] ISO, 2016, "ISO 13485:2016 Medical devices — Quality management systems — Requirements for regulatory purposes," [Online] Available: <https://www.iso.org/obp/ui/#iso:std:iso:13485:ed-3:v1:en>.
- [ISPE20] ISPE (International Society for Pharmaceutical Engineering), 2020, "Good Automated Manufacturing Practice (GAMP)," [Online] Available: <https://ispe.org/>.
- [JoTi10] John, K.H. and Tiegelkamp, M., *IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Decision-Making Aids*. Springer, Berlin, Heidelberg, 2010.
- [JSCL14] Jee, E., Shin, D., Cha, S., Lee, J.-S. and Bae, D.-H., "Automated test case generation for FBD programs implementing reactor protection system software," In: *Software Testing, Verification and Reliability*, vol. 24, no. 8, pp. 608–628, 2014.
- [JSSF17] Julius, R., Schürenberg, M., Schumacher, F. and Fay, A., "Transformation of GRAFCET to PLC code including hierarchical structures," In: *Control Engineering Practice*, vol. 64, pp. 173–194, 2017.
- [JTFN19] Julius, R., Trenner, T., Fay, A., Neidig, J. and Hoang, X.L., "A meta-model based environment for GRAFCET specifications," In: *Proceedings - IEEE International Systems Conference (SysCon)*, pp. 1–7, 2019.
- [KaFV04] Katzke, U., Fischer, K. and Vogel-Heuser, B., "Development and Evaluation of a Model for Modular automation in Plant Manufacturing," In: *Proceedings - 10th*

- International Conference on Information Systems Analysis and Synthesis (CITSA)*, pp. 15–20, 2004.
- [KBKW12] Kabra, A., Bhattacharjee, A., Karmakar, G. and Wakankar, A., "Formalization of sequential function chart as synchronous model in LUSTRE," In: *Proceedings - 2012 3rd National Conference on Emerging Trends and Applications in Computer Science, NCETACS-2012*, pp. 115–120, 2012.
- [KBMA15] Kustarev, P., Bykovskii, S., Milin, V. and Antonov, A., "Model-Driven Runtime Embedded Monitoring for Industrial Controllers," In: *Proceedings - IEEE Trustcom/BigDataSE/ISPA*, pp. 281–286, 2015.
- [KBSK10] Kainz, G., Buckl, C., Sommer, S. and Knoll, A., "Model-to-metamodel transformation for the development of component-based systems," In: *Proceedings - 13th international conference on Model driven engineering languages and systems: Part II*, pp. 391–405, 2010.
- [KeMa19] Keliris, A. and Maniatakos, M., "ICSREF: A Framework for Automated Reverse Engineering of Industrial Control Systems Binaries," In: *Proceedings - Network and Distributed System Security Symposium*, 2019.
- [Kern19] Kernschmidt, K.E.T., *Interdisciplinary structural modeling of mechatronic production systems using SysML4Mechatronics*. Dissertation, Technischen Universität München, Munich, Germany. Lehrstuhl für Automatisierung und Informationssysteme, 2019.
- [KHPL05] Kugler, H., Harel, D., Pnueli, A., Lu, Y. and Bontemps, Y., "Temporal Logic for Scenario-Based Specifications," In: *Proceedings - 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 445–460, 2005.
- [KKST13] Kim, I., Kim, T., Sung, M., Tisserant, E., Bessard, L. and Choi, C., "An open-source development environment for industrial automation with EtherCAT and PLCopen motion control," In: *Proceedings - 18th IEEE Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 1–4, 2013.
- [KoCh05] Konrad, S. and Cheng, B.H.C., "Real-time specification patterns," In: *Proceedings - 27th international conference on Software engineering (ICSE)*, p. 372, 2005.
- [KoTV12] Kormann, B., Tikhonov, D. and Vogel-Heuser, B., "Automated PLC Software Testing using adapted UML Sequence Diagrams," In: *IFAC Proceedings Volumes*, vol. 45, IFAC, no. 6, pp. 1615–1621, 2012.
- [KrPA03] Kristoffersen, K.J., Pedersen, C. and Andersen, H.R., "Runtime Verification of Timed LTL using Disjunctive Normalized Equation Systems," In: *Electronic Notes in Theoretical Computer Science*, vol. 89, no. 2, pp. 210–225, 2003.
- [KuMB09] Kumar, R., Mercer, E.G. and Bunker, A., "Improving Translation of Live Sequence Charts to Temporal Logic," In: *Electronic Notes in Theoretical Computer Science*, vol. 250, no. 1, pp. 137–152, 2009.

- [LÅFE14] Ljungkrantz, O., Åkesson, K., Fabian, M. and Ebrahimi, A.H., "An empirical study of control logic specifications for programmable logic controllers," In: *Empirical Software Engineering*, vol. 19, no. 3, pp. 655–677, 2014.
- [LaFL16] Ladiges, J., Fay, A. and Lamersdorf, W., "Automated Determining of Manufacturing Properties and Their Evolutionary Changes from Event Traces," In: *Intelligent Industrial Systems*, vol. 2, no. 2, pp. 163–178, 2016.
- [LAFY10] Ljungkrantz, O., Åkesson, K., Fabian, M. and Yuan, C., "A formal specification language for PLC-based control logic," In: *2010 8th IEEE International Conference on Industrial Informatics*, pp. 1067–1072, 2010.
- [LaGö99] Lauber, R. and Göhner, P., *Prozessautomatisierung 1*. Springer, Berlin, Heidelberg, 1999.
- [LaMM20] Lanotte, R., Merro, M. and Munteanu, A., "Runtime Enforcement for Control System Security," In: *Proceedings - 33rd IEEE Computer Security Foundations Symposium (CSF)*, pp. 246–261, 2020.
- [Lams88] Lamsweerde, A. Van, "Formal specification," In: *Annual Review in Automatic Programming*, vol. 14, no. 1, p. 83, 1988.
- [LHFL15] Ladiges, J., Haubeck, C., Fay, A. and Lamersdorf, W., "Learning Behaviour Models of Discrete Event Production Systems from Observing Input/Output Signals," In: *IFAC-PapersOnLine*, vol. 48, Elsevier Ltd., no. 3, pp. 1565–1572, 2015.
- [Li14] Li, W., *Specification Mining: New Formalisms, Algorithms and Applications*. Dissertation, University of California, Berkeley, USA. EECS Department, 2014.
- [LMCH14] Legat, C., Mund, J., Campetelli, A., Hackenberg, G., Folmer, J., Schütz, D., Broy, M. and Vogel-Heuser, B., "Interface behavior modeling for automatic verification of industrial automation systems' functional conformance," In: *At-Automatisierungstechnik*, vol. 62, no. 11, pp. 815–825, 2014.
- [LoMK07] Lo, D., Maoz, S. and Khoo, S.C., "Mining modal scenario-based specifications from execution traces of reactive systems," In: *Proceedings - 22nd IEEE/ACM international conference on Automated software engineering (ASE)*, p. 465, 2007.
- [Lude89] Ludewig, J., "Languages, Methods, and Tools for Software Specification," In: *Hardware and Software for Real Time Process Control*. Elsevier Science Publishers B.V., pp. 225–256, 1989.
- [MaAr00] Marre, B. and Arnould, A., "Test sequences generation from LUSTRE descriptions: GATEL," In: *Proceedings - 15th IEEE International Conference on Automated Software Engineering (ASE)*, pp. 229–237, 2000.
- [McBu00] McFarlane, D.C. and Bussmann, S., "Developments in holonic production planning and control," In: *Production Planning and Control*, vol. 11, no. 6, pp. 522–536, 2000.

- [Mcla13] McLaughlin, S., "CPS: stateful policy enforcement for control system device usage," In: *Proceedings - 29th Annual Computer Security Applications Conference*, pp. 109–118, 2013.
- [Micr18] Microsoft Corporation, "Introduction to WPF in Visual Studio," [Online] Available: <https://docs.microsoft.com/en-us/dotnet/desktop/wpf/getting-started/introduction-to-wpf-in-vs?view=netframeworkdesktop-4.8>, [Accessed: 2021-01-22].
- [MJBC17] Mund, J., Junker, M., Bougouffa, S., Cha, S. and Vogel-Heuser, B., "Model-based availability analysis for automated production systems," In: *Proceedings - 15th ACM/IEEE International Conference on Formal Methods and Models for System Design*, pp. 46–55, 2017.
- [MoGu96] Moore, K.E. and Gupta, S.M., "Petri net models of flexible and automated manufacturing systems: a survey," In: *International Journal of Production Research*, vol. 34, no. 11, pp. 3001–3035, 1996.
- [MPBC92] Moon, I., Powers, G.J., Burch, J.R. and Clarke, E.M., "Automatic verification of sequential control systems using temporal logic," In: *AICChE Journal*, vol. 38, no. 1, pp. 67–75, 1992.
- [MuGM05] Music, G., Gradisar, D. and Matko, D., "IEC 61131-3 Compliant Control Code Generation from Discrete Event Models," In: *Proceedings - IEEE International Symposium on, Mediterrean Conference on Control and Automation Intelligent Control*, pp. 346–351, 2005.
- [Mura89] Murata, T., "Petri nets: Properties, analysis and applications," In: *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.
- [NgMT12] Nguyen, C.D., Marchetto, A. and Tonella, P., "Combining model-based and combinatorial testing for effective test case generation," In: *Proceedings - International Symposium on Software Testing and Analysis (ISSTA)*, p. 100, 2012.
- [OdMB06] Oddos, Y., Morin-Allory, K. and Borriore, D., "On-Line Test Vector Generation from Temporal Constraints Written in PSL," In: *Proceedings - International Conference on Very Large Scale Integration (IFIP)*, pp. 397–402, 2006.
- [Omac09] OMAC (Organization for Machine Automation and Control), "Packaging Workgroup – Organization for Machine Automation and Control," [Online] Available: <http://omac.org/workgroups/packaging-workgroup/>.
- [PGLN20] Pagetti, H.B., Garoche, P.-L., Loquenl, T., Noulard, É. and Pagetti, C., "CoCoSim, a code generation framework for control/command applications An overview of CoCoSim for multi-periodic discrete Simulink models," In: *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, 2020.
- [PaTK01] Park, E., Tilbury, D.M. and Khargonekar, P.P., "A modeling and analysis methodology for modular logic controllers of machining systems using Petri net formalism," In: *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, vol. 31, no. 2, pp. 168–188, 2001.

- [PiPn18] Piterman, N. and Pnueli, A., "Temporal Logic and Fair Discrete Systems," In: *Handbook of Model Checking*. Springer, Cham, pp. 27–73, 2018.
- [PiQu13] Pill, I. and Quaritsch, T., "Behavioral diagnosis of LTL specifications at operator level," In: *Proceedings - 23rd International Joint Conference on Artificial Intelligence*, pp. 1053–1059, 2013.
- [Pnue77] Pnueli, A., "The temporal logic of programs," In: *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*, vol. 1977-October, 1977.
- [PPBV16] Pakonen, A., Pang, C., Buzhinsky, I. and Vyatkin, V., "User-friendly formal specification languages – conclusions drawn from industrial experience on model checking," In: *Proceedings - IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016.
- [QaTP17] Qamsane, Y., Tajer, A. and Philippot, A., "A synthesis approach to distributed supervisory control design for manufacturing systems with Grafset implementation," In: *International Journal of Production Research*, vol. 55, no. 15, pp. 4283–4303, 2017.
- [QuSi82] Queille, J.P. and Sifakis, J., "Specification and verification of concurrent systems in CESAR," In: *Proceedings - 5th International Symposium on Programming*, pp. 337–351, 1982.
- [RBKP13] Rochez, J., Blanco-Vinuela, E.B., Koutli, M. and Petrou, T., "Opening the Floor to PLCs and IPCs: CODESYS in UNICOS," In: *Proceedings - 14th International Conference on Accelerator & Large Experimental Physics Control Systems*, 2013.
- [ReWü07] Reinhart, G. and Wünsch, G., "Economic application of virtual commissioning to mechatronic production systems," In: *Production Engineering*, vol. 1, no. 4, pp. 371–379, 2007.
- [RNHW98] Raymond, P., Nicollin, X., Halbwachs, N. and Weber, D., "Automatic testing of reactive systems," In: *Proceedings - 19th IEEE Real-Time Systems Symposium*, pp. 200–209, 1998.
- [Rösc16] Rösch, S., *Model-based testing of fault scenarios in production automation*. Dissertation, Technischen Universität München, Munich, Germany. Lehrstuhl für Automatisierung und Informationssysteme, 2016.
- [RöVo17] Rösch, S. and Vogel-Heuser, B., "A Light-Weight Fault Injection Approach to Test Automated Production System PLC Software in Industrial Practice," In: *Control Engineering Practice*, vol. 58, Elsevier, no. Jan., pp. 12–23, 2017.
- [Rozi11] Rozier, K.Y., "Linear Temporal Logic Symbolic Model Checking," In: *Computer Science Review*, vol. 5, no. 2, pp. 163–203, 2011.
- [RSPG07] Razali, R., Snook, C.F., Poppleton, M.R., Garratt, P.W. and Walters, R., "Experimental Comparison of the Comprehensibility of a UML-based Formal

- Specification versus a Textual One," In: *Proceedings - 11th international conference on Evaluation and Assessment in Software Engineering*, 2007.
- [RTSV14] Rösch, S., Tikhonov, D., Schütz, D. and Vogel-Heuser, B., "Model-based testing of PLC software: test of plants' reliability by using fault injection on component level," In: *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 3509–3515, 2014.
- [SACO02] Smith, R.L., Avrunin, G.S., Clarke, L.A. and Osterweil, L.J., "PROPEL: an approach supporting property elucidation," In: *Proceedings - 24th International Conference on Software Engineering (ICSE)*, pp. 11–21, 2002.
- [SaLo05] Sarshar, K. and Loos, P., "Comparing the Control-Flow of EPC and Petri Net from the End-User Perspective," In: *Proceedings - 3rd International Conference on Business Process Management (BPM)*, pp. 434–439, 2005.
- [SAVF17] Spindler, M., Aicher, T., Vogel-Heuser, B. and Fottner, J., "Erstellung von Steuerungssoftware für automatisierte Materialflusssysteme per Drag & Drop (en: Engineering the Control Software of Automated Material Handling Systems via Drag & Drop)," In: *Logistics Journal*, vol. 2017, pp. 9–14, 2017.
- [Scha13] Schalles, C., "A Framework for Usability Evaluation of Modeling Languages," In: *Usability Evaluation of Modeling Languages: An Empirical Research Study*. Springer Gabler, Wiesbaden, pp. 43–68, 2013.
- [ScSF13] Schumacher, F., Schrock, S. and Fay, A., "Tool support for an automatic transformation of GRAFCET specifications into IEC 61131-3 control code," In: *Proceedings - 18th IEEE Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 1–4, 2013.
- [Sesh12] Seshia, S.A., "Sciduction: Combining induction, deduction, and structure for verification and synthesis," In: *Proceedings - Design Automation Conference*, IEEE, pp. 356–365, 2012.
- [ShLo88] Shanahan, T. and Lomax, R., "A developmental comparison of three theoretical models of the reading-writing relationship," In: *Research in the Teaching of English*, vol. 22, no. 2, pp. 196–212, 1988.
- [ShVy20] Shatrov, V. and Vyatkin, V., "Formal Verification of IEC 61499 Enhanced with Timed Events," In: *Proceedings - 11th Advanced Doctoral Conference on Computing, Electrical and Industrial Systems (DoCEIS)*, pp. 168–178, 2020.
- [SmHE01] Smith, M.H., Holzmann, G.J. and Etessami, K., "Events and constraints: a graphical editor for capturing logic requirements of programs," In: *Proceedings - 5th IEEE International Symposium on Requirements Engineering*, pp. 14–22, 2001.
- [SMZS16] Steinegger, M., Melik-Merkumians, M., Zajc, J. and Schitter, G., "Automatic generation of diagnostic handling code for decentralized PLC-based control architectures," In: *Proceedings - 21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, 2016.

- [SnHa04] Snook, C.F. and Harrison, R., "Experimental comparison of the comprehensibility of a Z specification and its implementation in Java," In: *Information and Software Technology*, vol. 46, no. 14, pp. 955–971, 2004.
- [Somm15] Sommerville, I., *Software Engineering (10th Ed.)*. Pearson, 2015.
- [SPGV19] Sinha, R., Patil, S., Gomes, L. and Vyatkin, V., "A Survey of Static Formal Methods for Building Dependable Industrial Automation Systems," In: *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 3772–3783, 2019.
- [SPMK15] Sinha, R., Pang, C., Martinez, G.S., Kuronen, J. and Vyatkin, V., "Requirements-Aided Automatic Test Case Generation for Industrial Cyber-physical Systems," In: *Proceedings - 20th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pp. 198–201, 2015.
- [StGo08] Strichman, O. and Godlin, B., "Regression Verification - A Practical Way to Verify Programs," In: *Proceedings - Working Conference on Verified Software: Theories, Tools, and Experiments (VSTTE)*, pp. 496–501, 2008.
- [SüVZ13] Sünder, C., Vyatkin, V. and Zoitl, A., "Formal verification of downtimeless system evolution in embedded automation controllers," In: *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 1, p. 17, 2013.
- [Thra10] Thramboulidis, K., "The 3+1 SysML View-Model in Model Integrated Mechatronics," In: *Journal of Software Engineering and Applications*, vol. 3, no. 2, pp. 109–118, 2010.
- [TNLM14] Teruel, M.A., Navarro, E., López-Jaquero, V., Montero, F. and González, P., "A CSCW Requirements Engineering CASE Tool: Development and usability evaluation," In: *Information and Software Technology*, vol. 56, no. 8, pp. 922–949, 2014.
- [TuSc05] Tuerk, T. and Schneider, K., "From PSL to LTL: A Formal Validation in HOL," In: *Proceedings - International Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, pp. 342–357, 2005.
- [Ulew18] Ulewicz, S., *Test Coverage Assessment for Semi-Automatic System Testing and Regression Testing Support in Production Automation*. Dissertation, Technischen Universität München, Munich, Germany. Lehrstuhl für Automatisierung und Informationssysteme, 2018.
- [UIVo15] Ulewicz, S. and Vogel-Heuser, B., "Automatisiertes Testen von Sondermaschinen – von der Modulbibliothek bis zur Anlage (en: Automated testing of special purpose machines - from the module library to the plant)," In: *Tagungsband Automation Symposium*, pp. 53–65, 2015.
- [UIVo18a] Ulewicz, S. and Vogel-Heuser, B., "Industrially Applicable System Regression Test Prioritization in Production Automation," In: *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 4, pp. 1839–1851, 2018.

- [UIVo18b] Ulewicz, S. and Vogel-Heuser, B., "Increasing system test coverage in production automation systems," In: *Control Engineering Practice*, vol. 73, no. Mar., pp. 171–185, 2018.
- [Usbu20] U.S. BUREAU OF LABOR STATISTICS, "Mechanical Engineers - Occupational Outlook Handbook," [Online] Available: <https://www.bls.gov/OOH/architecture-and-engineering/mechanical-engineers.htm>, [Accessed: 2020-06-17].
- [UUWK16] Ulewicz, S., Ulbrich, M., Weigl, A., Kirsten, M., Wiebe, F., Beckert, B. and Vogel-Heuser, B., "A verification-supported evolution approach to assist software application engineers in industrial factory automation," In: *Proceedings - IEEE International Symposium on Assembly and Manufacturing (ISAM)*, pp. 19–25, 2016.
- [VDFJ14] Vogel-Heuser, B., Diedrich, C., Fay, A., Jeschke, S., Kowalewski, S., Wollschlaeger, M. and Göhner, P., "Challenges for Software Engineering in Automation," In: *Journal of Software Engineering and Applications*, vol. 07, no. 05, pp. 440–451, 2014.
- [VERE10] Verein Deutscher Ingenieure (VDI) e.V., 2010, "VDI/VDE 3695: Engineering of Industrial Plants – Evaluation and Optimization – Subject Processes," .
- [VFAM15] Vogel-Heuser, B., Folmer, J., Aicher, T., Mund, J. and Rehberger, S., "Coupling simulation and model checking to examine selected mechanical constraints of automated production systems," In: *Proceeding - 2015 IEEE International Conference on Industrial Informatics, INDIN 2015*, pp. 37–42, 2015.
- [VFFL15] Vogel-Heuser, B., Feldmann, S., Folmer, J., Ladiges, J., Fay, A., Lity, S., Tichy, M., Kowal, M. et al., "Selected challenges of software evolution for automated production systems," In: *Proceeding - 2015 IEEE International Conference on Industrial Informatics, INDIN 2015*, no. 1, pp. 314–321, 2015.
- [VFFU17] Vogel-Heuser, B., Fischer, J., Feldmann, S., Ulewicz, S. and Rösch, S., "Modularity and architecture of PLC-based software for automated production Systems: An analysis in industrial companies," In: *Journal of Systems and Software*, vol. 131, no. 1, pp. 35–62, 2017.
- [VFND18] Vogel-Heuser, B., Fischer, J., Neumann, E. and Diehm, S., "Key maturity indicators for module libraries for PLC-based control software in the domain of automated Production Systems," In: *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1610–1617, 2018.
- [VFRF15] Vogel-Heuser, B., Fischer, J., Rosch, S., Feldmann, S. and Ulewicz, S., "Challenges for maintenance of PLC-software and its related hardware for automated production systems: Selected industrial Case Studies," In: *Proceedings - IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 362–371, 2015.
- [VFST15] Vogel-Heuser, B., Fay, A., Schaefer, I., Tichy, M., Schäfer, I., Tichy, M., Schaefer, I., Tichy, M. et al., "Evolution of software in automated production systems:

- Challenges and research directions," In: *Journal of Systems and Software (JSS)*, vol. 110, Elsevier Ltd., pp. 54–84, 2015.
- [VHCR17] Vogel-Heuser, B., Heinrich, R., Cha, S., Rostami, K., Ocker, F., Koch, S., Reussner, R. and Ziegltrum, S., "Maintenance effort estimation with KAMP4aPS for cross-disciplinary automated PLC-based Production Systems - a collaborative approach," In: *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4360–4367, 2017.
- [VLFF14] Vogel-Heuser, B., Legat, C., Folmer, J. and Feldmann, S., 2014, "Researching Evolution in Industrial Plant Automation: Scenarios and Documentation of the Pick and Place Unit," [Online] Available: <https://mediatum.ub.tum.de/doc/1208973>.
- [VMKL15] Vogel-Heuser, B., Mund, J., Kowal, M., Legat, C., Folmer, J., Teufl, S. and Schaefer, I., "Towards interdisciplinary variability modeling for automated production systems: Opportunities and challenges when applying delta modeling: A case study," In: *Proceedings - 13th IEEE International Conference on Industrial Informatics (INDIN)*, pp. 322–328, 2015.
- [VOBS18] Vogel-Heuser, B., Bougouffa, S. and Sollfrank, M., 2018, "Researching Evolution in Industrial Plant Automation: Scenarios and Documentation of the extended Pick and Place Unit," [Online] Available: <https://mediatum.ub.tum.de/1468863>.
- [VoCh20] Vogel-Heuser, B. and Cha, S., "Höhere Softwarequalität in Automatisierungsanwendungen (en: Higher software quality in automation applications)," [Online] Available: <https://www.embedded-software-engineering.de/hoehere-softwarequalitaet-in-automatisierungsanwendungen-a-904833/>, [Accessed: 2020-12-29].
- [Voge09] Vogel-Heuser, B., "Automation in the Wood and Paper Industry," In: *Springer Handbook of Automation.*, pp. 1015–1026, 2009.
- [Voge14] Vogel-Heuser, B., "Usability Experiments to Evaluate UML/SysML-Based Model Driven Software Engineering Notations for Logic Control in Manufacturing Automation," In: *Journal of Software Engineering and Applications*, vol. 07, no. 11, pp. 943–973, 2014.
- [VoOc18] Vogel-Heuser, B. and Ocker, F., "Maintainability and evolvability of control software in machine and plant manufacturing — An industrial survey," In: *Control Engineering Practice*, vol. 80, no. Sep., pp. 157–173, 2018.
- [VRFS16] Vogel-Heuser, B., Rösch, S., Fischer, J., Simon, T., Ulewicz, S. and Folmer, J., "Fault Handling in PLC-Based Industry 4.0 Automated Production Systems as a Basis for Restart and Self-Configuration and Its Evaluation," In: *Journal of Software Engineering and Applications*, vol. 9, no. 1, pp. 1–43, 2016.
- [Vyat13] Vyatkin, V., "Software Engineering in Industrial Automation: State-of-the-Art Review," In: *IEEE Transactions on Industrial Informatics*, vol. 9, no. 3, pp. 1234–1249, 2013.

- [VyBo08] Vyatkin, V. and Bouzon, G., "Using Visual Specifications in Verification of Industrial Automation Controllers," In: *EURASIP Journal on Embedded Systems*, vol. 2008, no. 1, pp. 1–9, 2008.
- [Weig19] Weigl, A., "geteta — Generalized Test Tables — VerifAPS 1.0 documentation," [Online] Available: <https://formal.iti.kit.edu/~weigl/verifaps/geteta/>, [Accessed: 2020-12-29].
- [Weig21] Weigl, A.S., *Formal Specification and Verification Methods for Automated Production System*. Dissertation, Karlsruher Institut für Technologie, Karlsruhe, Germany. Institut für Theoretische Informatik, 2021. (to be submitted).
- [WeZB15] Wenger, M., Zoitl, A. and Blech, J.O., "Behavioral type-based monitoring for IEC 61499," In: *Proceedings - 20th IEEE Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 1–8, 2015.
- [WiVo11] Witsch, D. and Vogel-Heuser, B., "PLC-Statecharts: An Approach to Integrate UML-Statecharts in Open-Loop Control Engineering – Aspects on Behavioral Semantics and Model-Checking," In: *IFAC Proceedings Volumes*, vol. 44, IFAC, no. 1, pp. 7866–7872, 2011.
- [WKMS19] Wang, R., Kristensen, L.M., Meling, H. and Stolz, V., "Automated test case generation for the Paxos single-decree protocol using a Coloured Petri Net model," In: *Journal of Logical and Algebraic Methods in Programming*, vol. 104, pp. 254–273, 2019.
- [WoRB03] Wong, N., Rindfleisch, A. and Burroughs, J.E., "Do Reverse-Worded Items Confound Measures in Cross-Cultural Consumer Research? The Case of the Material Values Scale," In: *Journal of Consumer Research*, vol. 30, no. 1, pp. 72–91, 2003.
- [WWUU17] Weigl, A., Wiebe, F., Ulbrich, M., Ulewicz, S., Cha, S., Kirsten, M., Beckert, B. and Vogel-Heuser, B., "Generalized test tables: A powerful and intuitive specification language for reactive systems," In: *Proceedings - 15th IEEE International Conference on Industrial Informatics (INDIN)*, pp. 875–882, 2017.
- [ZeVe07] Zeiss, B. and Vega, D., "Applying the ISO 9126 quality model to test specifications," In: *Software Engineering 2007*, vol. 105, pp. 231–244, 2007.
- [ZhZK07] Zheng, Y., Zhou, J. and Krause, P., "A Model Checking based Test Case Generation Framework for Web Services," In: *Fourth International Conference on Information Technology (ITNG'07)*, pp. 715–722, 2007.
- [ZJMC00] Zhinan Zhou, Ji Zhang, McKinley, P.K. and Cheng, B.H.C., "TA-LTL: Specifying Adaptation Timing Properties in Autonomic Systems," In: *Proceedings - 3rd IEEE International Workshop on Engineering of Autonomic & Autonomous Systems (EASE)*, pp. 109–118, .

-
- [ZLWG11] Zhang, P., Li, W., Wan, D. and Grunske, L., "Monitoring of Probabilistic Timed Property Sequence Charts," In: *Software: Practice and Experience*, vol. 41, no. 7, pp. 841–866, 2011.
- [ZSZL10] Zhang, P., Su, Z., Zhu, Y., Li, W. and Li, B., "WS-PSC Monitor: A Tool Chain for Monitoring Temporal and Timing Properties in Composite Service Based on Property Sequence Chart," In: *Proceedings - 1st International Conference on Runtime Verification (RV)*, pp. 485–489, 2010.

11. List of Figures

Figure 1:	Schematic representation of the thesis scope: complementing quality control driven by specification.....	3
Figure 2:	Technical process and technical system (reproduced from [VFRF15]).....	6
Figure 3:	Schematic view of a cyclic operation of PLCs (reproduced from [LaGö99])	7
Figure 4:	A step of SFC with entry and exit guard.....	9
Figure 5:	V-Modell XT with separation of the project-independent and project related activities (primarily presented in [VERE10], reproduced from the figure in [VFST15]).....	10
Figure 6:	Generic schematic view of the system verification [BaKa08] (reproduced).....	14
Figure 7:	Activities and users of the specification over the engineering process – depicting the information exchange (arrows).....	24
Figure 8:	Basic structure of the GTTs	45
Figure 9:	GTTs to check the initialization condition ($EN = TRUE$ and $I > 10$ for 2 seconds). (a) not correct with too relaxed condition at the beginning, (b) correct but requiring more rows, and (c) compact exclusion representation with progress flag	50
Figure 10:	GTTs to check the rising edge of X. (a) waiting for the input condition to be true and then output continues for 100 cycles by designating two different values of X in two rows, and (b) compact rising edge check expression with triggering symbol.....	50
Figure 11:	Conveyor separation module to sort different type of work pieces and the corresponding function block to control the separation process: (a) graphical overview of the machine, and (b) schematic view of the function block with interface indicated	51
Figure 12:	Separation behavior flow chart with the illustration of the machine and the work piece	51
Figure 13:	Test cases for (a) non-metal work piece, and (b) metal work piece	53
Figure 14:	Generation of a GTT.....	54
Figure 15:	Two-way quality assurance process – forward (F) and backward (B1 and B2).....	56
Figure 16:	Counterexample representation (i) timing contracting/expanding, and (ii) variable contracting/expanding	57

Figure 17:	Connecting the counterexample to the GTT	58
Figure 18:	Overview of the change verification procedure achieving change specification base on the existing code [CWUB18b].....	61
Figure 19:	Overview of monitoring function block generation and its usage [CUVW17]	62
Figure 20:	State diagram derived from the generalized test table with Sn relating to the currently active row in a generalized test table [CUVW17]	64
Figure 21:	Structure of the code block for cases 1 and 2 in IEC 61131-3 (ST)	65
Figure 22:	Embedding two-way quality assurance process using GTTs in the control software engineering process.....	66
Figure 23:	Prototypical toolchain of the formal verification using GTT (scope of the thesis is specified by the dashed line; the gray area is discussed in [Weig21]).....	70
Figure 24:	Add-on overview	70
Figure 25:	Editing GTT – (a) GTT editor plane, and (b) guided variable selection.....	71
Figure 26:	Displaying counterexample in case of verification fail	73
Figure 27:	Use cases mapped on the engineering processes	76
Figure 28:	Community demonstrator: extended Pick&Place Unit.....	77
Figure 29:	Single acting cylinder: (a) extension and (b) retraction.....	78
Figure 30:	Single acting cylinder behavior description considering the timing constraint and corresponding alarm signal: (a) normal extension behavior within 500 ms, (b) abnormal extension behavior with the piston not reaching the expended position within 500 ms, (c) normal retraction behavior within 200 ms, and (d) abnormal retraction behavior with the piston not reaching the retracted position within 200 ms	79
Figure 31:	Sorting module: Black WPs are sorted in Ramp1 and the others in Ramp2.....	80
Figure 32:	Excerpt of GTT for the sorting module behavior	80
Figure 33:	Sorting module (evolved from Figure 33: Black WPs are sorted in Ramp1 and the others in Ramp2 [CWUB18b]	81
Figure 34:	Part of SFC for sorting behavior of conveyor (Sc10d) with a specific behavior branch high-lighted with a dashed line [CWUB18b]	82

Figure 35: The GTT for the revised functionality achieved by editing the automatically generated one. Revised part is highlighted in dark gray [CWUB18b]	83
Figure 36: Double acting cylinder: (a) extension and (b) retraction	83
Figure 37: Extension behavior description with the resetting function (a) extending after resetting and (b) retracting.....	84
Figure 38: A cylinder driver function block and its monitoring function block [CUVW17] (reproduced).....	85
Figure 39: An excerpt of the automatically generated monitoring block for the cylinder behaviors in Figure 38	86
Figure 40: Profiles of the participants: (a)-(c) for Exp-1 and (d) for Exp-3 [CVWU21].....	94
Figure 41: Exercise score comparison of Exp-1 (normalized mean value with error bars of the standard error): (a) – task1 (understanding), (b) – task2 (understanding, scaled), and (c) task3 (creating) [CVWU21].....	95
Figure 42: Correlation (based on Exp-1) of (a) score vs. grade, (b) score vs. internship experience, and (c) score vs. working experience [CVWU21]	96
Figure 43: Score comparison of GTT and PN from Exp-2(a) Understanding task, (b) Creating task, and (c) Correlation of score to the total score [CVWU21]	97
Figure 44: Correlation of the GTT exercise score and the overall grade (Bachelor course) from Exp-3 [CVWU21].....	99
Figure 45: Qualitative evaluation result in Exp-3. (a) Understanding, (b) Creating, and (c) overall. Reproduced from [CVWU21]	100

12. List of Tables

Table 1:	SFC action qualifiers	9
Table 2:	Summary of the requirements	26
Table 3:	Formal specification approaches with respect to the requirements (See Table 2 for each requirement description).....	32
Table 4:	Possible cell values and notations	47
Table 5:	Possible durations and notations	48
Table 6:	SFC action qualifiers and its converting pattern into GTT	59
Table 7:	Requirements and the corresponding evaluation.....	75
Table 8:	C/E type PN notations and comparable elements in GTTs	88
Table 9:	Hypotheses and proving methods	89
Table 10:	Summary of the tasks performed in the experiments	93
Table 11:	Task and specification complexity comparison in Exp-1*	96
Table 12:	Summary of the hypotheses evaluation.....	102
Table 13:	Summary of the expert evaluation comments relating to the requirements	104
Table 14:	Summary of the evaluation of the presented approach with respect to the requirements	107

Appendix A. Materials of experiment 1 for Chapter 7

Appendix A.1 Demographic data and prior knowledge questionnaire

14 May 2019

Questionnaire 1

Participant ID

1. Age Less than 24 years Between 24 and 30 years (inclusive) More than 30 years

2. Gender ()

3. Overall grade ()

4. Hochschulsemester ()

* number of semesters from the bachelor degree including this semester

5. Tick the lectures that you have taken

- Grundlagen der modernen Informationstechnik I (Principles of Modern Information Technology I)
- Grundlagen der modernen Informationstechnik II (Principles of Modern Information Technology II)
- Automatisierungstechnik 1 (Automation 1)
- Automatisierungstechnik 2 (Automation 2)
- Industrielle Softwareentwicklung für Ingenieure I (Industrial software engineering for engineers I)
- Entwicklung intelligenter verteilter eingebetteter Systeme in der Mechatronik (EiveSiM)

6. In which program are they enrolled?

- Mechanical Engineering – Master of Science
- Mechanical Engineering and Management – Master of Science
- Mechanical Engineering and Information – Master of Science
- Else (Please specify the program:)

7. You got the bachelor degree...

- at TUM
- in the other university in Germany
- in the other university not in Germany

in the following major (you can tick multiple)

- Mechanical engineering / Mechatronics
- Electrical / Electronics engineering
- Computer science
- Others ()

8. Do you have working experience in engineering field? For how long? Please tick on the relevant item and specify the duration.

Field	HiWi/WiHi	Internship in a company	Parttime/fulltime in a company
Mechanical engineering / Mechatronics	<input type="checkbox"/> ()	<input type="checkbox"/> ()	<input type="checkbox"/> ()
Electrical / Electronics engineering	<input type="checkbox"/> ()	<input type="checkbox"/> ()	<input type="checkbox"/> ()
Computer science	<input type="checkbox"/> ()	<input type="checkbox"/> ()	<input type="checkbox"/> ()
Others ()	<input type="checkbox"/> ()	<input type="checkbox"/> ()	<input type="checkbox"/> ()
No experience	<input type="checkbox"/>		

9. Do you have any (pure) software development project experience? (you can tick multiple)

!! there is another item (no. 11) regarding the embedded system. Please separate them from this item.

Project information	From lecture (practice block)	From HiWi/WiHi job	From industry work
How many projects have you worked on?	()	()	()
For the biggest project you had	Briefly describe the purpose of the software.		
	How many team members?		
	How long did the project take? e.g. 3 weeks, 2 months...		
	Which programming language was used?		
Other experiences e.g. competitions			

10. How do you assess your programming skills?

Response in the Scale 1 (No experience) to 5 (Advanced).

- Scale 1 (No experience)
- Scale 2 (Beginner)
- Scale 3 (Intermediate)
- Scale 4 (Good)
- Scale 5 (Advanced)

Appendix A.2 Evaluation tasks with possible answers

14 May 2019

Group 1 - Evaluation 1

The main goal of this Generalized Test Tables evaluation session is to perform an evaluation of a new specification language for a reactive system function block by users not familiarized with it, in order to assess the understandability and usability. This evaluation will be performed through the development of a simple case studies of some components. The results gathered will be used to assess the language usability, to decide the level of the further lecture and exam, as well as its future enhancements.

Each task consists of the task description and the given material. The task will be explained by the master of the session but you may read the text for your own understanding.

Participant ID	
-----------------------	--

Task – A: Spot the errorExplanation of the task

In this task, you will compare a function block in IEC61131-3 code with the specification in Generalized Test Tables (GTT). After reading the function block description and the specification,

Please

- (1) Complete the timing diagram depending on the button clicking signals.
- (2) Tick the lines of the code where you think the code is implemented incorrectly.
- (3) Describe a reason and a possible change.

For this task, 20 min will be given.

Given material

- Informal description of the function block behavior

A drilling machine changes its mode by clicking two buttons (Button 1 and Button 2).
A function block *FB_ModeSelect* results in the selected mode decided by its logic.

INPUT

B1(BOOL): Button 1 click status → when it is clicked, the value is TRUE

B2(BOOL): Button 2 click status → when it is clicked, the value is TRUE

OUTPUT

S (INT): 0 (mode 0), 1 (mode 1), and 2 (mode 2)

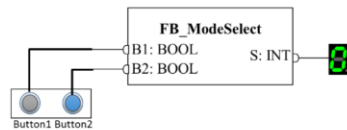


Figure 1. Function block *FB_ModeSelect*

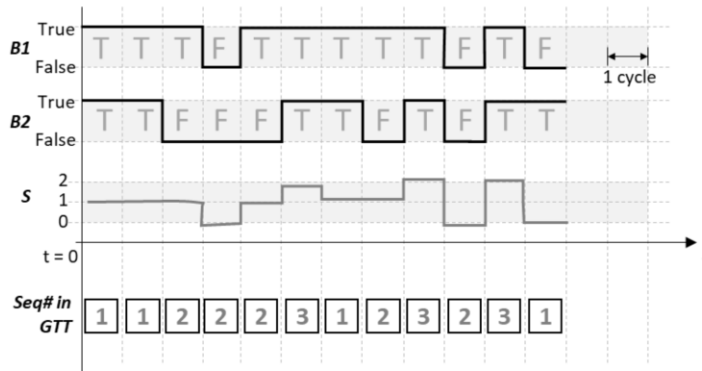
The following GTT specification describes the behavior of the function block *FB_ModeSelect* (Fig. 1) formally.

- Formal specification in GTT

Seq.	Input		Output	Duration
	B1	B2	S	
1	-	TRUE	$= SEL(B1,1,0)$	-
2	-	FALSE	$= SEL(B1,1,0)$	$[1, \infty]$
3	-	TRUE	2	1

* $SEL(x,a,b)$: binary selection – if x is true then it returns a , otherwise it returns b .

- Timing diagram
 - ⇒ (1) Complete the timing diagram for output signal S depending on the button clicking signals and fill in the corresponding row number (Seq #) of the GTT.



Time when you finish this task: _____

- Implemented code of a function block in Structured Text
 - ⇒ (2) Tick the lines of the code where the code is incorrect

<input type="checkbox"/>	01	FUNCTION_BLOCK FB_ButtonIndicator
<input type="checkbox"/>	02	VAR_INPUT
<input type="checkbox"/>	03	B1 : BOOL; (* Button 1 *)
<input type="checkbox"/>	04	B2 : BOOL; (* Button 2 *)
<input type="checkbox"/>	05	END_VAR
<input type="checkbox"/>	06	VAR_OUTPUT
<input type="checkbox"/>	07	S: INT;
<input type="checkbox"/>	08	END_VAR
<input type="checkbox"/>	09	VAR
<input type="checkbox"/>	10	B2_clicked : R_TRIG; (* Rising edge if Button 2 is pressed *)
<input type="checkbox"/>	11	END_VAR
<input type="checkbox"/>	12	B2_clicked(CLK:= B2);
<input checked="" type="checkbox"/>	13	IF B1 THEN
<input type="checkbox"/>	14	S:= 1;
		ELSE NOT B1
<input checked="" type="checkbox"/>	15	ELSIF B2_clicked.Q THEN
<input type="checkbox"/>	16	S:= 2;
<input checked="" type="checkbox"/>	17	END_IF
<input type="checkbox"/>	18	END_FUNCTION_BLOCK

* R_TRIG: the variable.Q gets TRUE once a rising edge is detected, or FALSE otherwise.

- ⇒ (3) Describe a reason and a possible change by answering following questions.

Describe

a) *Why is it wrong?*

rising edge of B2 is prior than B1 and
S := 0 needs to be handled

b) *How can it be corrected? (Natural language description or implementation)*

```
IF B2_clicked.Q THEN
  S := 2;
ELSIF B1 THEN
  S := 1;
ELSE
  S := 0;
END_IF
```



Time when you finish this task: _____

Task B: Spot the errorExplanation of the task

In this task, you will need to compare a function block in IEC61131-3 code with the specification in Petri Net and mark in the code where the violation appears.

Please

- (1) Figure out *Ready* signal characteristics after reading the specification.
- (2) Tick the lines of the code where you think the code is implemented incorrectly
- (3) Describe a reason and a possible change.

For this task, 20 min will be given.

Given material

- Informal description of the function block behavior.

Considering a fluid control system of Fig. 2, a function block *FB_Filling* (Fig. 3) is required to control the filling of the tanks.

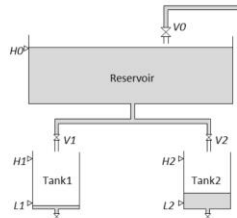


Figure 2. Schematics of fluid control system

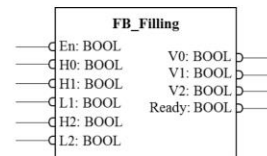
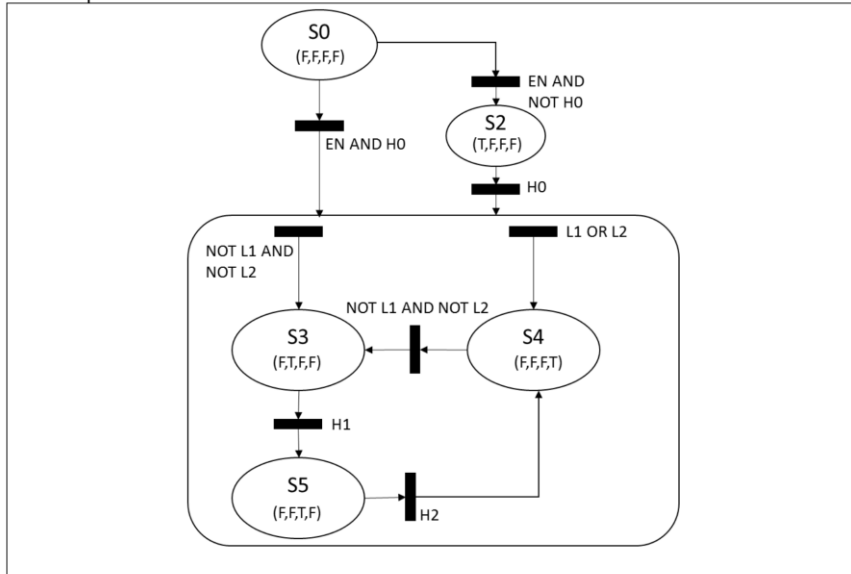


Figure 3. Function block *FB_Filling*

The following PN specification describes the behavior of the function block *FB_Filling* (Fig. 3) formally.

- Formal specification in a Petri Net



- After reading and understanding given materials..
 - ⇒ (1) Figure out **Ready** signal characteristics after reading the specification by answering following questions.

a) What is the condition for **Ready** to become TRUE from FALSE?

NOT L1 AND NOT L2

b) What is the condition for **Ready** to become FALSE from TRUE?

[Initially H0 is true AND (L1 OR L2)] OR [H1 AND H2]



Time when you finish this task: _____

- Implemented code of a function block in Structured Text
 - ⇒ (2) Tick the lines of the code where the code is incorrect

```

01 FUNCTION_BLOCK FB_Filling
02 VAR_INPUT
03   En: BOOL; (* function block enable *)
04   H0: BOOL; (* max level of tank0 *)
05   H1: BOOL; (* max level of tank1 *)
06   L1: BOOL; (* min level of tank1 *)
07   H2: BOOL; (* max level of tank2 *)
08   L2: BOOL; (* min level of tank2 *)
09 END_VAR
10 VAR_OUTPUT
11   V0: BOOL; (* valve open/close for tank0 *)
12   V1: BOOL; (* valve open/close for tank1 *)
13   V2: BOOL; (* valve open/close for tank2 *)
14   Ready: BOOL := TRUE;
15 END_VAR
16 VAR
17   init: BOOL := FALSE; (* initialization indicator *)
18   oldEn: BOOL := FALSE; (* to check enable trigger *)
19 END_VAR
20 IF En AND NOT oldEn THEN
21   Ready:= FALSE;
22   IF NOT H0 THEN
23     init := FALSE;
24     V0 := TRUE;
25   END_IF
26 END_IF
27 IF En AND H0 THEN
28   V0 := FALSE;
29   Ready:= FALSE;
30   init := TRUE;
31 END_IF
32 IF En AND init THEN
33   IF (NOT L1 AND NOT L2) THEN
34     Ready := FALSE;
35   ELSE
36     Ready := TRUE;
37   END_IF
38   IF NOT Ready THEN
x 39     IF NOT H2 THEN
40       V2 := TRUE;
41     ELSE
42       V2 := FALSE;
x 43     IF NOT H1 THEN
44       V1 := TRUE;
45     ELSE
46       V1 := FALSE;
47     Ready := TRUE;
48   END_IF
49 END_IF
50 END_IF
51 END_IF
52 oldEn := En;
53 END_FUNCTION_BLOCK

```

⇒ (3) Describe a reason and a possible change by answering following questions.

a) Why is it wrong?

Level check signal H1 and H2 are swapped in the code

b) How can it be corrected? (Natural language description or implementation)

```
IF NOT Ready THEN
  IF NOT H2 THEN
    V1 := TRUE;
  ELSE
    V1 := FALSE;
  IF NOT H1 THEN
    V2 := TRUE;
  ELSE
    V2 := FALSE;
  Ready := TRUE;
END_IF
END_IF
```



Time when you finish this task: _____

Task 1 – C: Generate the specification

Explanation of the task

In this task, you will need to read a description of a function block together with the implemented code in IEC61131-3 (ST) and generate the specification.

Please generate the specification in (1) GTT and (2) Petri Net. Time limit is 15 min for each.

For this task, 30 min will be given.

Given material

- Natural language description of the function block

A sorting module (Fig. 4) and the software function block (Fig. 5) is to assort different work pieces (WPs) into the different conveyors. There are three different WPs types: metal, black-plastic, white-plastic. At the sensor zone, there are an inductive sensor (*IND*), and an optical sensor (*OPT*). The arrival of the WP is indicated by two presence sensor P1 and P2 indicates the sensor zone. WPs are sorted depending on the type. Metal work, white, and black WPs are delivered to the conveyor 1, 2, and 3 respectively.

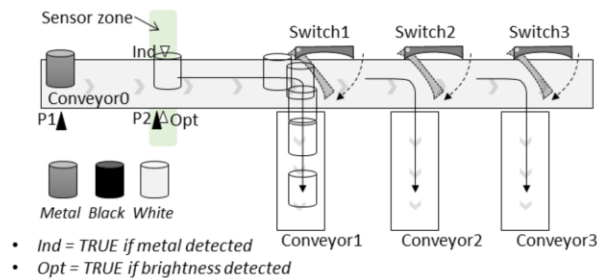


Figure 4. Schematics of sorting module

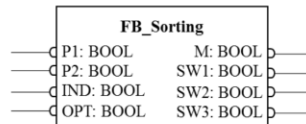


Figure 5. Function block *FB_Sorting*

- Code

```

01 FUNCTION_BLOCK FB_Sorting
02 VAR_INPUT
03     P1: BOOL;
04     P2: BOOL;
05     IND: BOOL;
06     OPT: BOOL;
07 END_VAR
08 VAR_OUTPUT
09     M : BOOL; (* activate/deactivate the conveyor0*)
10     SW1: BOOL; (* close/open the switch1*)
11     SW2: BOOL; (* close/open the switch2*)
12     SW3: BOOL; (* close/open the switch3*)
13 END_VAR
14 VAR
15     PROCESSING: BOOL := FALSE;
16     TYPEDECIDED: BOOL := FALSE;
17     COUNTER: INT := 0;
18     internalSW1: BOOL := FALSE;
19     internalSW2: BOOL := FALSE;
20     internalSW3: BOOL := FALSE;
21 END_VAR
22 IF P1 AND NOT PROCESSING THEN
23     PROCESSING:= TRUE;
24     M := TRUE;
26 END_IF
28 IF P2 AND NOT TYPEDECIDED THEN
29     M := FALSE;
30     internalSW1 := IND;
31     internalSW2 := NOT IND AND OPT;
32     internalSW3 := NOT IND AND NOT OPT;
36     COUNTER++;
37     IF COUNTER = 10 THEN
38         SW1 := internalSW1;
39         SW2 := internalSW2;
40         SW3 := internalSW3;
41         COUNTER := 0;
42         M := TRUE;
43         TYPEDECIDED := TRUE;
44     END_IF
45 END_IF
47 IF TYPEDECIDED THEN
48     TIMER(IN:= TRUE, PT:=T#5000MS);
49     IF TIMER.Q THEN
50         M := FALSE;
51         SW1 := FALSE;
52         SW2 := FALSE;
53         SW3 := FALSE;
54         PROCESSING := FALSE;
55         TYPEDECIDED := FALSE;
56     END_IF
57 END_IF

```


58 | END_FUNCTION_BLOCK

(1) Describe the specification in GTT

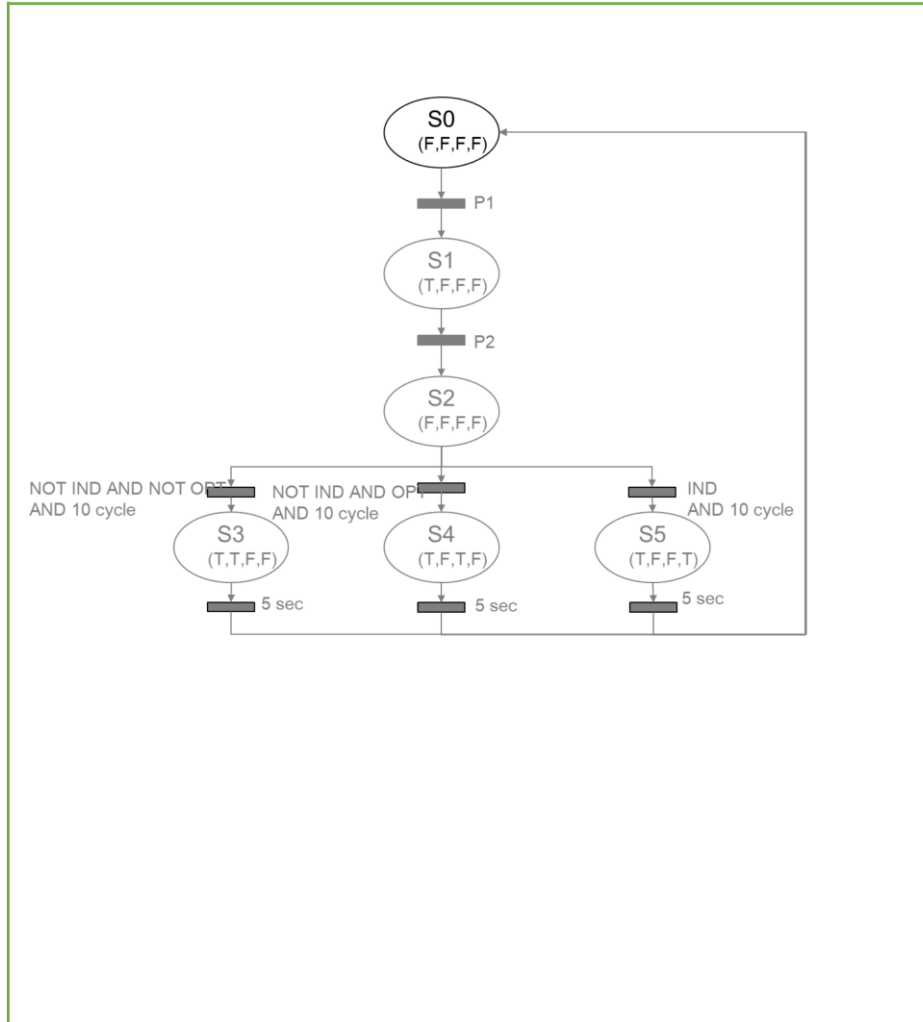
* You may use the template below

Seq.	Input				Output				Duration
	P1	P2	Ind	Opt	M	Sw1	Sw2	Sw3	
1	F	F	F	F	F	F	F	F	—
2	T	F	—	—	T	F	F	F	1
3	-	F	—	—	T	F	F	F	—
4	F	T	a	b	F	F	F	F	10
5	F	—	—	—	T	a	\bar{a} AND b	\bar{a} AND \bar{b}	5 sec
6	F	—	—	—	F	F	F	F	[1,∞]



Time when you finish this task: _____

(2) Describe the specification in Petri Net



Time when you finish this task: _____

Thank you for your participation and patience!

Appendix A.3 Given handout to be used during evaluation

Lehrstuhl AIS
Maschinenwesen

Notations for generalized test tables

GTT

Sequence
(row numbering)
Input section **(A)**
Output section **(A)**
Duration section **(B)**
Block repetition **(C)**

#	Input			OUTPUT			DURATION
	I_1	I_2	I_3	X	Y	Z	
1	A2 1	1	2	0	0	A4 $\neq 0$	B1 1
2	A1 —	—	—	—	A3 [1,2]	—	B5 — _p
3	0	=k	=k	A6 =2*k	A5 =X	A7 =Z[-1]	B2 [5,7]
4	—	=k+1	—	[0,k]	>Y[-1]	2*Z>Y	B2 [1,∞]

(A) Input/Output cells:

	Cell value	Description	Examples
1	—	Don't care	—
2	Specific values	Exact values (within the data type, or ENUM values)	1, 0, TRUE, GREEN
3	Intervals	Range of the values	[1,5], [7,∞]
4	Constraints	Value restriction	≥ 5 , $\neq 7$
5	References	Referring the other cell with a column name or a symbol	= I_3 , = O_5 , =a
6	ST-expression	Expressions allowed in ST language	= $I_3 * O_5$, = $\text{MIN}(I_1, I_2)$
7	Back references	Referring the same cell or the other cell with timing index	= $I_3[-1]$, = $O_3[-2]$,
8	Boolean constraints	Conditions to be true in the row	$I_2 > I_3$ AND $I_3 \geq 5$

(B) Duration cells:

	Cell value	Description	Examples
1	Specific values	Exact values for cycles (default) or time (with unit)	1, 10, 7 sec
2	Intervals	Range of the timing duration	[1,5], [7sec,∞]
3	—	Means [0,∞] which allows row to be skipped or repeated arbitrary times.	—
4	— _∞	If a program arrives at that row, it should stay .	— _∞
5	— _p	"Repeat until the next row input is satisfied" Usage: if a program repeats behavior waiting for a certain condition	— _p

(C) Block repetition (nestable):

	Value	Description	Examples
-	Grouping symbol with annotation as (B)	↕ symbol defines a group for some repetition Number of repetition is given by (B) (Without any annotation means same as — _∞ by default.)	$\text{↕} = \text{↕}_{-\infty}$, $\text{↕}_{[1,3]}$

Notations for generalized test tables

GTT

Be careful!

' $X > 3$ ' vs. ' $= X > 3$ '

O_1
$O_1 > 3$
$= I_2 > 3$

- .. Means ' $O_1 > 3$ ', i.e. this is a cell constraints.
- .. Means ' $O_1 = I_2 > 3$ ', i.e. the cell will have TRUE or FALSE depending on the evaluation of " $I_2 > 3$ ".

"—" vs. "¬p"

- "—" : the system may remain in the row as long as it satisfies the table
- "¬p" : the system may remain in the row as long as it satisfies the table AND as long as the new row is not satisfies.

#	Input			OUTPUT			DURATION
	I_1	...	I_m	O_1	...	O_n	
1							
2							
3							∞

- .. Means 'repeat the block'
- Usually the last row for arbitrary time with block repetition like $[1, \infty]$
- .. Means 'stay in this row!' → contradict!

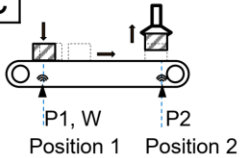
Hints to know

- > You can access standard FUNCTIONS. e.g. =SEL(W<5,10,20)
- > You can **not** access variables from the future. e.g. X[+1] is **not** allowed
- > **No** variable is allowed in Duration column
- > Look at **concrete** runs and try to generalize them
- > Read/write tables to catch/describe
 - Which **state** of the system each row represents (by seeing each row),
 - How the behaviors and states are **ordered** (by seeing over the rows), and
 - How values are **related** to each other (by seeing referencing)

Example

#	Input			OUTPUT			DURATION
	W	P1	P2	M	Mval	G	
1	0	FALSE	FALSE	FALSE	0	FALSE	—
2	[0,10]	TRUE	FALSE	TRUE	=SEL(W<5,20,10)	FALSE	1
3	—	—	FALSE	TRUE	=Mval[-1]	FALSE	[0, 7sec]
4	0	FALSE	TRUE	FALSE	0	TRUE	—
5	0	FALSE	FALSE	FALSE	0	FALSE	[1, ∞]

<5 kg → speed 20
 ≥ 5kg → speed 10
 (max 10kg)



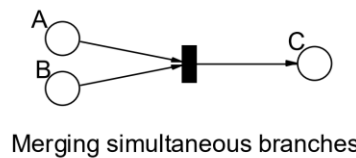
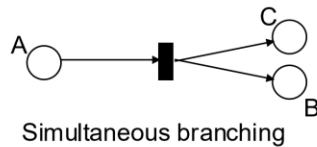
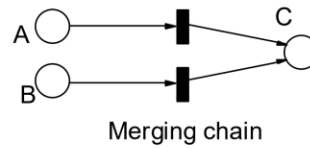
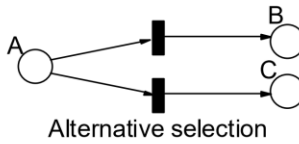
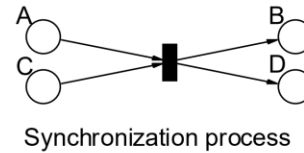
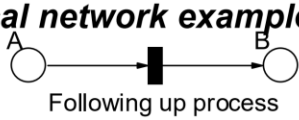
Notations for Petri Nets

Petri Net

Graphical symbol	Description	Interpretation
	Place (state)	Situation, status, or job processing
	Transition (event)	State transition, condition
	Edge	Connection between state and transition
	Initial transition (event)	Initial transition with omitting the common initial place
	Token	Marking active state

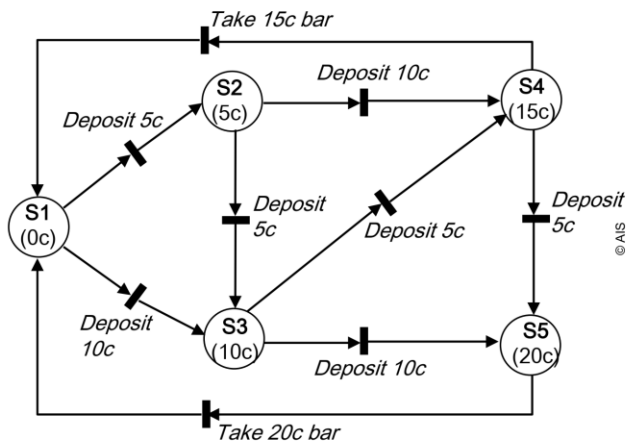
*Token can be omitted for the answers in this session (unmarked PN)

Typical network examples



Example

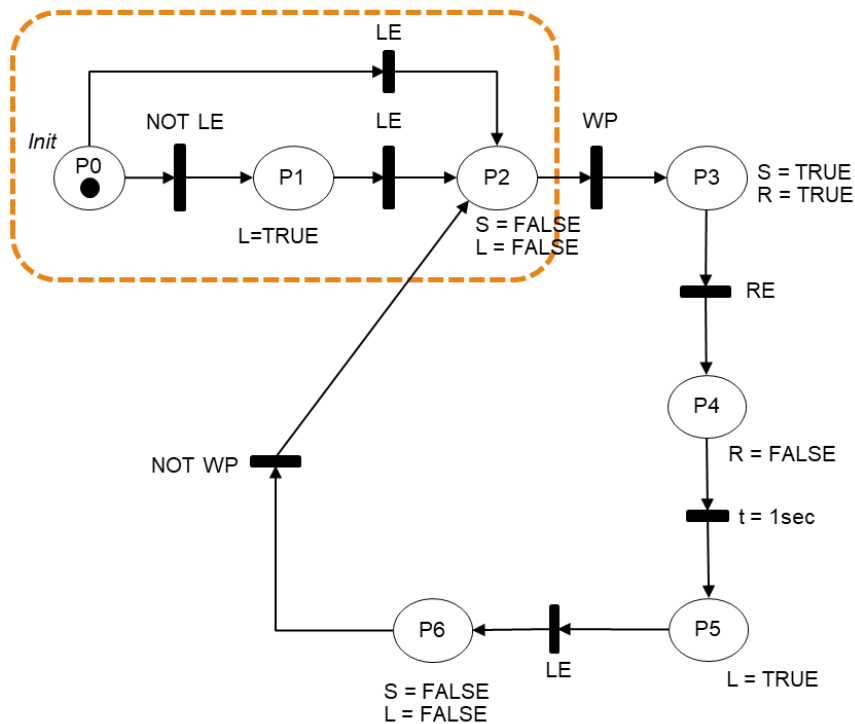
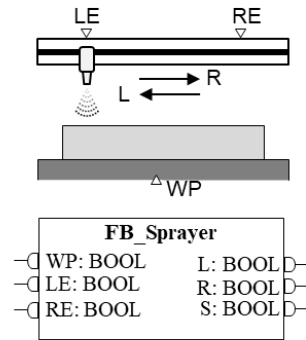
- The vending machine dispenses two kinds of snack bars: 20c bar or 15c bar.
- It takes two types of coins: 10c or 5c, and shows the current deposit.
- It does not return any change.



Appendix B. Materials of experiment 2 for Chapter 7

d) Below you will find a brief description of a moving color spray module and a corresponding Petri Net (conditional event net, initial place is P0) which describes its behavior of movement and spray. After reading these descriptions, answer the questions (i ~ v).

- A work piece arriving for being sprayed is indicated using a Boolean input variable WP .
- The nozzle head position is indicated using Boolean input variables LE and RE for left-end and right-end position respectively.
- The nozzle head movement is controlled by two Boolean output variables L and R to move left and right respectively.
- Spraying is controlled by a Boolean output variable S to enable or disable it.
- It is assumed that the processed WP is collected automatically after spraying is done.
- In the Petri net, all output variables are initialized as FALSE at the beginning and t indicates the elapsed time in a place.



- i. Describe the initial behavior (dashed area) of movement and spraying if the nozzle head is not at LE position at the beginning of the function block execution. Use the input/output signal variables to make it clear.

The nozzle head will be moved to the left end (LE) with the spraying off and stop at LE.

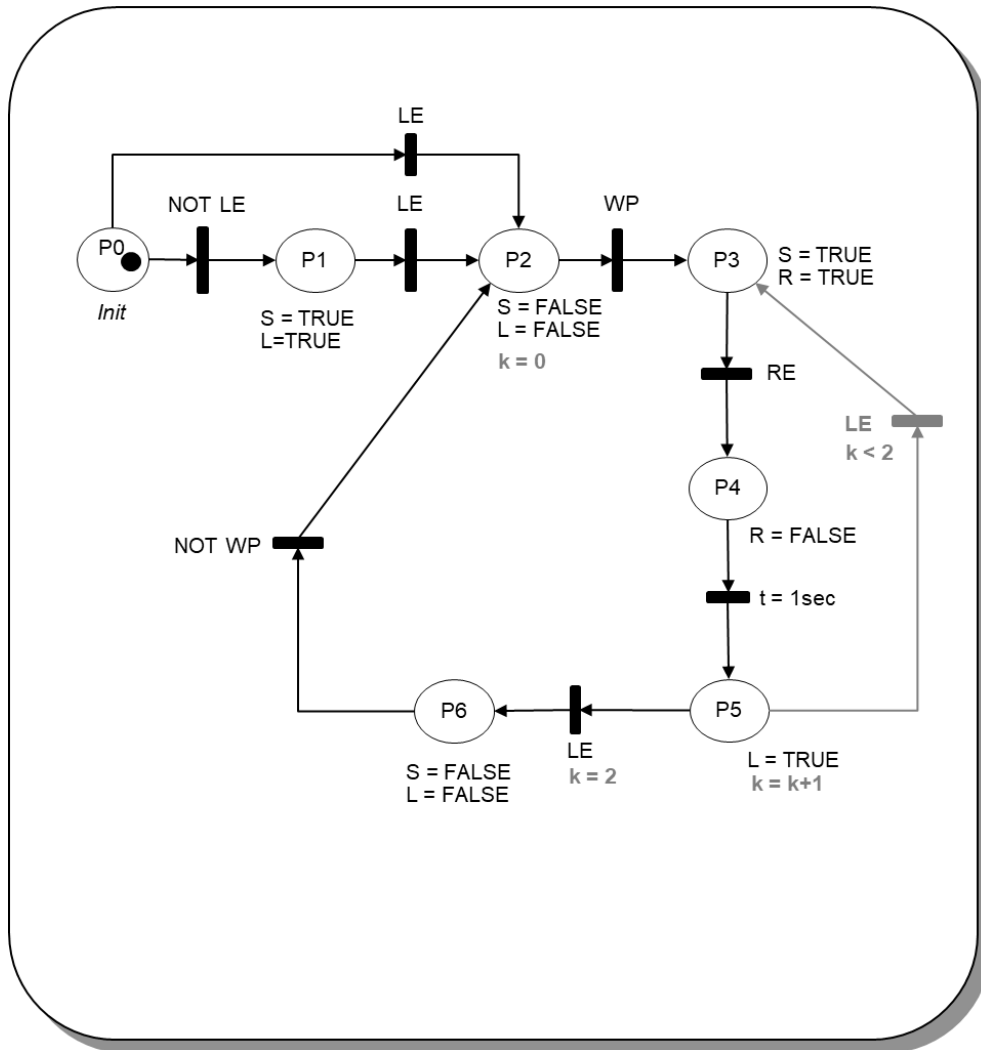
- ii. Describe the behavior when a WP is provided (WP = TRUE) until it is removed (WP = FALSE). Each step should be specified regarding its movement and spraying with correct time duration or the condition to transfer to the next step.

1. Spray ON / moving R / until it gets to the right end (RE).
2. At the right end, Spray (keep) ON / stop moving / for 1 sec
3. And then Spray keep ON / moving L / until it gets to the left end with keep spraying on.
4. Once it arrives to the left end, spray OFF / stop moving / until WP is collected

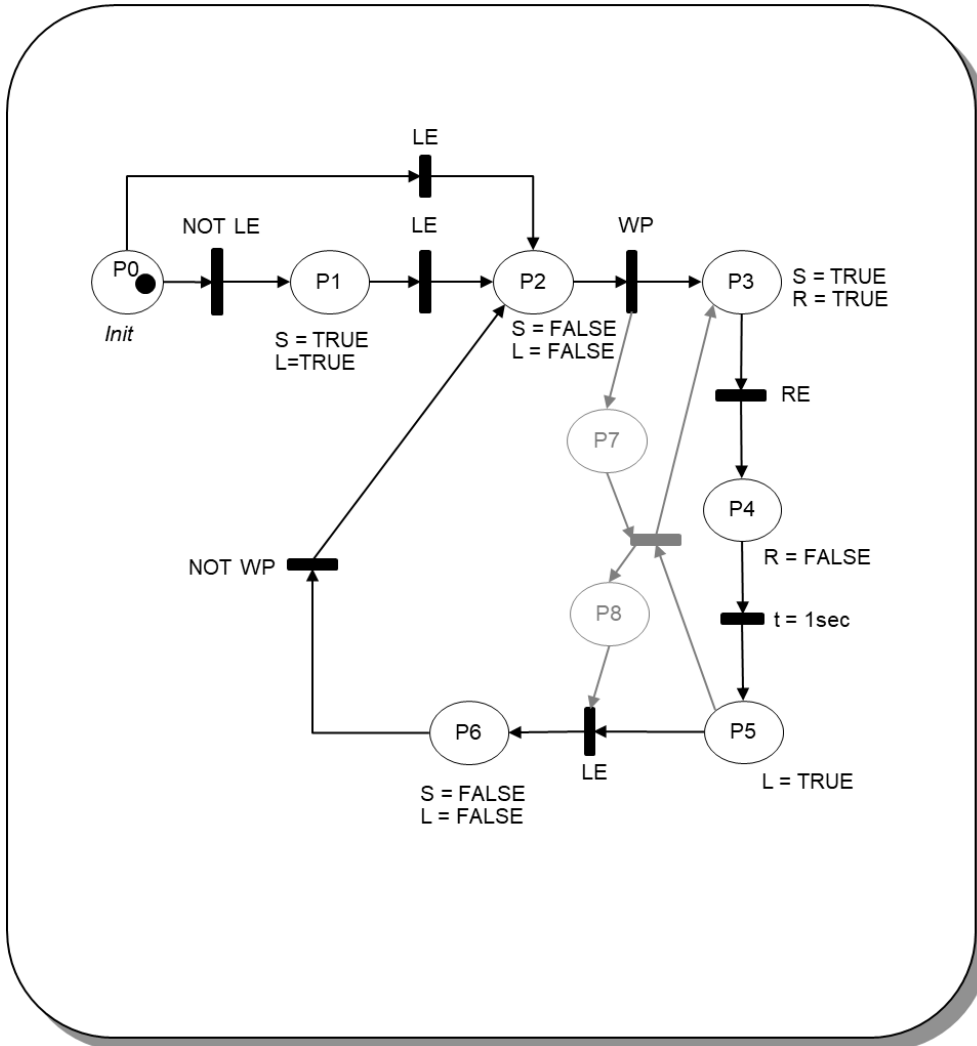
- iii. Regarding the signal pairs below, specify the condition or place ID for each pair where the given signals are true at the same time.

- | | |
|-------------------------|------------------|
| • L and R | It never happens |
| • L and S | P1,P5 |
| • R and S | P3 |

- iv. Now a new requirement is added to the specification: the coloring process for each WP should be done twice (double round-trips of the nozzle). Modify the original Petri Net below (identical to the one on page 11) by adding an internal variable k for counting. Adding new place is not allowed.

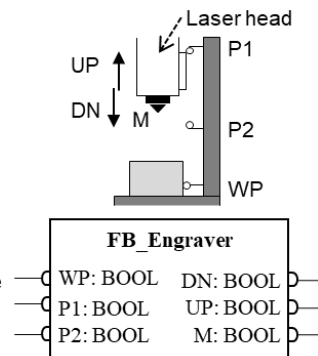


- v. For the new requirement in the previous question (iv, doubled coloring process), modify the original Petri Net below (identical to the one on page 11) without introducing new variable. Adding new place or transition is allowed if necessary.



e) Below you will find a brief description of a laser engraver function block and a corresponding generalized test table which describes its behavior (movement and engraving). After reading these descriptions, answer the questions (i ~ iv).

- A work piece provided to the engraver is indicated using a Boolean input variable *WP*.
- The upper position and lower position of the laser head are indicated using Boolean input variables *P1* and *P2* respectively.
- The vertical movement is controlled by two Boolean output variables *UP* and *DOWN*.
- Engraving (laser on and off) is controlled by a Boolean output variable *M*.
- It is assumed that the engraved WP is collected after the engraving is done.



Seq.	Input			Output			Duration
	<i>WP</i>	<i>P1</i>	<i>P2</i>	<i>DN</i>	<i>UP</i>	<i>M</i>	
1	—	FALSE	—	FALSE	TRUE	FALSE	—
2	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	—
3	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	1
4	TRUE	—	—	TRUE	FALSE	FALSE	$\neg p$
5	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	2 sec
6	TRUE	FALSE	—	FALSE	TRUE	FALSE	—
7	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	—

- i. Describe the initial behavior (row 1 and 2, dashed area) regarding the movement and engraving if the laser head is not at P1 at the beginning of the function block execution. Use the input/output signal variables to make it clear.

The Laser head will be moved up to P1 position with engraver deactivated (laser off) and stop at P1.

- ii. Describe the behavior of the engraver block when a WP is provided (WP = TRUE) until it is collected (WP = FALSE). Each step should be specified regarding vertical movement and laser activation with correct time duration or the condition to transfer to the next step.

1. Laser OFF / moving DN / until it gets to P2.
2. At P2, Laser ON / stop moving / for 2 sec
3. And then Laser OFF/ moving UP / until it gets to P1
4. Laser OFF / stop moving / until WP is collected

- iii. Regarding the signal pairs below, specify the condition or sequence ID (row number) for each pair where the given signals are true at the same time.

- **DN** and **UP** It never happens
- **DN** and **M** It never happens
- **UP** and **M** It never happens

- iv. Now a new requirement is added to the specification: Each WP must be engraved twice. Add block repetition element on the generalized test table below (identical to the page 15) implementing this requirement.
**It is assumed that the WP will be collected after being engraved twice.*

Seq.	Input			Output			Duration
	WP	P1	P2	DN	UP	M	
1	—	FALSE	—	FALSE	TRUE	FALSE	—
2	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	—
3	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	1
4	TRUE	—	—	TRUE	FALSE	FALSE	— _p
5	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	2 sec
6	TRUE	FALSE	—	FALSE	TRUE	FALSE	—
7	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	—

Appendix C. Materials of experiment 3 for Chapter 7

15 Jan 2020 – Automatisierungstechnik

ID

The main goal of this exercise is to practice a specification language, which is generalized test tables (GTTs).

Each task will be required to write down the beginning and ending time.

Basic information

1. Hochschulsemester

a. 6th b. > 6th c. < 6th

2. Age

3. Gender

a. female b. male c. others

4. Average score so far

a. 1.0 – 1.70 b. 1.71– 2.30 c. 2.31 – 3.00 d. 3.01 –

5. Have you had any previous experience of working with software or hardware specification other than lectures?

a. YES b. NO

6. Have you attended the lecture for Testing? (last week, 8 Jan)

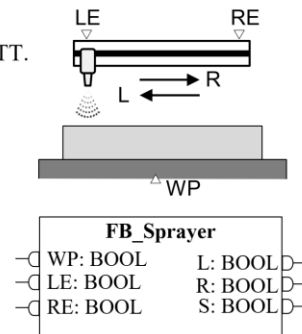
a. YES b. NO

Task 1.

You will find the specification of the **Moving color spray module** in GTT.

Basic information

- A work piece arrival is indicated by the sensor **WP**.
- The nozzle head reaching at left/right end is indicated by sensor **LE/RE** respectively.
- The nozzle head moves along the moving rail to the left or right by activating the function block output **L** and **R**.
- Spraying is controlled by the output **S** to enable (TRUE) or disable (FALSE).



Seq.	INPUT			OUTPUT			Duration
	WP	LE	RE	S	L	R	
1	FALSE	FALSE	—	FALSE	TRUE	FALSE	—
2	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	> 0 sec
3	TRUE	✓ TRUE	FALSE	TRUE	FALSE	TRUE	—
4	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	1 sec
5	TRUE	FALSE	—	TRUE	TRUE	FALSE	—
6	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	—

At what time do you begin?

:

Task 1 – 1. Answer if the statement is **True** or **False**. Also, indicate the sequence number (row number of the table) to support your decision.

	True	False	Seq.#
1. As an initialization behavior (which is not repeated), if the nozzle head is not at LE position (i.e. LE is FALSE), it is supposed to move left (L to be TRUE).	<input checked="" type="checkbox"/>	<input type="checkbox"/>	1
2. This GTT describes the situation that there is a WP already during the initialization behavior (which is not repeated).	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1
3. After initialized, the module waits for a work piece for a specific time.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	2
4. When WP is detected, the nozzle head moves to the right .	<input checked="" type="checkbox"/>	<input type="checkbox"/>	3
5. Spray is enabled for the first time when the nozzle head arrives at RE position.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	3
6. When the nozzle head arrives at RE position, the nozzle head immediately stops for 1 sec with spray on .	<input checked="" type="checkbox"/>	<input type="checkbox"/>	4
7. After stopping at RE position for 1 sec, the nozzle head moves to the left with the spray off .	<input type="checkbox"/>	<input checked="" type="checkbox"/>	5

Task 1 – 2. Answer if the statement is **True** or **False** and specify the sequence number (row number of the table) if it is True.

	True	False	Seq.#
1. There is a moment that both L and R gets to be TRUE at the same time.	<input type="checkbox"/>	<input checked="" type="checkbox"/>	None
2. There is a moment that both S and L gets to be TRUE at the same time.	<input checked="" type="checkbox"/>	<input type="checkbox"/>	5
3. There is a moment that both S and R gets to be TRUE at the same time.	<input checked="" type="checkbox"/>	<input type="checkbox"/>	3

At what time do you finish this page?

:

Task 1 – 3. Now, the behavior has to be changed: the spraying process for each WP should be done twice (double round-trips of the nozzle). Modify the original specification in GTT below **by adding block repetition symbol**.

**It should keep spraying when it starts second round.*

Seq.	INPUT			OUTPUT			Duration
	WP	LE	RE	S	L	R	
1	FALSE	FALSE	–	FALSE	TRUE	FALSE	–
2	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	> 0 sec
3	TRUE	√TRUE	FALSE	TRUE	FALSE	TRUE	–
4	TRUE	FALSE	TRUE	TRUE	FALSE	FALSE	1 sec
5	TRUE	FALSE	–	TRUE	TRUE	FALSE	–
6	TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	–

At what time do you finish this page?

:

Opinions

Interpreting GTTs (Task 1)

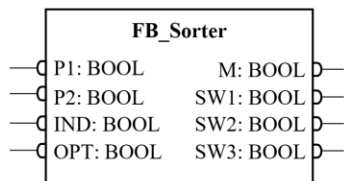
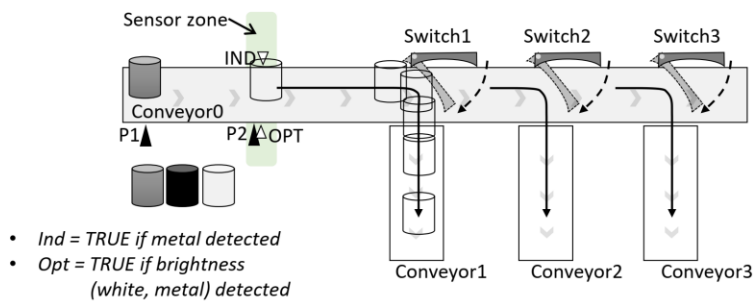
	Strongly disagree	Disagree	Uncertain	Agree	Strongly agree
1. I can understand the GTT concept.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2. Comprehending a GTT in Task1 was easy.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3. Remembering the syntax of the language was difficult.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4. I assume I could correctly interpret GTTs.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Task 2.

A sorting system is to assort different work pieces into the different conveyors.

Basic information

- There are three different work piece (WP) types: metal, black-plastic, white-plastic. They all have same size and shape.
- Conveying moves one direction (towards right) by enabling the motor (**M**).
- At the sensor zone, there are an inductive sensor (**IND**) for metal detection, and an optical sensor (**OPT**) for brightness detection which is the case of metal and white-plastic WP.
- The arrival of a WP is indicated by **P1**. Present sensor **P2** indicates the sensor zone.
- Metal, White, and Black WP is supposed to be sorted in Conveyor 1, 2 and 3 respectively.



At what time do you begin?

:

Task 2-1. Answer the questions.

- When a WP arrives at the sensor zone, sensors detect the type of the WP.

Complete the truth table (Fig. 2-1) of the sensor values (TRUE or FALSE) in the given table.

Type	IND	OPT
Metal	TRUE	TRUE/ don't care
White	FALSE	TRUE
Black	FALSE	FALSE

Fig. 2-1: A truth table of IND and OPT

- Based on the truth table, each switch signals can be described by IND and OPT sensor values.

Fill in the blanks in Fig. 2-2 with Boolean expressions using x, y as IND, OPT value respectively and logic operators (AND, OR, NOT).

Type	Boolean expression
Switch1	x
Switch2	$\text{NOT } x \text{ AND } y$
Switch3	$(\text{NOT } x \text{ AND}) \text{ NOT } y$

Fig. 2-2: Logical expressions of the switch signal

Task 2-2. Complete the GTT (Fig. 2-3) by following questions.

- When there is no WP at the beginning, the system stays inactive. This state continues for an arbitrary time. **Fill in the blanks of Seq1 row.**
- When a WP arrives at P1, the conveyor will start to move without activating three switches. **Fill in the blanks as the condition of Seq2 row. Be careful of the sensor which might change the value during the execution. (Hint: triggering symbol)**
- The conveying stops for 200 ms when the WP is at P2. **Fill in the blanks of Seq3.**
- For the next 10 sec, the conveyor will activate to sort out the WP by turning on the conveyor with activating the corresponding switch. **Fill in the blanks of Seq4 row using the Boolean expressions in Fig.2-3.**
- After sorting, the conveyor system will be empty and repeat the behavior from the beginning. **Describe the repeated behavior by using block repetition symbol.**

Seq.	INPUT				OUTPUT				Duration
	P1	P2	IND	OPT	M	Sw1	Sw2	Sw3	
1	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	—
2	√TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	—
3	FALSE	TRUE	x	y	FALSE	FALSE	FALSE	FALSE	200 ms
4	FALSE	√TRUE	—	—	TRUE	x	$\text{NOT } x \text{ AND } y$	$(\text{NOT } x \text{ AND}) \text{ NOT } y$	10 sec

Fig. 2-3: A generalized test table of the conveyor system

At what time do you finish this page?

:

Opinions

Creating GTTs (Task 2)

	Strongly disagree	Disagree	Uncertain	Agree	Strongly agree
1. I can apply the GTT concept.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2. Developing GTTs in Task2 was easy.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3. I did not understand parts of Task2 <i>(if No.3 answer is Uncertain/Agree/Strongly agree)</i> Where and what could you not understand in Task2?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
<div style="border: 1px solid gray; height: 40px; width: 100%;"></div>					
4. I assume I could correctly create GTTs.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

General impressions

	Strongly disagree	Disagree	Uncertain	Agree	Strongly agree
1. In general, GTT is easy to use. <i>(if No.2 answer is Uncertain/Disagree/Strongly disagree)</i>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
a. I think that I would need significant support of a technical person to be able to use GTTs.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2. I would imagine that most application developers can learn GTT quickly (within one day or less) as much as being required in the exercise.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3. I found the described specification in GTT is efficient (compactness).	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4. I am generally satisfied with GTTs.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5. I found GTT unnecessarily complex.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6. Once I work in industry, I am likely to use GTT for the requirement specification frequently.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Any feedback

