Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Practice-Oriented Formal Methods to Support the Software Development of Industrial Control Systems

Ph.D. Thesis Booklet

Dániel Darvas

Thesis supervisor:
István Majzik, Ph.D. (BUTE)

Advisor:
Enrique Blanco Viñuela, Ph.D. (CERN)

Budapest
2017

1 Preliminaries and Objectives

Dependability[1] is an integrating concept comprising availability, reliability, safety, integrity and maintainability. This is a desired property, especially for critical systems. A *failure* is an observable deviation from the system's required behaviour, thus a threat to dependability. The cause of a failure is the propagation of an *error*, which itself is a certain (internal) system state that can result in a failure. The causes of the errors are the *faults* [Avi+04].

There are various means to attain the attributes of dependability: *fault prevention*, *fault tolerance*, *fault removal* and *fault forecasting* [Avi+04]. Formal methods are well-known techniques for the prevention of *development faults* and some of the *operational faults*, by providing mathematically sound, unambiguous means for the description and verification of the system's requirements [Mar94]. Formal verification and formal specification are getting more and more used in *safety-critical* application domains where the consequences of a failure are *catastrophic* [Avi+04; Woo+09]. This can be either because a single failure may cause an accident or loss of life, or it implies a high economic loss (e.g. in avionics [Sou+09], railway systems [LSP07], space applications [Hav+00]), or the undesired consequence affects a large number of systems (e.g. mass-produced processors [Fix08; Kai+09]) causing a high total cost.

Industrial control systems are used in various settings. If the functionality of a control system is safety-critical, the IEC 61508-2 standard [I61508-2] defines required development and verification methods for each *safety integrity level* (SIL), depending on the probability of tolerable hazards (hazardous failures). The industrial control systems consist of many components. Often a key element is a *programmable logic controller* (PLC): a robust, reconfigurable, specialised computer that performs the control tasks. In certain cases, their operation is safety-critical, but many times – due to additional safety-related systems or measures (e.g. physical barriers, independent safety relays) – the target SIL for the PLC-based controller is below SIL 1, the lowest SIL defined in IEC 61508. Even though in these cases the expected failure is rare or not catastrophic, this does not mean that a failure (e.g. an *outage*) cannot cause significant economic losses. However, the lower SIL typically manifests in reduced verification budget. In these cases the use of *heavyweight* formal methods (e.g. B Method or Z for specification; or manual use of theorem provers for verification) would need excessive effort. Besides the difficult usage (high training costs, need for special expertise), another common obstacle to using formal verification methods is their performance. For example, exhaustively checking the behaviour (state space) of a model is a computationally difficult task, therefore most of the algorithms cannot scale up to the size of industrial problems.

Goal. The main goal of this research is to analyse the applicability of formal methods in the domain of industrial control systems and to propose specification and verification methods. As mentioned above, the two main challenges of using these methods are *performance* and *usability*. This dissertation proposes various solutions to both challenges. It aims to provide more efficient verification algorithms and easy-to-use, practice-oriented formal (specification and verification) methods that can be applied to PLC software used in industrial control systems, where the use of heavyweight methods is not necessary or not feasible. The methods to be proposed should take the particularities of the target domain into account.

Structure. This thesis booklet is structured as follows. The rest of Section 1 briefly introduces the concepts and techniques of formal verification (Section 1.1) and formal specification (Section 1.2). The section is concluded in Section 1.3 by summarising the challenges targeted in this work. Next, Section 2 discusses the research methodology and the new results of this dissertation.

[1] In the dissertation the taxonomy proposed by Avižienis, Laprie *et al.* [Avi+04] is used, which is briefly introduced and summarised in the following two paragraphs.

After, Section 3 overviews the application of the new results. Finally, Section 4 lists the author's related publications.

1.1 Introduction to Formal Verification

Verification is "[t]he process of evaluating a system or component to determine whether the products [...] satisfy the conditions imposed at the start" [I1012]. *Formal verification* techniques are mathematically sound methods to precisely determine the satisfaction of the given formalised requirements.

"*Model checking* is an automated [formal verification] technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model" [BK08]. As a more precise definition, model checking is a method to find all states in a model (given as Kripke structure) which satisfy a given temporal logic formula [Cla08]. This method was first described by Edmund M. Clarke and E. Allen Emerson in [CE82], also by Jean-Pierre Queille and Joseph Sifakis independently [QS82].

Being an automated method, it is a good candidate for lightweight formal verification, it has a potential to be a "push-button technology" whose usage does not require high degree of user interaction or expertise [BK08]. Furthermore, it can provide diagnostic traces (counterexamples or witnesses) that is useful feedback about the problems found.

Modelling and requirement description formalisms. While model checking operates on a Kripke structure according to its definition, typically higher-level modelling languages are used for representing the models, such as Petri nets [Mur89], (timed or untimed) automata [AD94], or process algebra [Fok00]. Similarly, multiple different temporal logic can be used to describe the requirements to check. The most frequently used ones are linear temporal logic (LTL) and computation tree logic (CTL). The different formalisms have different expressivity, also the model checking algorithms can differ significantly depending on the supported formalisms.

Model checking methods. E. Clarke, one of the creators of model checking said that the model checking algorithm is an "intelligent exhaustive search of the state space to determine if the specification is true or not" [CES09]. With the increase of the number of reachable states in a system, the "intelligence" of the state space exploration algorithms is getting more and more important. The so-called *explicit* methods represent each state of the given model individually. This allows to use simple algorithms from graph theory, but fails to provide solution for large models where the individual handling of each state is not possible. Large state sets (state spaces) may be caused by various reasons, e.g. the large number of inputs and outputs or concurrent behaviours. This is the well-known *state space explosion* problem. Over the years various solutions emerged to handle this issue:

- *Symbolic algorithms*, which store the state space in a more compact, encoded format, e.g. using decision diagrams;
- *Abstractions*, which simplify the model to have a smaller state space;
- *Bounded algorithms*, which limit the depth of the state space exploration (from the initial state) to reduce the size of the explored state space.

The first symbolic algorithms were based on binary decision diagrams [Bur+92]. Since then new algorithms were developed, using different exploration strategies and data structures. One of the promising solutions is *saturation* [CLS01], which "tends to perform extremely well when applied to discrete-event systems having multiple asynchronous events that depend and affect only relatively small subsystems" [CZJ12]. The fact that the algorithm "performs well" means that the set of verifiable models and requirements is larger or the execution time is shorter, but it does not mean that there are no limitations imposed by the performance needs of the algorithm. In certain cases excessive amount of memory is required to perform the verification.

There is no silver bullet for model checking, each method has its disadvantages and limitations. Some approaches tried already to combine ideas from different methods, e.g. [Cha+02; Cop+01; CNQ05]. A possible improvement of the saturation-based model checking is to combine it with bounded model checking, which – to the author's best knowledge – was not studied before this research project. This could also help the verification of industrial control systems by improving the earlier verification performance (e.g. compared to [*c*28]). Evaluation of this possibility is a challenge of this dissertation (Challenge 1).

Model checking as a part of the PLC software development process. Model checking has already proven to be useful in various domains [Cla08]. However, providing the necessary formal models and the requirements as temporal logic formulae is a difficult task, especially for the people not familiar with formal methods. Furthermore, model checking may also need manual adaptation, fine-tuning to the current problem to improve the performance. This implies a high cost of usage, which may be an obstacle to apply model checking in the development of PLC programs.

The academic algorithm design and development efforts led to high-performance general-purpose model checkers (e.g. UPPAAL [Amn+01], LTSmin [Kan+15], NuSMV/nuXmv [Cav+14]). However, general-purpose tools cannot improve the domain-specific usability of the verification method. To improve the usability and to integrate formal verification into the industrial control system development processes, the focus should be specifically set to this domain.

Although the use of model checking for PLC-based industrial control software was already studied in e.g. [GSF08; SD08; BBK12], these works did not provide generic solutions applicable in real-life, or this aspect was not emphasised. Making model checking adapted to the PLC program development domain, usable directly by the PLC developers; and making it scalable are challenges of this dissertation (Challenges 2, 3). Furthermore, special attention should be paid to a special branch of PLCs, the so-called *fail-safe* or *safety PLCs*. To attain a high level of confidence during the software development process for such PLCs, special restrictions and development methods are followed (e.g. coding conventions, programming language restrictions), these have to be taken into account for the solution to be proposed (Challenge 4).

1.2 Introduction to Formal Specification

Requirements engineering is a set of activities to explore, evaluate and document the objectives, capabilities, constraints and assumptions of a system to be designed [Lam09]. The *(requirements) specification* is the act of "detailing, structuring and documenting the agreed characteristics of the system-to-be" [Lam09]. According to [Lam00], *formal specification* "is the expression, in some formal language and at some level of abstraction, of a collection of properties some system should satisfy." One of the facts making the formal specification process difficult is that "[s]pecifications are never formal in the first place" and they are "hard to develop and assess" [Lam00]. The expected benefits of formalising the specification method are "a higher degree of precision in the formulation [...], precise rules for their interpretation and much more sophisticated forms of validation and verification" [Lam09].

Formal specification is studied since the late 1960s, and since then several methods emerged. Petri nets [Mur89], Lotos [I8807], the B Method [Abr96], Z [I13568], or the communicating sequential processes (CSP) [Hoa85] are widely-known techniques. Though widely-known, they are not widely used in the industry [Kni+97], because they are too complex, they need too deep mathematical knowledge, or their abstraction level is too high. Therefore the usage of these specification methods is restricted to highly critical domains, e.g. avionics [HLR98] or railway industry [But02].

In the industrial control systems domain, the state-of-the-art development processes still rely on informal specifications and hidden assumptions. These specifications are often ambiguous,

leading to misunderstanding and unintended behaviours in the implementation. The lack of unambiguous specification imposes a problem for the formal verification too: how can we decide if the implementation is correct, if we do not know what is correctness, i.e. what are the expected properties?

There are various attempts to provide better, PLC-specific specification methods, e.g. [Lju+10; Luk+13]. Analysing the existing methods and finding a suitable specification is a challenge of this work (Challenge 5). It is another challenge to provide formal verification for PLC software on the basis of the selected specification method (Challenge 6).

1.3 Summary of New Challenges

Challenge 1: Designing model checking algorithms combining bounded and saturation-based techniques to improve their performance. Both bounded model checking and saturation-based techniques increase the set of models on which verification is feasible compared to basic explicit model checking algorithms. Is it possible to combine these two approaches? Does it improve the performance with respect to the original saturation-based model checking?

Challenge 2: Making model checking easily accessible to the PLC developers. Model checking is rarely used for industrial control software, mainly because of the enormous effort needed to create formal models and requirements, furthermore to learn the usage of the model checker tools. How can model checking be made accessible and practically applicable in the PLC program development process? How can model checking be used without excessive effort, without exposing the users (PLC developers) to complex mathematical formalisms?

Challenge 3: Making the PLC model checking applicable to real-world PLC programs. The formal models of real PLC modules or programs and their state spaces tend to be extremely large, making the model checking infeasible using general-purpose model checker tools. Could heuristic model reductions reduce the performance needs of model checking and therefore cope with a bigger set of models?

Challenge 4: Extending the model checking approach to safety-critical PLC programs. The original PLC model checking workflow supported the Siemens SCL language only, which – being a high-level language – is more suitable for the implementation of complex programs. However, the development of PLCs used in safety-critical settings has specific procedures and restrictions, such as the mandatory usage of FBD or LAD languages (in case of Siemens PLCs). How can model checking be adapted to these lower-level programming languages used in safety-critical PLC program development?

Challenge 5: Providing lightweight formal specification for PLC software modules. Unambiguous requirements are essential for any development or verification activity. Formal specifications may reduce the ambiguity, but the general-purpose formal specification methods are too complex and non-intuitive to be used in the PLC development domain with a reasonable effort. What are the requirements towards a formal specification language specially adapted to the PLC domain? What formal specification method can aid the PLC program development process?

Challenge 6: Providing verification solutions based on formal specification. How could formal specification improve the PLC program verification? What verification methods can be used to check the conformance between a PLC program and its formal specification? How can this be made useful in practice, without excessive amount of false positives (i.e. without having a high number of detected differences that are considered to be acceptable by the developers)?

These challenges led to a research project with new scientific results in three different areas: a new, saturation-based bounded model checking algorithm with different iteration strategies (Thesis 1), a new method to apply model checking for PLC programs (Thesis 2) and the definition of a formal specification language for PLC software modules together with its application methods (Thesis 3). Table 1 presents the correspondence between the discussed challenges and the results of this work.

Table 1. Correspondence between the discussed challenges and the proposed solutions

		Challenge					
		1	2	3	4	5	6
Thesis 1	*Chapter 2 of the dissertation*	•					
Thesis 2	*Chapter 3 of the dissertation*		•	•	•		
Thesis 3	*Chapter 4 of the dissertation*					•	•

2 Research Methods and New Results

Research Methods. J.N. Amaral [Ama] classifies the computing science research into five categories: *formal*, *experimental*, *build*, *process*, and *model* methodologies. This dissertation follows the experimental and build methodologies. In the build methodology, the researcher builds a novel artefact "to demonstrate that it is possible" [Ama]. In the experimental methodology, the researcher first identifies "the questions to be asked about the system under evaluation" [Ama]. The second phase is about answering these questions using measurements, experiments.

The design of new algorithms and the definition of new formalisms, languages rely on mathematical logic, graph theory and automata theory. The research presented in this dissertation extends and builds on various earlier methods and results. Modelling of systems in this dissertation relies on Petri nets, finite state machines and timed automata. The property specification methods are partially based on the CTL and LTL temporal logic languages. The verification algorithms use model checking, behavioural equivalence relations and graph algorithms.

The results were evaluated on various industrial examples. A significant part of the presented research was conducted in the Industrial Control and Engineering group of the Engineering Department, and in the Industrial Controls and Safety group of the Beams Department at CERN (European Organization for Nuclear Research). This allowed me to interact often with industrial and control engineers. This provided key knowledge about the particularities and the needs of the target domain, and later valuable feedback on my work.

2.1 Bounded Model Checking Based on Saturation

A successful subclass of symbolic model checking methods is the family of the so-called *saturation*-based techniques [CZJ12; ZC09]. The different saturation-based techniques have already proved their efficiency in various settings. A notable example is described in [*c*28] which provided complete model checking for the first time of the so-called *PRISE safety logic* that is an industrial control subsystem used in Paks Nuclear Power Plant, Hungary.

Saturation-based CTL model checking is a two-step algorithm: first it explores the reachable state space, then it checks the given requirement. However, in many cases the requirement could be evaluated (e.g. a state violating the requirement could be found) by exploring only a part of the state space (exploring only the states that are for at most b *distance* from the initial state,

i.e. that are reachable within at most b steps). This is the general idea of the bounded model checking algorithms. Our idea was that bounded model checking combined with saturation-based techniques may improve the performance of saturation-based model checking in certain cases.

It is relatively simple to construct an explicit bounded model checking algorithm, i.e. to limit for example a depth-first search of the state graph. However, saturation works on sets of states in an encoded representation and it improves the performance by using a special iteration order that benefits from the locality of the decomposed models. These imply that it is difficult to count the steps (transitions) made in the state space, also the state sets may contain states with different distance values.

Yu *et al.* [YCL09] proposed a method for bounded state space exploration based on saturation, but they did not consider this algorithm to use for model checking. Their method is not directly usable for model checking for two main reasons: the lack of CTL formula evaluation (i.e. they considered reachability problems only, not the evaluation of temporal logic expressions) and the lack of iterativeness (i.e. they did not consider that the state space exploration should be continued with increasing bounds until the requirement can be evaluated).

Under the supervision of A. Vörös and T. Bartha, I have constructed a workflow that integrates bounded saturation-based state space exploration and saturation-based CTL formula evaluation. In contrast to the simple, two-step process of unbounded saturation-based model checking, the workflow is iterative, as shown in Figure 1.

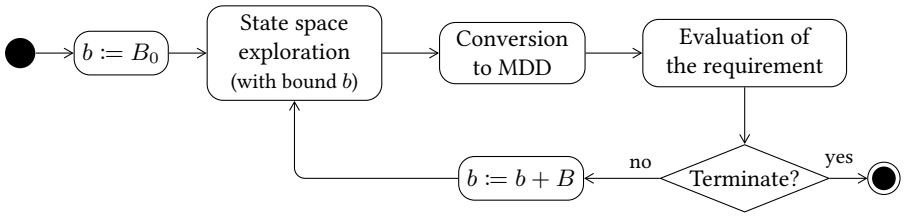


Figure 1. Overview of the iterative saturation-based bounded model checking (B-I-Sat)

The basic workflow integrates already existing building blocks in a novel way:

- The *bounded state space exploration* is based on the algorithm proposed by Yu *et al.* [YCL09] (B_0 denotes the initial bound value, B is the bound increment),
- The *evaluation of the temporal logic requirement* uses an algorithm similar to the one used in [*c*28; *j*26], based on [CS03; ZC09],
- The *conversion* between edge-valued decision diagrams (EDDs) and multivalued decision diagrams (MDDs) is constructed based on their definition. Note that EDDs are used for bounded state space exploration to store the distance information of the states, but this extra information is not needed for the evaluation of the requirement.

This workflow and its implementation showed the feasibility of the concept. However, there were certain challenges which necessitated new contributions to make the verification efficient and correct.

- **Iterations.** The algorithm of Yu *et al.* [YCL09] should be made iterative, so that the exploration can be continued until the satisfaction of the requirement can be evaluated. Different iteration strategies are possible. In this work I have proposed three different strategies: the restarting, continuing, and compacting strategies. They mainly differ in the amount and type of data kept between iterations and the initial state sets used in the iterations.
- **Termination.** A crucial part of the workflow is the *termination*. In order to provide a complete and correct solution, bounded model checking should only be terminated when an authoritative result can be given based on the partial state space, or if the whole state space is explored. The requirement evaluation can only give an answer regarding the explored

part of the state space. This result might not hold for the whole (partially unexplored) state space. In the frame of this work, I have provided termination conditions for the saturation-based bounded CTL model checking. This is based on three-valued logic, which allows the reasoning about uncertain results which may be caused by the partial state space exploration.

- **Distance information storage.** The caveat of bounded state space exploration is the need to store *distance information* attached to the states. This will cause a fragmentation of the symbolic state sets, leading to increased computational and storage needs. I have developed an advanced incremental search strategy (the so-called compacting strategy) to reduce the memory consumption (and therefore the execution time too) by storing distance information for the states only if it is necessary.

These considerations led to B-I-Sat (Bounded Iterative Saturation), a bounded iterative saturation-based CTL model checking algorithm. The B-I-Sat algorithm with different strategies was evaluated on various benchmark models. B-I-Sat was also applied to an industrial control system, previously verified using unbounded saturation-based algorithms.

Thesis 1 I designed *B-I-Sat* (Bounded Iterative Saturation), a novel computation tree logic (CTL) model checking algorithm, that efficiently combines bounded model checking with saturation-based techniques.

1.1 I defined the building blocks, and based on them the B-I-Sat algorithm to perform bounded CTL model checking using saturation-based techniques. I defined two strategies for B-I-Sat: the restarting and continuing strategies.

1.2 I defined termination conditions for the B-I-Sat algorithm using three-valued logic.

1.3 I developed an advanced incremental search strategy, the so-called compacting strategy to reduce the memory consumption of the B-I-Sat algorithm.

1.4 I evaluated the performance of the B-I-Sat algorithm with the different strategies on various benchmark models and an industrial example.

The importance of these results is that the B-I-Sat algorithm allows faster verification or the verification of greater models in certain cases than the unbounded saturation-based CTL model checking. For example, Figure 2(a) shows that the various B-I-Sat strategies scale better with the increasing model size than the unbounded algorithm. Figure 2(b) demonstrates that the various strategies of the B-I-Sat algorithm have a significant advantage when the requirement under evaluation is shallow (small i values), while the unbounded model checking provides better execution time if the requirement is deep, thus most of the state space needs to be explored for the evaluation.

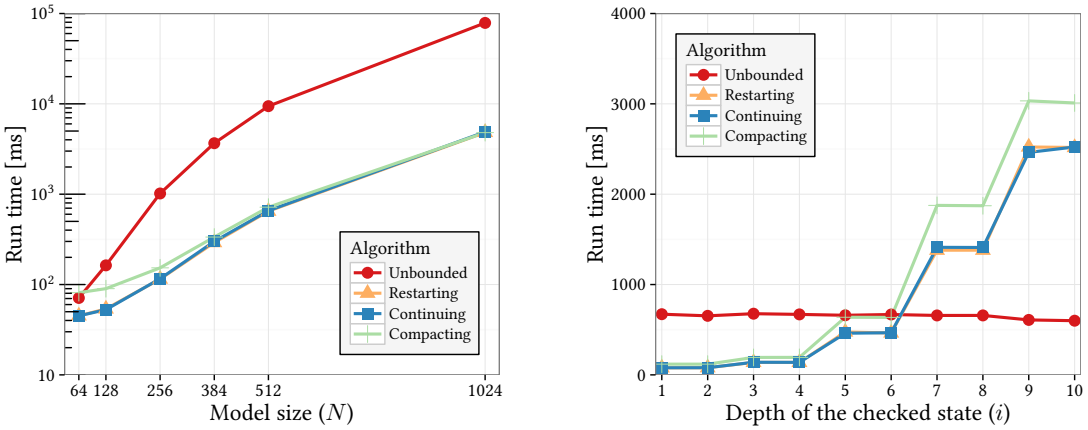
András Vörös and Tamás Bartha were taking part in this research as my B.Sc. and M.Sc. thesis supervisors. Parts of this work are presented in my B.Sc. and M.Sc. theses [*a*31; *a*30].

The results of Thesis 1 are presented in Chapter 2 of the dissertation. Related publications are the following: [*j*2; *j*4; *c*10; *c*17; *c*18; *e*20; *e*21].

2.2 Model Checking Critical PLC Programs

To introduce model checking to industrial development processes and widely use them (not only in extremely critical cases), manual formalisation of requirements or implementations is not a viable approach. To hide the formal requirements and formal models, automated methods are needed to generate these artefacts based on inputs that can be easily provided and understood by the targeted users. Furthermore, as the formal model should be hidden, the often needed manual optimisations should be automated as well.

In the frame of this work a generic model checking-based verification workflow was designed, specifically targeting the verification of PLC programs (see Figure 3). PLC programs come



(a) Execution time of evaluating EF ($bit_{12} = 1$) on the Counter–N models

(b) Execution time of evaluating EF ($q_i = 0$) on the Queen–10 model

Figure 2. Scalability comparison of the unbounded saturation and the B-I-Sat algorithm

in different languages: the IEC 61131 standard [I61131-3] defines five languages. Two of them, the high-level ST and the low-level IL are textual languages, while the FBD, LD and SFC languages are graphical. The different vendors implement these standard languages with minor or major differences. I have considered in this work the Siemens implementation of these languages (named SCL, STL, FBD, LAD and GRAPH/SFC, respectively). This wide variety of languages is not always accessible to the developers, for example the set of available languages and language features is restricted in the development of PLC software for safety-critical Siemens PLCs.

Different parts of the PLC software model checking were already addressed by different authors. Most of them target low-level languages (e.g. IL in [Can+00; LNN13]), but the high-level language ST is rarely supported [GSF08; BBK12]. Most works focus on the IEC 61131 standard version of the PLC languages, although the main PLC hardware vendors significantly deviate from the standards. Many PLC formal verification attempts use model checkers as underlying verification engines (e.g. CaSMV, NuSMV or UPPAAL [Can+00; GSF08; SF11]), while some others rely on SAT solvers (e.g. [LNN13]).

However, the main problem is that scalability and real-life usability are rarely targeted in the related work. Only a few authors reported explicitly about tools developed to support their ideas and algorithms, out of which only one, Arcade.PLC is available [BBK12]. Unfortunately, Arcade.PLC is not applicable in the considered application environment, partially because the users are exceedingly exposed to the formal details, for instance at the requirement specification.

Our workflow (depicted in Figure 3) specifically targets the scalability and usability challenges. The main steps and properties of the workflow are highlighted in the following.

- **Generic internal representation.** The workflow is built around an *intermediate model* (IM). This is a generic, simple representation of the formal model describing behaviour of the software. This description is independent from the verification engine used. The intermediate model is built on an extended automaton formalism that is appropriate to represent program behaviours and is close to many model checkers' input language.
- **Inputs of the workflow.** The inputs of the workflow are the PLC code (given in Siemens SCL language for non-safety and in STL for safety cases) and a requirement given as a concretised requirement pattern, similarly to [DAC99; CMS08]. A requirement pattern is a carefully-worded English sentence with placeholders that are filled by the user with Boolean expressions on input/output variables (signals). Then the temporal logic representation is automatically generated.
- **Reductions.** To make the model checking workflow less resource-consuming (thus

more feasible), automated property-preserving reductions are applied on the intermediate model. These comprise structural reductions, similar to optimisations performed by compilers. Certain reduction rules are taking the requirements into account and eliminate parts of the model that are not affecting the satisfaction of the given formula.

- **External verification.** To benefit from the state-of-the-art model checker developments and to optimise the development effort, no new model checker was developed in this project. Instead, widely-known general-purpose model checkers are wrapped and used as verification engines. The wrappers take care of the transformations between the intermediate representations and the model checker's (input and output) concrete syntax.
- **Reporting.** The different verification engines produce various, model checker-dependent, complex outputs. These are typically difficult to understand. Therefore these outputs are transformed into an intermediate representation. Based on this, a simplified verification report is generated that is more understandable for the PLC developers.

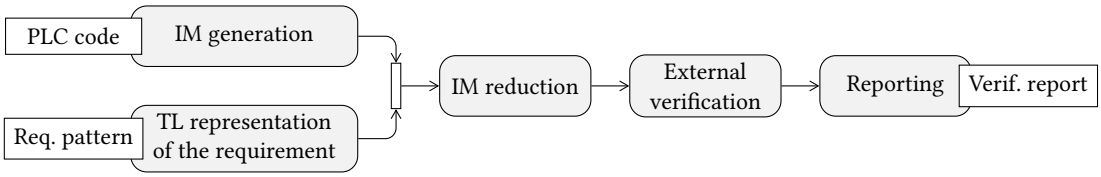


Figure 3. Simplified overview of the PLC program verification workflow

Thesis 2 I contributed to the development of a generic, flexible workflow to apply model checking to PLC programs without requiring extensive formal methods knowledge from the users. I designed essential parts of this workflow, as follows.

- 2.1 I designed an intermediate model (IM) language that can represent PLC programs and can act as a pivot language for different model checkers. The IM-based model checking is a fully automated method that can be used by developers who are not familiar with formal verification techniques.
- 2.2 I developed heuristics to automatically reduce the size of the intermediate models, making the model checking workflow less resource-demanding.
- 2.3 I extended this model checking workflow (originally developed only for SCL programming language) to support the PLC programming languages used in the development of safety PLCs: FBD and LAD, via the STL language.
- 2.4 I implemented the proposed model checking workflow in the *PLCverif* tool, providing push-button verification to the developers based on the source code of the PLC program and the pattern-based requirement specification. I evaluated the real-life applicability of this workflow using various PLC modules and applications developed and used at CERN.

The importance of these results is that the formal verification workflow implemented in the PLCverif tool allows the PLC developers to apply model checking to various PLC programs without requiring extensive formal verification knowledge. This makes possible to gain higher degree of confidence in the developed PLC programs. The PLCverif tool[2] is in use at CERN and its usage demonstrated that model checking can reveal issues in the specification and the implementation of PLC programs that testing could not [*j*3; *c*9; *c*16]. The proposed workflow was used among others in the development of a safety-critical system that is responsible for the safety of cryogenic magnet property measurements [*c*9]. 12 different problems were identified using model checking. Many of them were practically impossible to be found using the commonly used testing procedures.

[2] http://cern.ch/plcverif/

The work underlying Thesis 2 was a joint research with Borja Fernández Adiego from University of Oviedo and CERN. His main contributions were the high-level concept of the methodology (as described in earlier papers [Fer+13; FBM13]), the analysis of the PLC's behaviour, and the variable abstraction method as one of the IM reduction methods [*j*3]. His work resulted in a Ph.D. thesis [Fer14] at the University of Oviedo. My contribution is the detailed design and development of the intermediate model language, the design and development of the reduction methods, except the variable abstraction method. The transformation of the SCL programming language to the intermediate model language was a joint work. The extension and adaptation of this method to safety PLC programs is my contribution. Enrique Blanco Viñuela and Jean-Charles Tournier were involved in this research as advisors from CERN. Víctor M. González Suárez, Jan Olaf Blech, Simon Bliudze, András Vörös and Tamás Bartha were also providing helpful comments during this work.

The results of Thesis 2 are presented in Chapter 3 of the dissertation. Related publications are the following: [*j*1; *j*3; *c*8; *c*9; *c*11; *c*13; *c*14; *c*15; *c*16; *r*23; *r*24].

2.3 Formal Specification for PLC Modules

The PLCverif tool and the underlying workflow made formal verification accessible to the PLC developers without requiring special formal methods knowledge or alteration of the development process. However, it turned out that it is difficult to formalise the requirements based on the available informal specifications and documentations.

The pattern-based requirement specification used in the PLCverif tool and its workflow provides a limited solution for the formalisation challenge, as providing pattern-based requirements covering the complete set of behaviours is difficult. This is a significant problem especially in two cases: in the development of basic building blocks (reusable modules) of PLC programs, where a bug or controversial behaviour may impact a large number of applications; and in case of safety-critical systems. In these cases a set of declarative requirement patterns (that focuses on a limited set of typical properties) is not sufficient, as a complete behavioural specification is needed.

Providing well-defined specification languages for industrial control systems is an ongoing work for a long time. One of the most known specification languages is Grafcet [I60848], having its roots in mid-1970s [Bla77]. This is intended to be a precise specification language based on safe Petri nets, however it is not generally applicable to every kind of PLC programs, furthermore the description of complex logic tends to be ambiguous and error-prone. More recent approaches propose simpler solutions, such as ProcGraph [Luk+13], built on finite state machines. However, this often needs significant amount of source code written in the specification. Other solutions approach the problem from the world of formal languages: ST-LTL [Lju+10] is an adaptation of the LTL temporal logic to PLC programs. This method is formal and unambiguous, but it is more suitable to describe certain invariant properties than complete specifications of systems with significant complexity. More formal and non-formal, PLC-specific specification approaches exist, however I did not find any that could be considered formal, unambiguous and understandable enough for our usage at CERN.

This led to the development of PLCspecif, a domain-specific formal specification language for complete behaviour specification of PLC modules. After analysing the available solutions, the codebase at CERN and discussing with the developers, I have collected a set of general and domain-specific requirements regarding the definition of the language. General requirements towards PLCspecif are to make it formal, precise, lightweight (easy to learn). Moreover, it has to support understandability, verification and documentation. Domain-specific requirements are the support for events and low-level signals; to keep the logic description clean by separating the input/output-handling and the core logic; to support multiple formalisms in order to minimise the semantic gap between the specification formalism and the module to be specified; and the time handling should be adapted to the PLC timer semantics [*e*19].

Following the requirements, I have designed PLCspecif to be a hierarchical, modular language. Each module is either a composite module, refined by submodules; an alternative module, where the behaviour depends on a certain condition; or a leaf module, describing a part of the system's behaviour (see Figure 4 for a high-level overview of the module structure). The leaf modules are divided into three main parts: input handling, core logic description, and output handling. For each part, multiple formalisms are supported, thus the developer can choose the most suitable. The input/output handling statements can be described using simple Boolean or arithmetic expressions, or using one of the two defined tabular description formalisms. For the core logic description, a variant of state machines, a data-flow description and PLC timer descriptions can be used. These formalisms were selected based on the informal and semi-formal methods known and used by the PLC developers. Each module can be extended with invariant properties to support requirements that are difficult to be captured using the rather imperative core logic description methods.

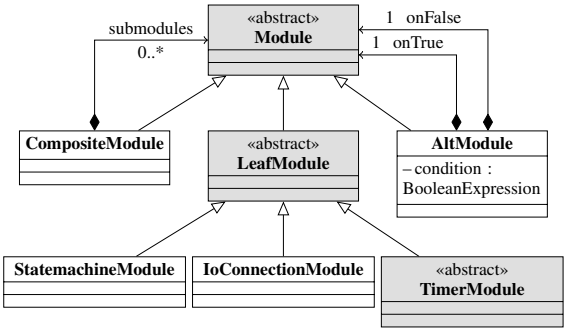


Figure 4. Metamodel of the PLCspecif module structure. [*e*19]

The abstract and concrete syntax were precisely defined for PLCspecif [*r*22], as well as the formal semantics, based on an extended automaton formalism.

The fact that PLCspecif has a precise, formal semantics ensures an unambiguous behaviour description, and it also allows us to use it for verification and code generation. In particular, I elaborated the following methods:

- **Static analysis.** To check the well-formedness of the specification, I have defined a set of rules. Most of them are implemented using graph algorithms. For some others I have provided translations into SAT problems and thus they can be verified using the Microsoft Z3 SAT/SMT solver [MB08].
- **Verifying invariant properties.** When a PLC developer cannot be sure that the imperative behaviour description satisfies some invariant properties, it is possible to include these properties explicitly in the specification. The previously discussed pattern-based requirement specification is used for the definition of invariant properties. Based on the semantics definition of PLCspecif, I gave a translation to PLCverif's intermediate model formalism that makes the verification of the specified invariant properties possible [*c*5].
- **Code generation.** As a PLCspecif specification describes the complete behaviour of a PLC program, it is possible to generate automatically an equivalent implementation. I have defined the formal semantics of PLCspecif in a way that it is close to a control flow graph. Based on this, I designed and developed a code generation method that creates Siemens SCL code for a given PLCspecif specification. However, it is not enough to have a generated code with a behaviour equivalent to the specification's. For maintainability reasons it is also required to have a readable, understandable generated code. Therefore the generated code follows the structure of the specification, and the code generator was designed to

be configurable, e.g. the structure of the generated code and the representation of state machines or enumerations can be adjusted; thus the generated code can fit into the rest of the code base.

- **Conformance checking.** In many real use cases, an implementation (PLC program) and a specification are given, but their equivalence is not known. For example this can be the case when a formal specification is developed for a legacy implementation, or when the generated code was manually modified, then the specification was also modified to reflect the code change. Behaviour equivalence checking can show the equivalence or differences between the checked artefacts. I have designed a method to check behavioural equivalence by translating the specification into intermediate models used by PLCverif and to create a composite model of the two artefacts. With this composite model a model checker via PLCverif can show or refute the equivalence.

  In practice, PLCs are often used to control slow processes, where small delays in the output do not have any observable effect on the controlled process. Requiring a strict equivalence between two artefacts in these cases results in false positives, i.e. differences that are acceptable for the users. A high number of false positives undermines the usability of equivalence checking. Therefore I have identified typical cases when strict equivalence is not required between a pair of outputs in the checked artefacts, and I defined more *permissive conformance relations*. This way the PLC developers can define for each output pair the level of conformance they require. The conformance relations are defined formally [*c*6], along with temporal logic expressions that express the violation of the relations, making their verification possible using model checkers.

Thesis 3 I designed *PLCspecif*, a formal behaviour specification language adapted to the needs identified in PLC program development. This language is designed to be used for code generation and verification purposes.

- 3.1 I designed the main language concepts, then I defined the precise syntax and semantics of this language. The design of the language is based on collecting the requirements by analysing the literature of formal specification methods and on the feedback from the PLC developer community at CERN.
- 3.2 I developed a transformation from the PLCspecif specification to the intermediate model (IM) language used by PLCverif. This allows the usage of various model checkers to verify the invariant properties of the specification.
- 3.3 I designed and implemented an automated code generation method for the PLCspecif specification language based on its formal semantics. This code generation method is flexible and configurable, and produces Siemens SCL code that is systematically derived from the formal specification.
- 3.4 I designed new conformance relations, which can be used in the PLC software development domain and allow designers to define acceptable discrepancies between the specified and implemented behaviours (such as short delays in output signals). I defined a conformance checking method based on model checking to determine whether or not a relation holds for an implementation-specification pair. Accordingly, these relations provide means to verify a legacy implementation or a manually-modified generated code against a specification. I provided an implementation for checking these relations and evaluated their practical applicability.

The importance of these results is that the proposed specification language allows the precise, complete, formal definition of the behaviour of a PLC module without requiring extensive knowledge about formal methods or mathematical logic. The precise and complete specification

allows the usage of equivalence and conformance checking which can serve as a complementary method to pattern-based model checking. The case study in [*c*9] showed that conformance checking can reveal problems of real, safety-critical PLC programs used at CERN that were not found by thorough model checking of the pattern-based property specification. The introduction of permissive conformance relations makes this verification method useful in cases where the strict equivalence is not required. Furthermore, the PLCspecif specification language can reduce the manual implementation effort by providing a code generation solution suitable for the targeted application domain.

István Majzik was taking part in this research as my Ph.D. thesis supervisor. Enrique Blanco Viñuela was taking part in this research as my advisor in the CERN Doctoral Student Programme.

The results of Thesis 3 are presented in Chapter 4 of the dissertation. Related publications are the following: [*c*5; *c*6; *c*7; *c*8; *c*9; *c*12; *e*19; *r*22].

2.4 Summary of the Proposed Verification Methods

These theses proposed the usage of model checking in various settings, e.g. using different models and requirements. Table 2 compares the main aspects of the proposed approaches.

1. **Direct model checking** (Thesis 1). In the first thesis, model checking is used without any additional aid or adaptation. The formal models and requirements are created manually, in this case using Petri nets and CTL. Then the B-I-Sat model checking algorithm (implemented in the PetriDotNet framework [*c*10]) is executed and its results are evaluated manually. This is a suitable method for example when a non-deterministic model is needed that over-approximates the set of behaviours, or when the verification is a rare event and it is not efficient to develop a dedicated toolchain for the given platform, e.g. in case of the verification of the so-called PRISE safety logic.

2. **Model checking based on requirement patterns and PLC code** (Thesis 2). The inputs of this workflow (implemented in PLCverif [*c*13]) are the implementation (PLC code) and a filled requirement pattern. Both are understandable for the PLC developers. The implementation is then automatically translated to an intermediate model. A wrapper of the model checker takes care of transforming the model and requirement from an intermediate representation to the concrete syntax of the model checker. The result of the verification process is a verification report, readable for the users. This workflow is suitable for users without extensive formal verification knowledge. Additionally, thanks to the automation and adaptation, this method can be included in the PLC development process efficiently, without excessive effort.

3. **Conformance checking** (Thesis 3). Conformance checking takes a formal specification besides the PLC code as input. Both artefacts can be transformed into IM, then reduced and combined into a composite model in order to perform conformance checking. Based on the selected conformance relations a temporal logic requirement is generated. Then the wrapped model checker evaluates the satisfaction of this requirement on the composite model, similarly to use case #2. The result is presented in a verification report. If a formal specification is available, conformance checking provides an easy-to-use and thorough way to compare the implemented and the expected behaviour.

4. **Code generation and invariant property checking** (Thesis 3). If the implementation is generated using a correct code generator, there is no need to compare the behaviours of the implementation and the specification. However, in this case model checking can be used in the specification phase to ensure the satisfaction of invariant and/or safety properties. For this, the formal specification is transformed into IM, like in use case #3.

The invariant properties are defined using requirement patterns. Then the wrapped model checker evaluates the satisfaction of the invariant property and a report is generated.

Table 2. Overview of the proposed model checking methods

	#1 Direct model checking	#2 PLC code model checking	#3 Conformance checking	#4 Invariant checking
Model	Direct modelling	IM from implementation + Automated reductions	Composite IM from impl. and formal specif. + Automated reductions	IM from formal specif. + Automated reductions
Requirement	Manual	From requirement pattern	From conformance relation	From requirement pattern
Model checker	Unwrapped	Wrapped	Wrapped	Wrapped
Result	Raw	Verification report	Verification report	Verification report

3 Application of the New Results

3.1 Saturation-Based Bounded Model Checking in Practice

The saturation-based bounded model checking algorithms were implemented in the PetriDotNet framework[3] [*c*10]. The efficiency of these algorithms was checked on various benchmark models. Besides that, the algorithms were applied in the verification of a safety logic (named PRISE) used in the Paks nuclear power plant in Hungary [*j*2]. This safety logic was successfully verified before using different saturation algorithms [*c*28; *j*26], but bounded model checking provided lower verification times in several cases [*j*2].

Variants of the developed B-I-Sat algorithms were used for test generation purposes [*e*20] in the frame of the R3-COP Artemis project [R3C].

3.2 Model Checking Critical Industrual Control Systems Using PLCverif

In the frame of my research, I have developed a tool called PLCverif[4] that implements the discussed PLC model checking workflow [*c*13]. This tool made model checking applicable in various industrial control system development projects at CERN. This included the verification of basic building blocks of the UNICOS framework [*c*16; *j*3], verification of larger control applications, such as a cryogenics subsystem of the Large Hadron Collider (LHC) [*j*3] and the safety logic of a special, safety-critical controller used in the SM18 Cryogenics Test Facility [*c*9].

3.3 Formal Specification of Industrual Control Systems Using PLCspecif

To support the PLCspecif[5] formal specification language, I have developed a prototype implementation of the specification tool. This demonstrated the real-life applicability of the incorporated code generation methods [*c*7].

This prototype implementation is integrated with the PLCverif tool for verification (model checking and conformance checking) purposes. This combination was used in the verification of the safety logic used in the SM18 Cryogenics Test Facility [*c*9], and the conformance checking using the PLCspecif specification successfully identified errors in a safety-critical implementation that was already verified using model checking of pattern-based property specification, proving the complementarity of the property-oriented (pattern-based) and behavioural (PLCspecif-based) specification checking methods.

[3] http://petridotnet.inf.mit.bme.hu/en/
[4] http://cern.ch/plcverif/
[5] http://cern.ch/plcspecif/

4 Publication List

Number of publications:	29
Number of peer-reviewed journal papers (written in English):	6
Number of articles in journals indexed by WoS or Scopus:	6
Number of publications (in English) with at least 50% contribution of the author:	3
Number of peer-reviewed publications:	22
Number of independent citations:	23

4.1 Publications Linked to the Theses

	Journal papers	International conference and workshop papers	Local events	Technical reports
Thesis 1	[*j*2],[*j*4]	[*c*10],[*c*17],[*c*18]	[*e*20],[*e*21]	—
Thesis 2	[*j*1],[*j*3]	[*c*8]*,[*c*9]*,[*c*11],[*c*13],[*c*14],[*c*15],[*c*16]	—	[*r*23],[*r*24]
Thesis 3	—	[*c*5],[*c*6],[*c*7],[*c*8]*,[*c*9]*,[*c*12]	[*e*19]	[*r*22]

* These publications are attached to multiple theses.
This classification follows the faculty's Ph.D. publication score system.

Journal Papers

[*j*1] Dániel Darvas, István Majzik, and Enrique Blanco Viñuela. PLC program translation for verification purposes. *Periodica Polytechnica, Electrical Engineering and Computer Science*, 2017. URL: http://mit.bme.hu/~darvas/publications/PerPol2017_DarvasEtAl.pdf. Accepted, under publication.

▷ *Own contribution, joint paper with Ph.D. supervisors.*

[*j*2] Dániel Darvas, András Vörös, and Tamás Bartha. Improving saturation-based bounded model checking. *Acta Cybernetica* 22(3), 2016, pp. 573–589. DOI: 10.14232/actacyb.22.3.2016.2.

▷ *Own contribution, joint paper with M.Sc. thesis supervisors.*

[*j*3] Borja Fernández Adiego, Dániel Darvas, Enrique Blanco Viñuela, Jean-Charles Tournier, Simon Bliudze, Jan Olaf Blech, and Víctor M. González Suárez. Applying model checking to industrial-sized PLC programs. *IEEE Transactions on Industrial Informatics* 11(6), 2015, pp. 1400–1410. DOI: 10.1109/TII.2015.2489184. $IF_{2014} = 8.78$.

▷ *The intermediate representation of the SFC language and the intermediate model is my contribution. The PLC behaviour description and the variable abstraction method are contributions of B. Fernández Adiego. The rest of the verification method is a joint work with B. Fernández Adiego. The contribution of J.O. Blech is the discussion about the correctness of the approach. E. Blanco Viñuela, J-C. Tournier, S. Bliudze and V.M. González Suárez were helping the work as advisors.*

[*j*4] András Vörös, Dániel Darvas, and Tamás Bartha. Bounded saturation-based CTL model checking. *Proceedings of the Estonian Academy of Sciences* 62(1), 2013, pp. 59–70. DOI: 10.3176/proc.2013.1.07. $IF_{2013} = 0.37$.

▷ *Own contribution, joint paper with M.Sc. thesis supervisors.*

International Conference and Workshop Papers

[*c*5] Dániel Darvas, István Majzik, and Enrique Blanco Viñuela. Well-formedness and invariant checking of PLCspecif specifications. In: *Proceedings of the 24th PhD Mini-Symposium*,

Budapest University of Technology and Economics, Department of Measurement and Information Systems, 2017. In press.

▷ *Own contribution, joint paper with Ph.D. supervisors.*

[*c*6] Dániel Darvas, István Majzik, and Enrique Blanco Viñuela. Conformance checking for programmable logic controller programs and specifications. In: *11th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pp. 29–36. IEEE, 2016. DOI: 10.1109/SIES.2016.7509409.

▷ *Own contribution, joint paper with Ph.D. supervisors.*

[*c*7] Dániel Darvas, Enrique Blanco Viñuela, and István Majzik. PLC code generation based on a formal specification language. In: *14th IEEE International Conference on Industrial Informatics (INDIN)*, pp. 389–396. IEEE, 2016. URL: http://mit.bme.hu/~darvas/publications/INDIN2016_DarvasEtAl.pdf.

▷ *Own contribution, joint paper with Ph.D. supervisors.*

[*c*8] Dániel Darvas. Practice-oriented formal methods for PLC programs of industrial control systems. In: *Proceedings of the PhD Symposium at iFM'16 on Formal Methods: Algorithms, Tools and Applications (PhD-iFM'16)*, Reykjavik University, 2016. URL: http://mit.bme.hu/~darvas/publications/PhD-iFM2016_Darvas.pdf. Extended abstract. Accepted and presented, in press.

[*c*9] Dániel Darvas, István Majzik, and Enrique Blanco Viñuela. Formal verification of safety PLC based control software. In: Erika Ábrahám and Marieke Huisman (eds.), *Integrated Formal Methods*, Lecture Notes in Computer Science, vol. 9681, pp. 508–522. Springer, 2016. DOI: 10.1007/978-3-319-33693-0_32.

▷ *Own contribution, joint paper with Ph.D. supervisors.*

[*c*10] András Vörös, Dániel Darvas, Vince Molnár, Attila Klenik, Ákos Hajdu, Attila Jámbor, Tamás Bartha, and István Majzik. PetriDotNet 1.5: Extensible Petri net editor and analyser for education and research. In: Fabrice Kordon and Daniel Moldt (eds.), *Application and Theory of Petri Nets and Concurrency*, Lecture Notes in Computer Science, vol. 9698, pp. 123–132. Springer, 2016. DOI: 10.1007/978-3-319-39086-4_9.

▷ *The implementation of the PetriDotNet framework is a joint work of the authors, based on the original work of Bertalan Szilvási. The saturation-based model checking algorithms were developed by D. Darvas and A. Jámbor, supervised by A. Vörös and T. Bartha. The design and development of bounded saturation-based algorithms are my contributions, supervised by A. Vörös and T. Bartha.*

[*c*11] Dániel Darvas, István Majzik, and Enrique Blanco Viñuela. Generic representation of PLC programming languages for formal verification. In: *Proceedings of the 23rd PhD Mini-Symposium*, pp. 6–9. Budapest University of Technology and Economics, Department of Measurement and Information Systems, 2016. DOI: 10.5281/zenodo.51064.

▷ *Own contribution, joint paper with Ph.D. supervisors.*

[*c*12] Dániel Darvas, Enrique Blanco Viñuela, and István Majzik. A formal specification method for PLC-based applications. In: Lou Corvetti, Kathleen Riches, and Volker R.W. Schaa (eds.), *Proceedings of the 15th International Conference on Accelerator and Large Experimental Physics Control Systems*, pp. 907–910. JACoW, 2015. DOI: 10.18429/JACoW-ICALEPCS2015-WEPGF091.

▷ *Own contribution, joint paper with Ph.D. supervisors.*

[*c*13] Dániel Darvas, Borja Fernández Adiego, and Enrique Blanco Viñuela. PLCverif: A tool to verify PLC programs based on model checking techniques. In: Lou Corvetti, Kathleen Riches, and Volker R.W. Schaa (eds.), *Proceedings of the 15th International Conference on Accelerator and Large Experimental Physics Control Systems*, pp. 911–914. JACoW, 2015. DOI:

10.18429/JACoW-ICALEPCS2015-WEPGF092.

▷ *The high-level ideas of the presented workflow are joint work of the authors. The detailed design and development of the tool is my contribution.*

[*c*14] Borja Fernández Adiego, Dániel Darvas, Enrique Blanco Viñuela, Jean-Charles Tournier, Víctor M. González Suárez, and Jan Olaf Blech. Modelling and formal verification of timing aspects in large PLC programs. In: Edward Boje and Xiaohua Xia (eds.), *Proceedings of the 19th IFAC World Congress*, IFAC Proceedings Volumes, vol. 47 (3), pp. 3333–3339. Elsevier, 2014. DOI: 10.3182/20140824-6-ZA-1003.01279.

▷ *The concrete time representation ("realistic approach", Section 4.1) is a joint contribution with B. Fernández Adiego. The abstract time representation (Section 4.2) is my own contribution. The refinement between the two approaches is the contribution of J.O. Blech.*

[*c*15] Dániel Darvas, Borja Fernández Adiego, András Vörös, Tamás Bartha, Enrique Blanco Viñuela, and Víctor M. González Suárez. Formal verification of complex properties on PLC programs. In: Erika Ábrahám and Catuscia Palamidessi (eds.), *Formal Techniques for Distributed Objects, Components, and Systems*, Lecture Notes in Computer Science, vol. 8461, pp. 284–299. Springer, 2014. DOI: 10.1007/978-3-662-43613-4_18.

▷ *The high-level ideas of the verification workflow are joint work of the authors. The development and analysis of the model reduction methods (Section 3) are my own contributions.*

[*c*16] Borja Fernández Adiego, Dániel Darvas, Jean-Charles Tournier, Enrique Blanco Viñuela, and Víctor M. González Suárez. Bringing automated model checking to PLC program development – A CERN case study. In: Jean-Jacques Lesage, Jean-Marc Faure, José E. Ribiero Cury, and Bengt Lennartson (eds.), *Proceedings of the 12th International Workshop on Discrete Event Systems*, IFAC Proceedings Volumes, vol. 47 (2), pp. 394–399. Elsevier, 2014. DOI: 10.3182/20140514-3-FR-4046.00051.

▷ *The case study presented in this paper is a joint contribution based on my detailed definition and implementation of the verification workflow.*

[*c*17] Dániel Darvas, András Vörös, and Tamás Bartha. Efficient saturation-based bounded model checking of asynchronous systems. In: Ákos Kiss (ed.), *Proceedings of the 13th Symposium on Programming Languages and Software Tools, SPLST'13*, pp. 259–273. Szeged, Hungary: University of Szeged, 2013. URL: http://petridotnet.inf.mit.bme.hu/publications/SPLST2013_DarvasVorosBartha.pdf.

▷ *Own contribution, joint paper with M.Sc. thesis supervisors.*

[*c*18] András Vörös, Dániel Darvas, and Tamás Bartha. Bounded saturation based CTL model checking. In: Jaan Penjam (ed.), *Proceedings of the 12th Symposium on Programming Languages and Software Tools, SPLST'11*, pp. 149–160. Tallinn, Estonia: Tallinn University of Technology, Institute of Cybernetics, 2011. URL: http://petridotnet.inf.mit.bme.hu/publications/SPLST2011_VorosDarvasBartha.pdf.

▷ *Own contribution, joint paper with B.Sc. thesis supervisors.*

Local Conference and Workshop Papers

[*e*19] Dániel Darvas, István Majzik, and Enrique Blanco Viñuela. Requirements towards a formal specification language for PLCs. In: *Proceedings of the 22nd PhD Mini-Symposium*, pp. 18–21. Budapest University of Technology and Economics, Department of Measurement and Information Systems, 2015. DOI: 10.5281/zenodo.14907.

▷ *Own contribution, joint paper with Ph.D. supervisors.*

[*e*20] Dániel Darvas and András Vörös. Szaturációalapú tesztbemenet-generálás színezett Petrihálókkal [in Hungarian; Saturation-based test input generation using coloured Petri nets]. In: *Mesterpróba 2013. Konferenciakiadvány*, pp. 48–51. Budapest University of Technol-

ogy and Economics, 2013. URL: http : / / petridotnet . inf . mit . bme . hu / publications / Mesterproba2013_Darvas.pdf.

▷ *Own contribution in frame of the R3-COP project, joint paper with M.Sc. thesis supervisor.*

[*e*21] Dániel Darvas. Szaturáció alapú korlátos modellellenőrzési technikák Petri-hálók analízisére [in Hungarian; Saturation based bounded model checking methods for the analysis of Petri nets]. In: *XVII. Fiatal Műszakiak Tudományos Ülésszaka*, pp. 83–86. Cluj Napoca, Romania: Erdélyi Múzeum-Egyesület Műszaki Tudományok Szakosztálya, 2012. URL: http://petridotnet.inf.mit.bme.hu/publications/FMTU2012_Darvas.pdf.

Technical Reports

[*r*22] Dániel Darvas, Enrique Blanco Viñuela, and István Majzik. *Syntax and semantics of PLC-specif*. Report EDMS 1523877. CERN, 2015. URL: https://edms.cern.ch/document/1523877.

▷ *Own contribution, joint report with Ph.D. supervisors.*

[*r*23] Borja Fernández Adiego, Dániel Darvas, Jean-Charles Tournier, Enrique Blanco Viñuela, Jan Olaf Blech, and Víctor M. González Suárez. *Automated generation of formal models from ST control programs for verification purposes*. Internal Note CERN-ACC-NOTE-2014-0037. CERN, 2014. URL: http://cds.cern.ch/record/1708853/.

▷ *The formal definition of the IM language is my contribution. The transformation of PLC programs to IM is a joint contribution of the authors.*

[*r*24] Dániel Darvas, Borja Fernández Adiego, and Enrique Blanco Viñuela. *Transforming PLC programs into formal models for verification purposes*. Internal Note CERN-ACC-NOTE-2013-0040. CERN, 2013. URL: http://cds.cern.ch/record/1629275/.

▷ *The definition of the IM language is my contribution. The transformation of PLC programs to IM and the NuSMV representation of the IM are joint contributions of the authors.*

4.2 Additional Publications (Not Linked to Theses)

Journal Papers

[*j*25] Vince Molnár, András Vörös, Dániel Darvas, Tamás Bartha, and István Majzik. Component-wise incremental LTL model checking. *Formal Aspects of Computing* 28(3), 2016, pp. 345–379. DOI: 10.1007/s00165-015-0347-x. $IF_{2014} = 0.80$.

[*j*26] András Vörös, Dániel Darvas, Attila Jámbor, and Tamás Bartha. Advanced saturation-based model checking of well-formed coloured Petri nets. *Periodica Polytechnica, Electrical Engineering and Computer Science* 58(1), 2014, pp. 3–13. DOI: 10.3311/PPee.2080.

International Conference and Workshop Papers

[*c*27] Vince Molnár, Dániel Darvas, András Vörös, and Tamás Bartha. Saturation-based incremental LTL model checking with inductive proofs. In: Christel Baier and Cesare Tinelli (eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, vol. 9035, pp. 643–657. Springer, 2015. DOI: 10.1007/978-3-662-46681-0_58.

[*c*28] Tamás Bartha, András Vörös, Attila Jámbor, and Dániel Darvas. Verification of an industrial safety function using coloured Petri nets and model checking. In: *Proceedings of the 14th International Conference on Modern Information Technology in the Innovation Processes of the Industrial Enterprises (MITIP 2012)*, pp. 472–485. Hungarian Academy of Sciences,

Computer and Automation Research Institute, 2012. URL: http://petridotnet.inf.mit.bme.hu/publications/MITIP2012_BarthaEtAl.pdf.

[*c*29] András Vörös, Tamás Bartha, Dániel Darvas, Tamás Szabó, Attila Jámbor, and Ákos Horváth. Parallel saturation based model checking. In: *Proceedings of the 10th International Symposium on Parallel and Distributed Computing (ISPDC)*, pp. 94–101. IEEE, 2011. DOI: 10.1109/ISPDC.2011.23.

4.3 Additional Work

[*a*30] Dániel Darvas. Incremental extension of the saturation algorithm-based bounded model checking of Petri nets. Master's thesis. Budapest University of Technology and Economics, 2014. URL: http://petridotnet.inf.mit.bme.hu/publications/Diplomaterv2013_Darvas.pdf.

[*a*31] Dániel Darvas. Petri-háló alapú formális modellek analízise hatékony korlátos modellellenőrzési technikák segítségével [in Hungarian; Efficient bounded model checking techniques for Petri net based formal models]. Bachelor's thesis. Budapest University of Technology and Economics, 2011.

[*a*32] Dániel Darvas and Attila Jámbor. Komplex rendszerek modellezése és verifikációja [in Hungarian; Modeling and verification of complex systems]. Scientific Students' Association Report. 2011. URL: http://petridotnet.inf.mit.bme.hu/publications/TDK2011_DarvasJambor.pdf.

▷ *The extension of the saturation algorithms to coloured Petri nets is the contribution of A. Jámbor. The saturation-based bounded model checking algorithm is my own contribution.*

[*a*33] Dániel Darvas. Szaturáció alapú automatikus modellellenőrző fejlesztése aszinkron rendszerekhez [in Hungarian; Implementing a saturation-based model checker of asynchronous systems]. Scientific Students' Association Report. 2010. URL: http://petridotnet.inf.mit.bme.hu/publications/OTDK2011_Darvas.pdf.

References

[Abr96] Jean-Raymond Abrial. *The B-book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science* 126(2), 1994, pp. 183–235. DOI: 10.1016/0304-3975(94)90010-8.

[Ama] José Nelson Amaral. About Computing Science Research Methodology. URL: https://webdocs.cs.ualberta.ca/~c603/readings/research-methods.pdf.

[Amn+01] Tobias Amnell et al. UPPAAL – Now, next, and future. In: *Modeling and Verification of Parallel Processes*, Lecture Notes in Computer Science, vol. 2067, pp. 99–124. Springer, 2001. DOI: 10.1007/3-540-45510-8_4.

[Avi+04] Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing* 1(1), 2004, pp. 11–33. DOI: 10.1109/TDSC.2004.2.

[BBK12] Sebastian Biallas, Jörg Brauer, and Stefan Kowalewski. Arcade.PLC: A verification platform for programmable logic controllers. In: *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pp. 338–341. IEEE, 2012. DOI: 10.1145/2351676.2351741.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[Bla77] Michel Blanchard. Le GRAFCET pour une representation normalisée de cahier des charges d'un automatisme logique. *Automatique et Informatique Industrielle* (61), 1977, pp. 27–32.

[Bur+92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation* 98(2), 1992, pp. 142–170. DOI: 10.1016/0890-5401(92)90017-A.

[But02] Michael Butler. A system-based approach to the formal development of embedded controllers for a railway. *Design Automation for Embedded Systems* 6(4), 2002, pp. 355–366. DOI: 10.1023/A:1016503426126.

[Can+00] Géraud Canet, Sandrine Couffin, Jean-Jacques Lesage, Antoine Petit, and Philippe Schnoebelen. Towards the automatic verification of PLC programs written in instruction list. In: *IEEE International Conference on Systems, Man & Cybernetics*, vol. 4, pp. 2449–2454. IEEE, 2000. DOI: 10.1109/ICSMC.2000.884359.

[Cav+14] Roberto Cavada et al. The nuXmv symbolic model checker. In: *Computer Aided Verification*, Lecture Notes in Computer Science, vol. 8559, pp. 334–342. Springer, 2014. DOI: 10.1007/978-3-319-08867-9_22.

[CE82] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In: *Logic of Programs*, Lecture Notes in Computer Science, vol. 131, pp. 52–71. Springer, 1982. DOI: 10.1007/BFb0025774.

[CES09] Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis. Model checking: Algorithmic verification and debugging. *Communications of the ACM* 52(11), 2009, pp. 74–84. DOI: 10.1145/1592761.1592781.

[Cha+02] Pankaj Chauhan, Edmund M. Clarke, James Kukula, Samir Sapra, Helmut Veith, and Dong Wang. Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis. In: Mark D. Aagaard and John W. O'Leary (eds.), *Formal Methods in Computer-Aided Design*, Lecture Notes in Computer Science, vol. 2517, pp. 33–51. Springer, 2002. DOI: 10.1007/3-540-36126-X_3.

[Cla08] Edmund M. Clarke. The birth of model checking. In: Orna Grumberg and Helmut Veith (eds.), *25 Years of Model Checking*, Lecture Notes in Computer Science, vol. 5000, pp. 1–26. Springer, 2008. DOI: 10.1007/978-3-540-69850-0_1.

[CLS01] Gianfranco Ciardo, Gerald Lüttgen, and Radu Siminiceanu. Saturation: An efficient iteration strategy for symbolic state-space generation. In: *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, vol. 2031, pp. 328–342. Springer, 2001. DOI: 10.1007/3-540-45319-9_23.

[CMS08] José C. Campos, José Machado, and Eurico Seabra. Property patterns for the formal verification of automated production systems. In: *Proceedings of the 17th IFAC World Congress*, IFAC Proceedings Volumes, vol. 41 (2), pp. 5107–5112. Elsevier, 2008. DOI: 10.3182/20080706-5-KR-1001.00858.

[CNQ05] Gianpiero Cabodi, Sergio Nocco, and Stefano Quer. Are BDDs still alive within sequential verification? *International Journal on Software Tools for Technology Transfer* 7(2), 2005, pp. 129–142. DOI: 10.1007/s10009-004-0172-7.

[Cop+01] Fady Copty, Limor Fix, Ranan Fraer, Enrico Giunchiglia, Gila Kamhi, Armando Tacchella, and Moshe Y. Vardi. Benefits of bounded model checking at an industrial setting. In: Gérard Berry, Hubert Comon, and Alain Finkel (eds.), *Computer Aided Verification*, Lecture Notes in Computer Science, vol. 2102, pp. 436–453. Springer, 2001. DOI: 10.1007/3-540-44585-4_43.

[CS03] Gianfranco Ciardo and Radu Siminiceanu. Structural symbolic CTL model checking of asynchronous systems. In: *Computer Aided Verification*, Lecture Notes in Computer Science, vol. 2725, pp. 40–53. Springer, 2003. DOI: 10.1007/978-3-540-45069-6_4.

[CZJ12] Gianfranco Ciardo, Yang Zhao, and Xiaoqing Jin. Ten years of saturation: A Petri net perspective. In: *Transactions on Petri Nets and Other Models of Concurrency V*, Lecture Notes in Computer Science, vol. 6900, pp. 51–95. Springer, 2012. DOI: 10.1007/978-3-642-29072-5_3.

[DAC99] Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In: *Proceedings of the 21st International Conference on Software Engineering*, pp. 411–420. ACM, 1999. DOI: 10.1145/302405.302672.

[FBM13] Borja Fernández Adiego, Enrique Blanco Viñuela, and Alexey Merezhin. Testing & verification of PLC code for process control. In: *Proceedings of the 14th International Conference on Accelerator & Large Experimental Physics Control Systems*, pp. 1258–1261. JACoW, 2013. URL: http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/thppc080.pdf.

[Fer+13] Borja Fernández Adiego, Enrique Blanco Viñuela, Jean-Charles Tournier, Víctor M. González Suárez, and Simon Bliudze. Model-based automated testing of critical PLC programs. In: *11th IEEE International Conference on Industrial Informatics (INDIN)*, pp. 722–727. 2013. DOI: 10.1109/INDIN.2013.6622973.

[Fer14] Borja Fernández Adiego. Bringing automated formal verification to PLC program development. Ph.D. thesis. University of Oviedo, 2014. URL: http://cds.cern.ch/record/1983193.

[Fix08] Limor Fix. Fifteen years of formal property verification in Intel. In: Orna Grumberg and Helmut Veith (eds.), *25 Years of Model Checking*, Lecture Notes in Computer Science, vol. 5000, pp. 139–144. Springer, 2008. DOI: 10.1007/978-3-540-69850-0_8.

[Fok00] Wan Fokkink. *Introduction to Process Algebra*. 1st edition. Springer, 2000. DOI: 10.1007/978-3-662-04293-9.

[GSF08] Vincent Gourcuff, Olivier de Smet, and Jean-Marc Faure. Improving large-sized PLC programs verification using abstractions. In: *Proceedings of the 17th IFAC World Congress*, IFAC Proceedings Volumes, vol. 41 (2), pp. 5101–5106. Elsevier, 2008. DOI: 10.3182/20080706-5-KR-1001.00857.

[Hav+00] Klaus Havelund, Mike Lowry, Seung Joon Park, Charles Pecheur, John Penix, Willem Visser, and Jon L. White. Formal analysis of the Remote Agent before and after flight. In: C. Michael Holloway (ed.), *Lfm2000: Fifth NASA Langley Formal Methods Workshop*, pp. 163–174. NASA, 2000.

[HLR98] Mats P.E. Heimdahl, Nancy G. Leveson, and Jon D. Reese. Experiences from specifying the TCAS II requirements using RSML. In: *Proceedings of the 17th AIAA/IEEE/SAE Digital Avionics Systems Conference*, vol. 1, pp. C43/1–C43/8. IEEE, 1998. DOI: 10.1109/DASC.1998.741499.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985. URL: http://www.usingcsp.com/cspbook.pdf.

[I1012] *IEEE Std 1012-2012 – IEEE Standard for system and software verification and validation*. IEEE, 2012.

[I13568] *ISO/IEC 13568 Information technology – Z formal specification notation – Syntax, type system and semantics*. ISO/IEC, 2002.

[I60848] *IEC 60848 – GRAFCET specification language for sequential function charts*. IEC, 2013.

[I61131-3] *IEC 61131-3 Programmable controllers – Part 3: Programming languages*. IEC, 2013.

[I61508-2] *IEC 61508-2 Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 2: Requirements for electrical/electronic/programmable electronic safety-related systems*. IEC, 2010.

[I8807] *ISO 8807 Information processing systems – Open Systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour*. ISO, 1989.

[Kai+09] Roope Kaivola et al. Replacing testing with formal verification in Intel® Core™ i7 processor execution engine validation. In: Ahmed Bouajjani and Oded Maler (eds.), *Computer Aided Verification*, Lecture Notes in Computer Science, vol. 5643, pp. 414–429. Springer, 2009. DOI: 10.1007/978-3-642-02658-4_32.

[Kan+15] Gijs Kant, Alfons Laarman, Jeroen Meijer, Jaco van de Pol, Stefan Blom, and Tom van Dijk. LTSmin: High-performance language-independent model checking. In: Christel Baier and Cesare Tinelli (eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, vol. 9035, pp. 692–707. Springer, 2015. DOI: 10.1007/978-3-662-46681-0_61.

[Kni+97] John C. Knight, Colleen L. DeJong, Matthew S. Gibble, and Luís G. Nakano. Why are formal methods not used more widely? In: C. Michael Holloway and Kelly J. Hayhurst (eds.), *Fourth NASA Langley Formal Methods Workshop (LFM)*, pp. 1–12. 1997. URL: http://www.cs.virginia.edu/~jck/publications/lfm.97.pdf.

[Lam00] Axel van Lamsweerde. Formal specification: A roadmap. In: Anthony Finkelstein (ed.), *Proceedings of the Conference on The Future of Software Engineering (ICSE)*, pp. 147–159. ACM, 2000. DOI: 10.1145/336512.336546.

[Lam09] Axel van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[Lju+10] Oscar Ljungkrantz, Knut Åkesson, Martin Fabian, and Chengyin Yuan. A formal specification language for PLC-based control logic. In: *Proceedings of the 8th IEEE International Conference on Industrial Informatics (INDIN)*, pp. 1067–1072. 2010. DOI: 10.1109/INDIN.2010.5549591.

[LNN13] Tim Lange, Martin R. Neuhäußer, and Thomas Noll. Speeding up the safety verification of programmable logic controller code. In: *Hardware and Software: Verification and Testing*, Lecture Notes in Computer Science, vol. 8244, pp. 44–60. Springer, 2013. DOI: 10.1007/978-3-319-03077-7_4.

[LSP07] Thierry Lecomte, Thierry Servat, and Guilhem Pouzancre. Formal methods in safety-critical railway systems. In: *Proceedings of the 10th Brasilian Symposium on Formal Methods (SBMF)*, 2007. URL: http://rodin.cs.ncl.ac.uk/Publications/fm_sc_rs_v2.pdf.

[Luk+13] Tomaž Lukman, Giovanni Godena, Jeff Gray, Marjan Heričko, and Stanko Strmčnik. Model-driven engineering of process control software – Beyond device-centric abstractions. *Control Engineering Practice* 21(8), 2013, pp. 1078–1096. DOI: 10.1016/j.conengprac.2013.03.013.

[Mar94] John J. Marciniak. *Encyclopedia of Software Engineering*. Vol. 1. John Wiley & Sons, 1994. DOI: 10.1002/0471028959.

[MB08] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In: C. R. Ramakrishnan and Jakob Rehof (eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer, 2008. DOI: 10.1007/978-3-540-78800-3_24.

[Mur89] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* 77(4), 1989, pp. 541–580. DOI: 10.1109/5.24143.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In: Mariangiola Dezani-Ciancaglini and Ugo Montanari (eds.), *International Symposium on Programming*, Lecture Notes in Computer Science, vol. 137, pp. 337–351. Springer, 1982. DOI: 10.1007/3-540-11494-7_22.

[R3C] *R3-COP final dissemination, demonstration, PR and E&T report (Period 3+, M42)*. Tech. rep. WP 8, D 8.1.4c+d, D 8.1.6. 2013. URL: http://www.r3-cop.eu/wp-content/uploads/2013/10/R3-COP_D-8.1.4d_-D-8.1.6_1.2r_Dissemination_Final.pdf.

[SD08] André Sülflow and Rolf Drechsler. Verification of PLC programs using formal proof techniques. In: Géza Tarnai and Eckehard Schnieder (eds.), *Formal Methods for Automation and Safety in Railway and Automotive Systems*, pp. 43–50. L'Harmattan, 2008. URL: http://www.informatik.uni-bremen.de/agra/doc/konf/08_forms_VerificationPLCPrograms.pdf.

[SF11] Doaa Soliman and Georg Frey. Verification and validation of safety applications based on PLCopen safety function blocks. *Control Engineering Practice* 19(9), 2011, pp. 929–946. DOI: 10.1016/j.conengprac.2011.01.001.

[Sou+09] Jean Souyris, Virginie Wiels, David Delmas, and Hervé Delseny. Formal verification of avionics software products. In: Ana Cavalcanti and Dennis R. Dams (eds.), *FM 2009: Formal Methods*, Lecture Notes in Computer Science, vol. 5850, pp. 532–546. Springer, 2009. DOI: 10.1007/978-3-642-05089-3_34.

[Woo+09] Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. Formal methods: Practice and experience. *ACM Computing Surveys* 41(4), 2009, 19:1–19:36. DOI: 10.1145/1592434.1592436.

[YCL09] Andy Jinqing Yu, Gianfranco Ciardo, and Gerald Lüttgen. Decision-diagram-based techniques for bounded reachability checking of asynchronous systems. *International Journal on Software Tools for Technology Transfer* 11(2), 2009, pp. 117–131. DOI: 10.1007/s10009-009-0099-0.

[ZC09] Yang Zhao and Gianfranco Ciardo. Symbolic CTL model checking of asynchronous systems using constrained saturation. In: *Automated Technology for Verification and Analysis*, Lecture Notes in Computer Science, vol. 5799, pp. 368–381. Springer, 2009. DOI: 10.1007/978-3-642-04761-9_27.