MŰEGYETEM 1782

Budapest University of Technology and Economics
Department of Measurement and Information Systems

Symbolic Verification of Petri Net Based Models

PhD thesis booklet

András Vörös

Advisor:
Tamás Bartha, Ph.D. (BME)

Budapest, 2018

1 Preliminaries and Objectives

Ensuring the correctness of systems is a long-standing requirement in the engineering disciplines. Engineers have been using various techniques to analyse their projects and find design problems before the implementation. Various means from the field of mathematics and physics helped to build more stable buildings, more robust and stronger machines and so on. The design workflow depicted on Figure 1, which has worked for many centuries in the engineering disciplines is also applied in the design of computer-based ICT (Information and Communication Technology) systems. However, the analysis of complex ICT systems requires new techniques and algorithms [BKL08] compared to the traditional engineering domains like the mechanical engineering domain or architecture.

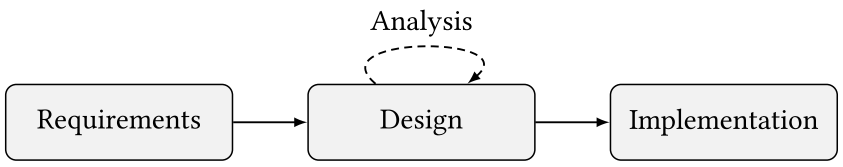

Figure 1: Development process

In my dissertation, I focus on the correctness analysis of critical ICT systems, and specifically the logical correctness checking, i.e. the verification of such systems. In my work I have investigated how the development can be supported by modelling languages, verification algorithms, and a framework, making all these techniques available to the computer engineers. The outcome of the verification process will help the engineers producing systems with better quality and fewer errors.

1.1 Target of the Dissertation

A system is safety-critical if its failure could result in loss of life or significant damage. There are many well-known safety-critical areas such as medical devices, aircraft flight control and nuclear systems. Ensuring the correctness of these systems is especially important, in which advanced verification techniques play a significant role.

Safety-critical systems are inherently distributed, components responsible for various functions in these systems cooperate to keep up the proper operation. The distributed characteristics of the components and their interaction results in intricate system level behaviour. This fact raises the main challenge: the resulting behaviour is not only difficult to understand and to modify, but also to analyse.

Due to the technological development, recent safety-critical systems are becoming more and more complex, raising challenges in the modelling, development and also in the verification. In my thesis, I aim to provide solutions to support the modelling and verification of safety-critical systems. As no single approach can cover all aspects of ICT systems, in my work I focus on the verification of asynchronous systems, such as communication and distributed systems.

Verification analyses if the model of the system fulfils the given correctness criteria. Various kinds of requirements [BKL08] are expected to be fulfilled by the system:

- Safety requirements express that the system does not reach an error/dangerous state.
- Boundedness properties express restrictions to the resource usages and other aspects of the system.

- Liveness properties expect the system to respond to a request after finite delay and also avoid deadlock, for example, a client will finally send an answer to a request.
- Reversible systems can reach certain states again and again.
- Persistence requirements express that some property will finally hold in the system after a transient phase. For example, in a distributed system, the connection will be established, and it remains in that state stably.

In my dissertation, I aim to support a rich set of properties to specify the correctness requirements.

1.2 Verification Techniques in Systems Engineering

In this section, a typical engineering workflow based on the widely known V-model is used to show the role of verification techniques at the various phases of development (the V-model received its name as it forms a V shape). This workflow is used as a general guideline in the development of safety-critical systems. Many variants of this workflow were developed by the industry tailored to the special needs of the different sectors. The V-model defines the elementary steps and draws a general workflow for the design and implementation of the system as depicted on Figure 2. In addition, the workflow defines verification and validation steps in the development to ensure that the correct design is developed and it fulfils the requirements. In the following, the various verification and validation techniques are summarised that ensure the correctness at the different phases of the development. The V-model defines the verification goals to ensure correctness of:

- the design with regard to the requirements and
- the implementation with regard to the design.

The systems engineering process depicted on Figure 2 starts by designing the requirements, which are then refined in the next, so-called system design phase. This phase defines the main functionalities of the system. In the next phase, designing the architecture provides the necessary decomposition to be able to construct the component level design. At each step, the designer refines the outcome of the former steps by providing more details. At the final step of the left wing of the V-model, one can produce the implementation for each component from the design models. Implementation has to be tested and verified against coding and other implementation errors. After the component level validation, system integration builds the smaller pieces together where extensive integration testing is executed to validate that the components work properly together. Finally, system validation ensures if the outcome is the system, which is desired by the customer.

Various analysis techniques serve the verification and validation of the system design and implementation. In the following these analysis techniques, such as (computer) simulation, testing and formal verification are shortly summarised:

Simulation is the process of executing the model of a system. Simulation is a design-time activity to assess the dynamic aspects of systems.

Testing is an activity in which a system or component is executed under specified conditions, the results are observed or recorded, and evaluated according to the specification [Ins10; Mic13]. Systems are tested at various phases of the development [Ana+13] from component level implementation [GA14] up to system level integration.

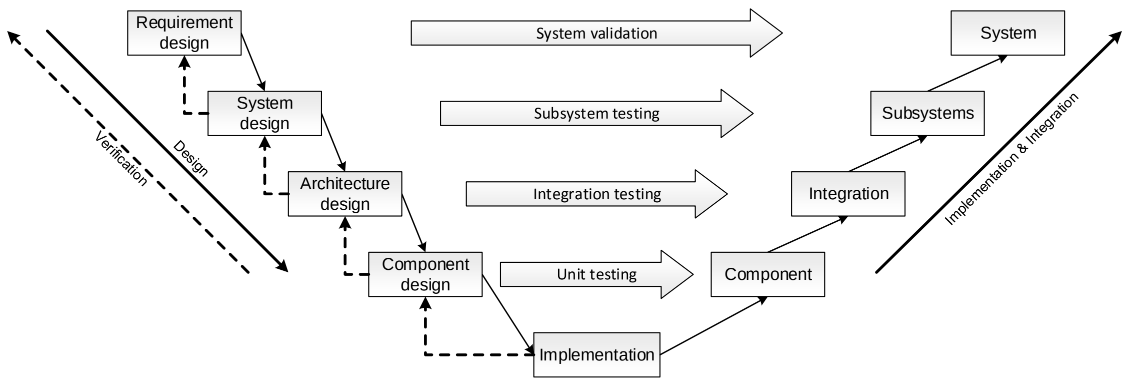

Figure 2: Verification during system engineering

Formal verification is the procedure of proving or disproving the correctness of a system with respect to a certain formal specification or property [CES86]. Formal verification is based on the mathematics of computation. Both design models and also implementation [Bey17] can be verified. However, as the models are becoming more detailed and the design approaches the implementation level, the computational complexity increases.

Simulation is the elementary task of inspecting and analysing the system behaviour. The prerequisite is the model of a real or imagined system, which shall be designed and then experiments are conducted on the model during simulation to reproduce the possible behaviour. The purpose of simulation experiments is to understand the behaviour of the system or evaluate strategies for the operation of the system. Simulation uses an abstract model (a computational model) for execution. Simulation provides analysis capabilities at an early stage of the development when only models are available.

The testing procedure can be carried out at various levels of abstraction. In one hand, testing the model by simulating it and evaluating the behaviour can provide feedback for the developers at an early stage of the design. Simulation is used as the elementary procedure to test models. On the other hand, testing the implementation provides inputs and observes the reactions of the concrete system. Testing is one of the most widely used verification approaches [MSB11].

In general, testing analyses the runs of the system by providing inputs, simulating the behaviour and examining the reaction (output), by comparing it to an explicitly stated (provided as assertions) or implicitly assumed (such as no crash should occur) expected behaviour. Testing is efficient in finding problems, and it has many advantages. Testing the model of the system relies on simulation, which can be computed efficiently. Besides, testing is easy-to-use for the developers: no additional knowledge is required, it works on the model of the system. Testing can also be applied at the implementation level so the revealed problems do not come from the inaccurate modelling but they are real problems in the implementation. In addition, when exhaustive verification is not possible, testing can still help locating problems.

On the other side, neither simulation nor testing can be complete in the sense that they usually can not explore all the behaviour of a system so neither simulation nor testing can prove correctness alone.

Formal verification extends their strengths with mathematically established proofs based on the exhaustive traversal of all the possible behaviour.

Finding errors is one side of the problem: the need to be able to prove correctness naturally raised. This need led to the development of formal verification techniques to support the engineers with tools providing certainty about the correctness of their design.

The combination of the various verification methods can ensure the high quality of computer-based systems. These techniques can be used in different phases of the systems engineering process, and they together constitute a powerful tool to find errors at an early stage of the development.

2 Formal Verification

Formal verification is the analysis of hardware, software and systems that provides mathematically established proofs for correctness or existing errors. Formal verification is performed on the abstract representation (model) of the system or directly on the source code.

Nowadays, the application of formal methods is gaining high importance in the development of modern ICT and especially safety-critical systems. Standards, like IEC 61508 also recommend the application of formal techniques in the development process.

2.1 Applying Formal Verification in System Design

The traditional application of verification is depicted on Figure 3 [13]. The goal is to verify the correctness of the system by checking if the engineering model fulfils the requirements [LMM99; Cse+02]. Applying formal verification in the system development process consists of the following steps.

1. Engineering models and requirements are developed.
2. Formal models and formal requirements are created.
3. The verification procedure is executed.
4. Results are interpreted and back-propagated to the engineering levels. Corrections are made, and verification is run again if needed.
5. Implementation is derived from the engineering models.

The input of the procedure is the engineering model, which is usually described by a domain expert in the language of design tools, for example, SysML or UML. The requirements also come from the engineers. Both the engineering models and the requirements have to be formalized and transformed into the input language of some verification tools. Formal models provide a mathematically precise way to describe the system: this enables the application of formal verification techniques on the systems' design. The result of the verification is interpreted on the level of the formal model and formal specification: this result has to be back-propagated to the engineering domain to be understood by the engineers. This back-propagation procedure is depicted on the figure with dashed lines pointing towards the engineering level.

Supporting the whole verification process involves all the aforementioned steps. In this thesis, I focus on how to support the formal modelling, and I will introduce novel verification algorithms to enhance the verification process.

2.2 Formal Modelling

Formal modelling is the process of developing a formal representation of the system under analysis in a formal modelling language. The resulting model can be analysed by various techniques to prove its correctness or find design errors.

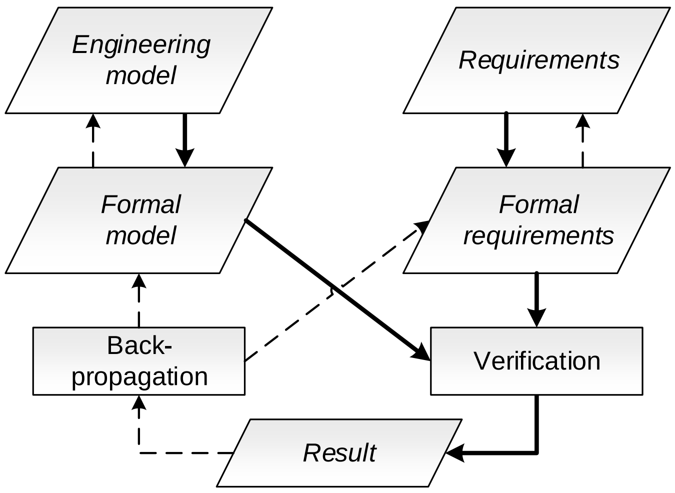

Figure 3: High-level view of the verification process

2.2.1 Development of the Formal Representation

There are two main directions to develop the formal models: they can be developed manually, or formal models are automatically generated from the engineering models by using model transformations. Verification engineers develop formal models from the system description. Increasing the expressiveness of the formal modelling language supports the efficient development of the formal models. It means a smaller abstraction gap between the engineering and formal modelling level and also provides more information for the underlying verification engines.

Many approaches try to support the automatic generation of formal models from engineering models, but they rarely provide a proper solution for the problem in their own as the generated formal models might contain too many details preventing successful verification [Dar17],[13],[12].

2.2.2 Formal Modelling Languages

There are many formalisms to represent the system under analysis. As systems possess various characteristics, formal representations have to be able to express these properties and exploit them for verification.

Finite state automaton and their extensions are popular as they are easy to use and an automaton can naturally represent certain problem domains. Additionally, networks of automata provide a compact representation for distributed systems. Programming language-like formalisms are popular for their expressiveness and as they are similar to those languages that software engineers are used to. Petri nets and related models constitute an expressive class of formal modelling languages: they are popular for their simplicity, but Petri nets still possess high expressive power. There are two main types of Petri nets:

Petri net based modelling languages solve the problem of graphical and formal representation of concurrent and distributed systems. Petri nets naturally handle the inherent asynchronism of such systems. Petri nets can represent both finite and infinite state systems. Moreover, various subclasses and extensions exist to support the modelling and analysis of concurrent systems.

Coloured Petri nets (CPN) extend Petri nets with various data types and variables, and additional guard expressions are used to refine the possible behaviour of the systems further.

Coloured Petri nets can raise the abstraction level to help the efficient development of formal models.

Ordinary Petri nets are well-suited to model control flow and data dependent behaviour. Petri net based models can have finite state space which means that a finite number of states are reachable from the initial state. Various subclasses exist for representing the various problems. Finite state machines and networks of finite state machines are expressed with finite Petri nets. The marked graph is a special subclass of Petri nets being able to represent decision-free parallel activities. For these Petri net subclasses, efficient verification methods exist [Mur89; BKP17].

On the other hand, Petri net based models can also represent infinite state systems, with an infinite number of reachable states. Such ordinary Petri nets are used to model concurrent multi-threaded programs with finite data structures. In general, the expressive power of Petri nets equals the expressive power of Vector Addition Systems [EN94].

In coloured Petri nets, various data types can be used as colour types, and guard expressions can be evaluated on them. Coloured Petri nets combine the inherent concurrency of Petri nets with data dependent behaviours expressed with colour types and other language elements. Coloured Petri nets are also a proper means to describe parametric systems. Compared to ordinary Petri nets where the structure of the net encodes the modelled behaviour, in coloured Petri nets, a wider range of language elements support the modelling.

2.3 Formal Requirements

Formal requirements capture the requirements of the design phase in a formally interpreted precise language. In this thesis, our goal is to verify the behaviour of systems, so we restrict the introduction to the formal requirement languages being able to express behavioural properties.

Formal requirements are usually expressed with the help of temporal logic. Various temporal logics exist, the two most common are Computation Tree Logic (CTL) and Linear Temporal Logic (LTL). They have different semantics and expressive power. For example, deadlock freedom can be only expressed with CTL while fairness properties are only expressible with LTL. It is desirable for a model checker to support both formalisms. However, only a few model checkers provide support for both of them.

2.4 Formal Verification Techniques

There are various formal verification techniques to ensure the correctness of systems [DKW08]. Formal verification does not rely on the concrete execution of the software/system, so these techniques are often referred to as static analysis techniques. Widely used static analysis techniques are – among others – abstract interpretation, model checking and theorem proving.

Abstract interpretation techniques derive properties from the structure of the models or the source code. Abstract interpretation iterates through the program and approximates the possible behaviours without executing the calculations of the program. Over-approximation and various abstractions tailored to the domain provide efficiency. However, accurate results can not be provided due to the coarse approximations.

Model checking analyses a model representation of the system. Model checking algorithms exhaustively explore the possible states of the system and verify the requirements given as temporal logic formulae. Model checking algorithms produce a counterexample if a property violation is found.

Theorem proving based verification reduces the verification problem to solving first-order or higher-order logic problems. First-order theorem provers might work fully automatically, higher-order logic provers are mainly interactive. Both the property and the system representation have to be expressed as a logic problem in the input language of the chosen theorem prover.

From the various approaches, there are semi-automatic procedures like interactive theorem proving, and on the other side, there are fully automatic techniques such as model checking and abstract interpretation. There is a huge gap also in precision: abstract interpretation examines safety properties being reduced to the reachability checking of some erroneous states. Model checking extends the set of analysis questions, and it is able to answer liveness or even complex fairness and timed properties. Theorem proving can prove even properties expressed in higher-order logic, however only for certain (very restricted subset of) systems. In general, static analysis techniques (like abstract interpretation) are known to be computationally cheaper but less precise, on the other side model checking is more precise for the more computational cost.

The approaches use different formalisms to design the system representation: the verification engineers can choose between directly verifying the implementation or execute the analysis on a higher level of abstraction. The first approach provides information directly from the implementation: this advantage is however very expensive as the verification of programs is algorithmically rarely tractable. Abstract models are usually easier – however in practice still difficult – to verify, but implementation and coding problems are not detected at this higher level.

Formal verification is getting more and more widely used for industrial problems [Cal+15; Adi+15; DMB16; LS09; Kle+09; SD10; BP12; Kai+09]. From the various techniques, model checking provides a good trade-off between precision, expressiveness and computational costs [DKW08].

2.4.1 Model Checking

Model checking is an automated formal verification technique: given a formal model representing the system, and a formal specification, a model checking algorithm traverses the possible behaviours of the formal model and decides if the formal specification is fulfilled. The state space is represented in the internal data structures of the model checker and used for the analysis of the formal properties. When the formal specification is fulfilled, the correctness of the design is proven. Otherwise, the model checker produces a counterexample, which shows how the system can reach an incorrect/undesired situation.

There are two main families of model checking algorithms: *explicit techniques* use traditional graph algorithms to explore the states one-by-one. On the other side, *symbolic model checking* algorithms apply special encoding of the state space and the transition relation. Explicit state model checking can be fast but often faces the so-called state space explosion problem: even small models can have huge state spaces, which can not fit into the memory of modern computers. Symbolic algorithms try to solve the state space explosion problem by avoiding the explicit representation of the state graph and using a compact representation instead. One of the symbolic approaches is saturation, which was devised for the verification of asynchronous, concurrent systems.

Model checking is a difficult problem in general: even small systems can have huge state space due to the large number of interactions of asynchronous, concurrent or distributed systems, or caused by the data content of the state variables. In many cases, formal models have infinite state spaces, which have to be traversed and represented by the model checking algorithms. Explicit model checking algorithms store the states and transitions one-by-one and traditional graph traversal algorithms are used: the memory requirements of storing huge state space graphs often prevent their application. On the contrary, symbolic model checking algorithms handle sets of states together instead of manipulating them individually and clever encodings help to fit the state space representation into the memory.

Symbolic state space representation. In symbolic model checking, characteristic functions are used to encode sets of states and decision diagrams can be used for efficient storage. A decision diagram is a directed acyclic graph, representing a Boolean or multi-valued function. Various reduction rules ensure that decision diagrams are a canonical and compact representation of a given function or set, which makes it a proper means to store set of states. Traditional symbolic algorithms encode the reachable states and also the next state functions in decision diagrams. State variables are mapped to the variables of the decision diagram and state vectors are stored in the decision diagram.

The other option is the application of *SAT-based* techniques to manipulate the characteristic function of the symbolic representation and efficient solvers help the state space traversal. Induction can help to find proofs for correctness or bounded state space exploration searches counterexamples.

Symbolic model checking algorithms can manipulate a huge set of states together, but their efficiency highly relies on the used encoding. Finding a good encoding can be a complex task.

Efficient state space traversal. In model checking, the state space has to be traversed. During the exploration, states have to be stored or memorised, to avoid redundant exploration, redundant computations. Decision diagrams offer a compact representation and storage for the state space, but the construction of the state space representation i.e. the exploration strategy of the state space has to be chosen. The states of synchronous hardware systems are traditionally explored by breadth-first (BFS) traversal in symbolic model checking. However, BFS is used to be inefficient for asynchronous systems [CMS05]. On the other hand, depth-first (DFS) traversal does not fit the traditional decision diagram based symbolic algorithms as symbolic algorithms are not able to handle states individually.

Ciardo and his colleagues developed a special iteration strategy to solve this problem, the so-called saturation iteration algorithm, which combines BFS and DFS strategies tailored to the structure of the decision diagram representation of the state space. Saturation is efficient[CMS05] for asynchronous and GALS (Globally Asynchronous Locally Synchronous) systems. Saturation was developed for Petri nets, and some extensions also support model checking of various properties.

Checking temporal logic specifications. CTL and LTL are widely used temporal logics with different expressiveness and different verification algorithms. CTL model checking is reduced to compute greatest and least fixed points of the next-state functions in the state space iteratively. This approach is called structural model checking and decision diagram based symbolic approaches efficiently solve the problem of traversing and storing the possible states. On the other side, LTL model checking requires different algorithms as it is reduced to the checking of language inclusion of the property automaton and the state space of the model. Model checking LTL properties is usually composed of two challenges: one must compute the synchronous product of the state space and the automaton model of the desired property, then look for counterexamples that is reduced to finding strongly connected components (SCCs) in the state space of the product. Checking LTL properties is computationally a harder problem than checking CTL properties in general.

2.5 Target Problem of the Dissertation

In this dissertation, I focus on the modelling and verification of concurrent, asynchronous systems, which constitute a significant part of the set of safety-critical systems. These are typically discrete-event systems (DES), so I show how such systems can be efficiently modelled and verified.

Summarising the verification challenges in general: there is a need to precisely (formally) represent the system and the requirements, efficient model checking algorithms are required to solve the verification problem, and we need tool support with the aforementioned capabilities.

3 New Challenges

In this section, I overview the challenges in the model checking process. In [BKL08], authors defined the following phases in the application of model checking in systems engineering:

- Modelling phase:
  - Model the system using the description language of the model checker.
  - Perform sanity checks by simulation
  - Formalize the requirements using a property specification language.
- Running phase: run the model checker and check the validity of the property in the model.
- Analysis phase:
  - If the property is satisfied, check the next property.
  - If the property is violated, then:
    1. Analyse the generated counterexample;
    2. Refine the model, design or the property;
    3. Repeat the entire procedure.
  - Out of memory or time-out necessitates the reduction of the model or the application of a different model checking algorithm.

Formal verification is often desirable though complex task and each phase of the model checking process has their own challenges. Developing formal models is time-consuming, and the verification of real industrial problems is computationally hard. The huge gap between engineering and formal models are difficult to bridge by automated techniques. On the other hand, verification engineers might develop the proper formal models from engineering models. However, this process is time-consuming. Even if the models are available in a formal representation, the requirements have to be also expressed formally, which needs a rich set of formal requirement languages. After all, the high computational cost of formal verification often prevents its successful application.

3.1 Model Checking of Asynchronous Systems

Formal verification is a computationally difficult problem: small systems still have huge state spaces, which has to be traversed by the verification algorithms. This is especially true for the asynchronous models of distributed systems: the various overlapping of the components' behaviour yields a huge number of possible behaviour. Advanced techniques are required to handle this explosion. The number of possible states grows exponentially with the growing number of components in a distributed system, even up to the Cartesian product of the states of the individual components. As formal verification has to be exhaustive, the large number of states poses huge challenges for the verification algorithms.

As it can be seen, formal verification of distributed systems is computationally expensive so choosing the proper approach is crucial. Saturation provides an efficient solution for the model checking of Petri net models: my work is based on saturation-based techniques.

Due to the computationally extensive nature of model checking, the question naturally arises that how modern multi-core processor could be exploited for further enhancing the performance of model checking. Efficient model checking approaches such as symbolic algorithms use complex data structures and iteration strategies making the parallelisation task difficult. The reason for symbolic model checking being inherently sequential is that fixed-point computation and detection needs the results of the previous steps. This problem is especially true for saturation, which was also discussed in the literature [CZJ09]. With the newer and newer advances of model checking, the challenge of exploiting multi-core computers in saturation-based model checking is raised.

3.2 Formal Modelling

Petri nets are a popular modelling language to describe the behaviour of concurrent and asynchronous systems, but many application domains require a more expressive formalism: in such cases, coloured Petri nets provide efficient means to describe asynchronous systems with data dependent behaviours.

Coloured Petri nets (CPN) extend ordinary Petri nets with various data types that can be used as colour types, and guard expressions can be evaluated on them. However, this also yields challenges for the verification algorithms. These intricate language elements of coloured Petri nets are difficult to be handled by the model checking algorithms. This issue should be addressed in a formal verification tool analysing coloured Petri net models.

Many efficient techniques and tools exist for the verification of Petri net models. The drawback of the application of CPNs is the lack of efficient verification techniques, what we also faced in our research. The reason for that is twofold: by choosing decision diagram based techniques, one can efficiently represent the state space of Petri net models, but complex guard expressions are not efficient to be encoded in decision diagrams. On the other side, techniques based on advanced solver technologies such as SAT and SMT being able to handle intricate guard expressions efficiently are not good at verifying concurrent systems. These facts lead to the situation that coloured Petri net based symbolic verification tools are not available. The most commonly used tool for verifying CPN models is CPNTools [JKW07], which provides techniques based on explicit state traversal and representation. Such explicit techniques rarely scale to real-life problems.

3.3 Formal Requirements

From the wide range of requirement languages, formal verification relies on formal languages such as CTL or LTL: these are the most widely used formal specification languages being able to express many kinds of properties of interest. CTL model checking is reduced to compute greatest and least fixed points of the next state functions in the state space iteratively. This approach is called structural model checking and decision diagram based symbolic approaches efficiently solve the problem of traversing and storing the possible states. On the other side, LTL model checking is reduced to solving language containment problem. Model checking LTL properties is usually composed of two challenges: one must compute the synchronous product of the state space and the automaton model of the desired property, then look for counterexamples that is reduced to finding strongly connected components (SCCs) in the state space of the product. Symbolic model checking approaches exist for the SCC computation. However, the efficient construction of the synchronous product is still an open question, especially when using saturation-based algorithms. Related work in this field uses traditional binary decision diagram encoding of the composite state space and encodes the synchronous next state relation in a big, monolithic decision diagram. Saturation-based approaches avoid the explicit computation of the synchronous product [Thi15] by dividing the iteration order into smaller parts. This approach might break the iteration strategy of saturation, which may decrease the efficiency. Synchronous product computation is not yet integrated into saturation-based traversal for LTL model checking in a way that it would fully exploit the efficiency of the iteration strategy.

3.4 Objectives

My goal is to introduce a model checking approach for the verification of Petri net based models of complex systems. The proposed approach supports the formal modelling by providing coloured Petri nets as a high-level formalism to represent the system. Temporal logics such as CTL and LTL supports the development of formal specifications. Saturation is used to explore the state space and the next-state function and stores them symbolically. Temporal logic model checking processes this representation of the possible behaviours and evaluates the CTL or LTL specification. The result of the procedure is an error trace to show the problem in the system. Otherwise, the model of the system is assumed to be correct. In the following section, I summarise the challenges concerning the individual steps of the verification process.

3.4.1 Summarizing the Challenges

Challenge 1: Verification of complex systems High-level modelling languages are needed to model complex systems. High-level models of complex systems require rigorous verification techniques, so the existing verification approaches and algorithms have to be extended to overcome the challenges.

Saturation was introduced for the analysis of Petri-nets and their variants/simple extensions. However, in practice, higher level languages provide a better means to describe complex, real-life systems. Coloured Petri nets are a popular formalism, but saturation-based algorithms have not yet been extended for their analysis. Complex data structures of Coloured Petri nets have prevented the application of efficient saturation-based symbolic model checking algorithms in this field.

Challenge 2: Increase the efficiency of model checking algorithms New techniques are needed to increase the efficiency of model checking algorithms and decrease runtime requirements.

Parallelization is a common approach to improve the performance of algorithms. However, saturation is inherently sequential, so it is difficult to parallelise [CZJ09]. The reason behind is the fact that saturation heavily relies on the results of former computations. Indeed, the parallel manipulation of a decision diagram is a difficult problem on its own, which is the prerequisite to develop parallel saturation-based algorithms. Exploit the computational power of modern multi-core computers in saturation-based algorithms is a huge challenge.

Challenge 3. Verification support for various requirements Research and industrial case-studies revealed the need for a wide range of specification languages to support the various types of requirements of the use-cases.

CTL and LTL temporal logics have different strength and weaknesses, so it is important for a model checker to support both formalisms from the usability point of view. Saturation was traditionally used for CTL model checking as the traditional approaches for LTL model checking are difficult to implement in symbolic settings. LTL model checking is reduced to automata based model checking, and saturation-based model checking approaches have to be extended to support automata based formal specifications. Automaton based specification provide the semantics for high-level specification languages such as LTL. LTL model checking requires solving an additional problem during the state space generation: namely the synchronous product computation with an automata representation of the property under analysis. This problem is easy to solve by explicit state space traversal algorithms, but intricate synchronisation constraints often prevent the efficient application of symbolic methods. This lead to that former saturation-based synchronous product generation approaches do not compute synchronisation constraints symbolically instead they try to divide the problem into locally solvable parts. However, this breaks the iteration might degrade it to a breadth-first like iteration.

Challenge 4: Tool support for formal modelling and verification. The wide range of industrial problems necessitates a formal modelling and verification framework with various modelling languages and verification algorithms. As no single formalism or algorithm can support the many aspects of the use-cases, a configurable framework is needed, which can be fine-tuned to handle the verification problems.

Therefore the goal of the dissertation was to define a framework addressing these challenges and develop the necessary algorithms for supporting the verification procedure.

Beside the algorithmic developments, there is a need for tool support for the envisioned verification framework. This involves modelling, verification and counterexample generation of complex systems. There has not been any tool yet for combining the aforementioned algorithms together in one tool to the efficient support of verification of various Petri net based models.

4 New Results

The goal of my work was to provide a comprehensive approach covering all phases of the verification process.

The aforementioned challenges belong to three aspects of the verification problem according to [BKL08]:

1. Modelling of complex systems.
2. Specifying formal requirements.
3. Verifying the model with regard to the requirements.

The goal of my research was to introduce a framework supporting the main tasks in verification. In this section the proposed verification approach and the corresponding subtasks are introduced. In Section 4.1 I overview the modelling and verification approach used during my work, then I introduce my theoretical, algorithmic and engineering contributions.

4.1 Modelling and Verification Approach

In this section I propose a modelling and verification approach. This new approach was designed according to lessons learnt from the research and industrial projects and case-studies of the research group. The introduced approach targets a certain class of problems: the modelling and verification of asynchronous, safety-critical systems with control and finite data.

Problem. Formal modelling necessitates a proper modelling formalism, which is able to capture the problem of the domain. Our experience showed that there is no comprehensive tool and approach which would support the modelling and analysis of asynchronous systems. Petri net based modelling languages provide modelling means for asynchronous systems. However, either a tool has an expressive formalism, but weak analysis such as [JKW07], or it provides efficient analysis techniques but difficult to use [Cia+03].

Formal verification requires expressive specification languages to be able to capture the intent of the designers. This is often a problem, as tools either support reachability checking, or CTL or LTL, but not all of them.

Verification tools supporting Petri net formalisms tend to use only one technique for verification. However, one technique is rarely enough to analyse all aspects of a system. There exists a framework using a wide range of techniques for the verification of synchronous hardware or software systems [Cav+14], but these techniques are not efficient for asynchronous systems [2]. LTSmin is another framework for process algebra, timed automata and extended state machines and it supports various symbolic and explicit techniques [Kan+15]. However, LTSmin does not support Petri nets as a modelling language, and it lacks those techniques from the literature that are efficient Petri net based models.

Summarizing the problems, we need a tool to support all aspects of the verification problem and an approach to support the verification of asynchronous, distributed safety-critical systems.

Goal. As it was discussed, ordinary Petri nets and Coloured Petri nets efficiently capture the behaviour of asynchronous, distributed safety-critical systems, so I propose to combine efficient model checking algorithms from the literature to provide LTL and CTL model checking for Petri net based models.

The overall goal is to provide a tooling for verification engineers. The approach targets verification engineers who aim to develop formal models and execute verification tasks. In order to cover the tasks arising during the verification of complex systems, three main functionalities have to be provided by the framework:

- Editor and persistence support for designing formal models in the Petri net and Coloured Petri net formalism,
- Specifying the formal requirements with CTL and LTL temporal logics,
- Model checking of the formal models if they satisfy the temporal logic requirements.

The goal is to provide modelling and verification support tailored to not a specific domain, but for a wide range of problems, which can be naturally captured by Petri net based models. The target problem domain of the framework is **asynchronous**, **concurrent** or **distributed systems** with data dependence.

Proposed approach. I propose a modelling and analysis approach which combines the expressive power of Petri net based models with the efficiency of saturation and abstraction based algorithms. The approach supports widely-used specification languages such as LTL and CTL.

I propose the verification workflow depicted on Figure 4 consisting of various methods to cover the main aspects of designing and analysing formal models. As verification is a complex task, a wide-range of algorithms is available, and the goal of the workflow is to combine the advantages of these techniques.

According to the literature, I propose to use, integrate and extend the following algorithms into a framework to provide formal modelling and verification support for the engineers:

- Saturation-based algorithms for the efficient state space exploration of PN and CPN models of asynchronous systems.
- Saturation-based LTL and CTL model checking algorithms.
- Bounded saturation and abstraction based algorithms such as CEGAR (Counterexample-Guided Abstraction Refinement) extends the verification capabilities of the framework to handle finite and infinite state models.
- Bounded saturation and abstraction based algorithms are used to generate counterexamples for safety properties.

Figure 4 depicts the verification workflow starting with the formal modelling step and the development of the formal specification. The goal of the framework is to provide Petri net based modelling languages such as Petri nets and Coloured Petri nets and also temporal logic-based specification languages such as CTL and LTL.

The proposed (symbolic) analysis methods need to encode the state space and also the next-state function symbolically. Kronecker matrix based representations are used for PN models, and I propose special algorithms for the handling of CPN models, these algorithms are the disjunctive-conjunctive and the lazy decomposition algorithms.

Various algorithms can be used to explore the state space: beside traditional saturation algorithm, bounded saturation can help verifying models with infinite state space and parallel saturation can exploit the computation power of recent multicore computers. In addition, if the model has infinite state space (for example when representing parametric systems), Counterexample-Guided Abstraction Refinement solves the safety verification problem efficiently. CEGAR can also be used efficiently in other cases of safety verification.

In case of intended LTL model checking, synchronous product generation is needed to compute the product representation of the state space and the property automaton.

Finally, we need temporal logic model checking algorithms in the framework to verify CTL and LTL properties.

Extending the state-of-the-art. In this section, I summarise the work I was involved in, and I will discuss how we advanced the state-of-the-art.

The proposed approach uses the state space exploration algorithm for Petri net and Coloured Petri net models, which is based on [CMS03; CY05]. An efficient structural CTL model checking approach of [CS03; ZC09] is integrated into the framework.

The idea of handling models with infinite state space continues the work of Ciardo et al. [Cia+03; YCL09]. The combination of counterexample generation with saturation follows also this line of research. However, beside the existing techniques, our research group extended the set of bounded state space exploration algorithms and introduced new bounded model checking algorithms. In addition, we utilised also a CEGAR approach of [WW11] to handle a new set of problems and also to provide efficient trace generation.

I propose to integrate saturation-based bounded model checking with structural model checking approaches.

The LTL model checking approach continues the results in this field of [Thi15; Wan+01; Dur+11; CGH97; STV05; Ger+95] and the goal is to exploit and combine their strengths, such as: on-the-fly LTL model checking, abstraction techniques specialised for LTL model checking, synchronous product generation and saturation.

However, some of these algorithms did not exist before I started to devise the approach. In addition, their integration into an efficient analysis framework was also far from trivial.

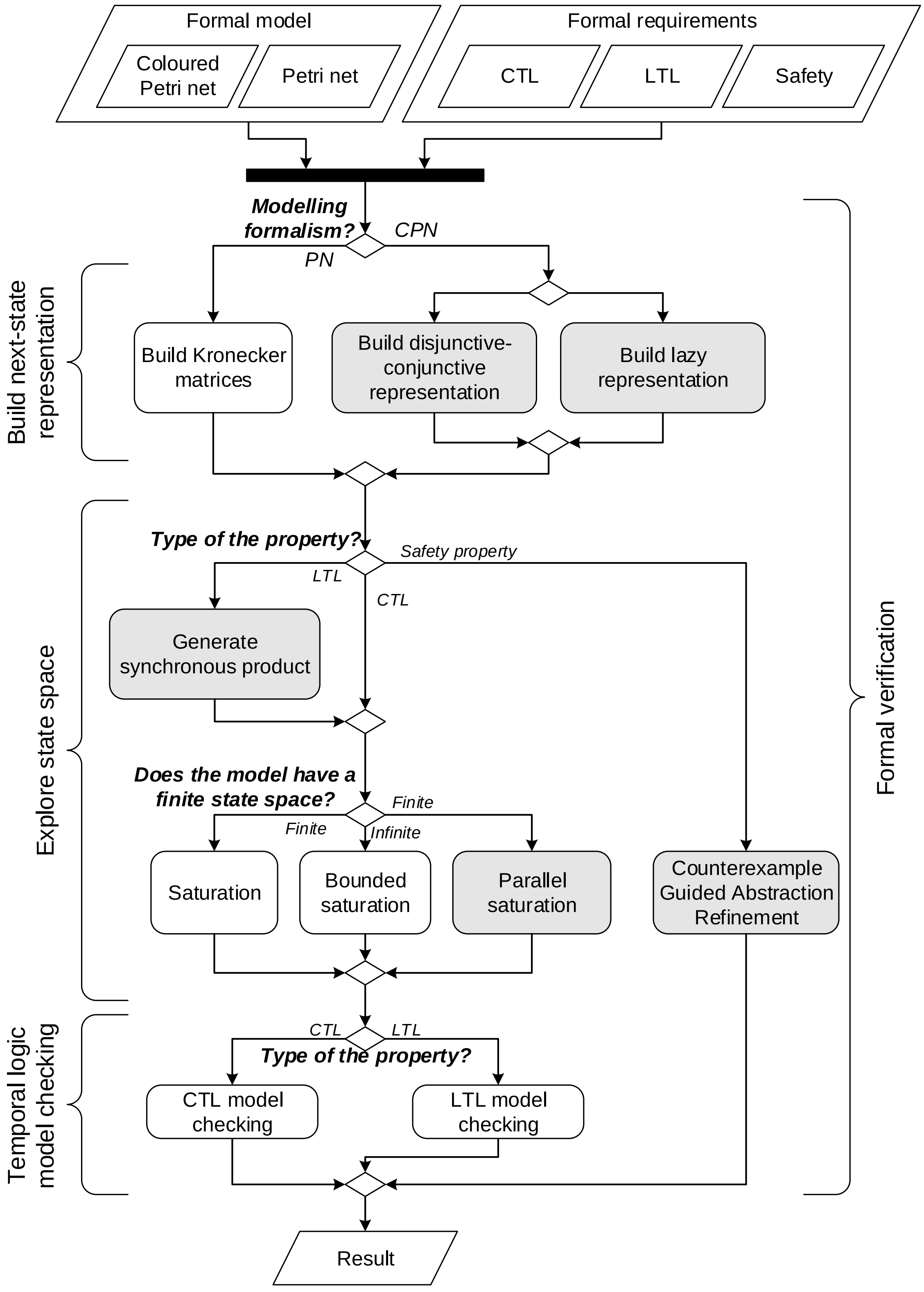

Figure 4: High-level view of the proposed verification approach

During the development, we aimed to extend the existing approaches with new algorithms to fill the research gaps. In the following the extensions are summarised, which had to be contributed to provide a flexible and configurable model checking process:

- Bounded saturation-based CTL model checking was presented in [3],[6],[14] and [18] in order to be able to efficiently combine bounded saturation-based state space traversal of [YCL09] with structural model checking [CS03; ZC09].
- SCC computation and efficient on-the-fly LTL model checking based on saturation presented in [11].
- New CEGAR algorithms in [4], [10] and [15] to extend the solvable set of reachability problems.

My contribution. Beside the complex approach that was put together by the extensive work of our research team, my contributions also significantly extended the applicability of the approach. The model checking framework was introduced in [1] and [7]. The verification process of the framework supports the verification of a wide set of problems, with the help of multiple combinations of algorithms. Beside the whole approach, Figure 4 highlights the steps improved by my work of this dissertation with a grey background.

My algorithmic contributions to the model checking approach are summarised as follows:

- New model checking algorithms for Coloured Petri net models presented in [5].
- New parallel saturation algorithm presented in [17].
- A new synchronous product generation algorithm, presented in [20] in order to provide saturation-based LTL model checking [2].
- Theoretical investigation of the Petri net CEGAR approach in [4] and [15].

4.2 Model Checking of High-level Models

I used the Coloured Petri net formalism to develop formal models of complex systems. I have examined various verification approaches being able to analyse systems designed in high-level modelling languages, especially Coloured Petri nets. I investigated an industrial case study used as a motivation example, which revealed the shortcomings of explicit state model checking techniques: due to state space explosion, they can rarely handle the state space of real-life problems. Symbolic model checking algorithms provide a solution, and from the available approaches, I chose saturation as an extremely powerful method for the verification of Petri nets. However, systematically reviewing the literature I realised that saturation was not extended to handle Coloured Petri nets. I elaborated an approach to support the verification of high level Coloured Petri net based models. The existing algorithms can not handle complex guard expressions of Colured Petri nets, so I developed a new encoding of the next-state relation and I introduced efficient algorithms for the construction of the symbolic representation. The result of the research was integrated into the PetriDotNet model checking framework and proved its efficiency in an industrial setting.

Thesis 1 *I developed new verification algorithms for Coloured Petri nets. I devised an advanced disjunctive-conjunctive decomposition algorithm for the efficient representation of complex next-state relations of coloured Petri net models. The introduced new decomposition algorithm combined with the efficiency of saturation made the verification of even industrial problems possible. In addition, I developed an algorithm for the temporal decomposition of the construction of the complex* *next-state relations. This new algorithm further decreased the space requirements and runtime of the verification of models with complex guard expressions. I proved the correctness of the presented algorithms.*

The results of my first thesis decreased the space requirements of handling complex next-state relations by constructing smaller next-state representations for Coloured Petri nets. As a consequence, the time requirements of the verification process also decreased, and a new set of problems could be verified: the result of the thesis made possible to solve even real-life industrial examples. The new algorithms are evaluated on a model of an industrial safety-critical system (PRISE): it was the first time when the verification of the correctness properties could be verified on the entire Coloured Petri net model of the safety-logic. Successful verification proved the correctness of the system with regard to deadlock freedom, safety and liveness properties.

Publications: My new results introduced in this thesis were published in the journal paper [5] and in the following conference papers: [22] and [16]. The results contributed to the conference paper [7] and journal paper [1].

4.3 Parallel State Space Exploration Techniques

I investigated various techniques to speed-up the model checking algorithms. Extending existing algorithms to exploit the computational power of modern multicore computers necessitates the construction of rigorous parallel algorithms and synchronisation mechanisms. Extending symbolic algorithms to run parallel is especially challenging due to the complex data structures and intricate symbolic computations. Saturation uses a special iteration strategy for traversing and building the symbolic representation of both the state space and the next-state relation in an incremental manner, which means that the steps heavily rely on the results of the former computations. This makes the parallel implementation a challenging task. I investigated the existing parallel saturation algorithm, and I identified some points where it could be improved. I introduced a new synchronisation mechanism which reduces the synchronisation overhead and it could significantly speed up the model checking algorithm. The developed parallel algorithm could exploit the computational power of modern multi-core processor computers in saturation-based state space exploration.

Thesis 2 *I developed a parallel saturation based state space traversal algorithm using a novel synchronisation method and locking strategy. The new locking strategy applies a fine-grained locking mechanism, which only synchronises the manipulation of the state space representation. The algorithm prevents the occurrence of inconsistent states and ensures the correct execution of the saturation iteration order. The new synchronisation algorithm decreased the synchronisation overhead and led to increased parallelism. The new parallel algorithm can exploit the computational power of modern multicore computers by decreasing the synchronisation overhead – for certain benchmark models – significantly. I proved the correctness of the new parallel algorithm.*

Various measurements showed the competitiveness of the new algorithm on benchmark models. The new algorithm scales with the growing number of computation units better than the former approaches. The new synchronisation algorithm significantly reduced the synchronisation overhead, and it could lead to significant performance gain compared to former approaches.

Publications: My new results introduced in this thesis were published in the conference paper [17].

4.4 On-the-fly Synchronous Product Generation for Model Checking Regular Safety Properties

Users have to analyse various kinds of properties, which can be expressed with the help of temporal logics. CTL and LTL are widely used temporal logics and they have different expressive power. Deadlock-freedom is expressible in CTL while fairness properties are supported by LTL. To support the engineers in verification, it is suggested to provide verification for both specification languages. I investigated the literature and efficient saturation based algorithms exist for the structural model checking of CTL properties. However, LTL model checking lacks the verification support based on efficient symbolic algorithms. In this thesis, I focus on a significant subset of the LTL language, namely regular safety properties. Model checking regular safety properties can be traced back to two main problems: synchronous product generation and detection of accepting states. Synchronous product generation is a difficult problem in a symbolic setting where one has to encode the property automaton and has to synchronise the steps with the state space. This is a difficult problem as saturation traverses the states in an irregular order which makes the synchronous product computation extremely difficult. I propose an efficient technique to compute synchronous product on the fly during the state space exploration and model checking of safety regular properties. The goal of the approach is to enable on-the-fly model checking during the state space traversal.

Thesis 3 *I developed a saturation based model checking algorithm for the safety regular subset of LTL properties. I propose a symbolic encoding of the automaton, and I introduce a new symbolic constraint to the saturation algorithm. I also introduce a new state space traversal technique to compute synchronous product on the fly during the state space traversal and do on-the-fly LTL model checking. The new algorithm served as the foundation of a new saturation-based LTL model checking procedure.*

My solution is the first algorithm which provides verification for a rich set of specification languages based on the saturation algorithm. The new algorithm extends the set of systems and requirements which could be verified by saturation. Various measurements showed the competitiveness of the new algorithm on benchmark models. In addition, the LTL model checker based on the new synchronisation algorithm was the first which could verify LTL properties of the PRISE industrial case study.

Publications: My new results introduced in this thesis were published in the paper [20]. The results contributed to the conference paper [7] and journal papers [2] and [1].

4.5 PetriDotNet Model Checking Framework

The usability of the developed algorithms cannot be achieved without tool support. I have investigated many existing tools and approaches and evaluated them amongst others on the industrial problem of the PRISE safety system. According to the experiences, I chose to use Petri nets as a simple formalism and Coloured Petri nets as a convenient formalism to develop high-level models. However, the available tool support for the verification of Petri nets was weak in the sense that either good editor and tooling were available or advanced verification algorithms. However, the verification problem is difficult in general so no single algorithm or approach can be efficient on their own. Therefore I developed a model checking workflow to combine the various advantages of the different algorithms and approaches. The novel combination of the algorithms was implemented in the PetriDotNet model checking framework. Theoretical examination of the algorithms was needed in order to extend them and combine their strength in a framework. The proposed model checking framework addressed the problem of modelling safety-critical systems with high-level modelling languages, specifying the requirements with the help of temporal logics and verify finite state and infinite state models with various algorithms.

Thesis 4 *I worked out an approach for the modelling and verification of complex systems. We developed a framework to support the Coloured Petri net based modelling and verification of complex systems. The framework provides CTL and LTL model checking based on novel algorithms. In order to extend the handled classes of models, infinite state and trace generation algorithms were integrated. We extended the model checking algorithms to be able to handle infinite state systems by applying bounded model checking and a special algorithm based on Counterexample-Guided Abstraction Refinement (CEGAR). The latter algorithm provides traces as a feedback for the developers. I did theoretical investigations, and I examined the CEGAR algorithm from the completeness point of view: I proved the incompleteness of the CEGAR-based Petri net reachability algorithm.*

This thesis encapsulates the various results together into a framework supporting the engineers developing correct systems. A theoretical analysis was elaborated on a well-known algorithm, and its applicability for trace generation was examined. This opens new directions for further developments in the future[10]. I envisioned and designed the PetriDotNet model checking framework where we could successfully integrate the research results of the participants of the research group and students supervised by myself.

Publications: My new results introduced in this thesis were published in the journal papers [4] and [1] and in the following conference papers: [15] and [7].

5 Applications of New Results

In this section, the practical applications of the results of the dissertation are summarised. The researches of our team resulted in the PetriDotNet framework, which is available from the following website:

`https://inf.mit.bme.hu/research/tools/petridotnet`

The PetriDotNet framework was used in many industrial and research projects. Beside the theoretical and practical results of this dissertation, various extensions of the PetriDotNet were built on top of the results of this dissertation.

5.1 Verification of an Industrial Safety Function

The PetriDotNet verification framework was used to design the formal model of a safety function of the Paks Nuclear Power Plant. The formal model was developed as a Coloured Petri net model. The algorithms of thesis 1. were used to verify the correctness of the safety function. CTL specification captured the requirements, and the introduced disjunctive-conjunctive decomposition method was the first algorithm being able to explore the full state space of the model. In addition to the general correctness requirements, we have developed the correctness criteria also in the regular language and also LTL (and some additional criteria, which were not expressible in CTL). With the help of the synchronous product generation algorithm, the LTL requirements could be analysed and the approach of thesis 3. was the first solution being able to provide LTL model checking of the PRISE safety function.

Summary. The experiments with the verification of the PRISE safety logic have demonstrated that (1) CPNs are useful for modelling industrial modules, (2) the PetriDotNet tool is capable of modelling industrial modules, and (3) the novel saturation-based CTL model checking algorithms for CPNs make the verification of complex, industrial modules feasible.

5.2 Usage of the PetriDotNet Framework in Practice

Various other scenarios and applications of PetriDotNet are known, in the following the most important case-studies are listed.

Robustness test generation for autonomous robot systems. High-level modelling languages turned to be useful for test generation in the R3-COP[1] project, where PetriDotNet was used to generate test input sequences for testing the robustness of communicating robots. This case-study underlines the overall applicability of the framework of thesis 4. With the help of the disjunctive-conjunctive decomposition introduced in thesis 1., and by using the various bounded saturation algorithms, we were able to generate a diverse set of tests fulfilling various robustness test goals.

Modelling and analysis of public transportation networks. In a project urban public transport networks were modelled and analysed using Petri nets. The PetriDotNet framework could support the modelling and analysis such as described in thesis 4. The goal in this case-study was to analyse if a configuration of the transportation network with a given transportation capacity can fulfil the travellers need. Due to the fact that the system was unbounded, the CEGAR algorithm was used for the analysis. Infinite state Petri nets were used for modelling and reachability checking was applied to analyse the behaviour of the system.

Modelling railway interlocking systems. PetriDotNet was applied to the model and study railway interlocking systems by a group of domain experts. The models were developed manually by using the Petri net formalism, and the correctness criteria were expressed with the help of the CTL language. These requirements included the following examples: "A signal can only be cleared (set to a state permitting train movements) if all corresponding points are set and locked in correct position." or "While a signal is cleared, the corresponding points cannot be released or set to a different direction." These requirements were verified using the saturation-based CTL model checking, included in PetriDotNet. This case study shows the usability of the PetriDotNet framework as the domain experts were not the member of the developer team. This further emphasizes the applicability of thesis 4.

Model Checking Contest. Model Checking Contest is an annual competition to compare Petri net based model checking tools. We have made extensive measurements to compare the LTL model checking algorithm integrated into PetriDotNet and state-of-the-art model checkers. The LTL model checking algorithm of PetriDotNet extends the algorithm introduced in thesis 3 with a new automata representation and also with on-the-fly SCC detection. As a result, the comparison showed that PetriDotNet is superior compared to the other tools in LTL model checking.

[1]http://www.r3-cop.eu/

6 Publication list

Journal papers

[1] A. Vörös, D. Darvas, Á. Hajdu, A. Klenik, K. Marussy, V. Molnár, T. Bartha, and I. Majzik. "Industrial Applications of the PetriDotNet Modelling and Analysis Tool". In: *Science of Computer Programming* (2017). In press. ISSN: 0167-6423. DOI: 10.1016/j.scico.2017.09.003

[2] V. Molnár, A. Vörös, D. Darvas, T. Bartha, and I. Majzik. "Component-wise Incremental LTL Model Checking". In: *Formal Aspects of Computing* 28.3 (2016), pp. 345–379. ISSN: 0934-5043. DOI: 10.1007/s00165-015-0347-x

[3] D. Darvas, A. Vörös, and T. Bartha. "Improving Saturation-based Bounded Model Checking". In: *Acta Cybernetica* 22.3 (2016), pp. 573–589. ISSN: 0324-721X. DOI: 10.14232/actacyb.22.3.2016.2

[4] Á. Hajdu, A. Vörös, T. Bartha, and Z. Mártonka. "Extensions to the CEGAR Approach on Petri Nets". In: *Acta Cybernetica* 21.3 (2014), pp. 401–417. DOI: 10.14232/actacyb.21.3.2014.8

[5] A. Vörös, D. Darvas, A. Jámbor, and T. Bartha. "Advanced Saturation-based Model Checking of Well-formed Coloured Petri Nets". In: *Periodica Polytechnica, Electrical Engineering and Computer Science* 58.1 (2014), pp. 3–13. ISSN: 2064-5279. DOI: 10.3311/PPee.2080

[6] A. Vörös, D. Darvas, and T. Bartha. "Bounded saturation-based CTL model checking". In: *Proceedings of the Estonian Academy of Sciences* 62.1 (2013), pp. 59–70. ISSN: 1736-6046. DOI: 10.3176/proc.2013.1.07

International conference and workshop papers

[7] A. Vörös, D. Darvas, V. Molnár, A. Klenik, Á. Hajdu, A. Jámbor, T. Bartha, and I. Majzik. "PetriDotNet 1.5: Extensible Petri Net Editor and Analyser for Education and Research". In: *Application and Theory of Petri Nets and Concurrency*. Ed. by F. Kordon and D. Moldt. Vol. 9698. Lecture Notes in Computer Science. Springer, 2016, pp. 123–132. ISBN: 978-3-319-39086-4. DOI: 10.1007/978-3-319-39086-4_9

[8] K. Marussy, A. Klenik, V. Molnár, A. Vörös, I. Majzik, and M. Telek. "Efficient decomposition algorithm for stationary analysis of complex stochastic Petri net models". In: *Application and Theory of Petri Nets and Concurrency*. Ed. by F. Kordon and D. Moldt. Vol. 9698. Lecture Notes in Computer Science. Springer, 2016, pp. 281–300. ISBN: 978-3-319-39086-4. DOI: 10.1007/978-3-319-39086-4_17

[9] K. Marussy, A. Klenik, V. Molnár, A. Vörös, M. Telek, and I. Majzik. "Configurable Numerical Analysis for Stochastic Systems". In: *Proceedings of the 2016 Workshop on Symbolic and Numerical Methods for Reachability Analysis (SNR)*. ed. by E. Ábrahám and S. Bogomolov. Vienna, Austria: IEEE, 2016. ISBN: 978-1-5090-3079-8. DOI: 10.1109/SNR.2016.7479383

[10] Á. Hajdu, A. Vörös, and T. Bartha. "New search strategies for the Petri net CEGAR approach". In: *Application and Theory of Petri Nets and Concurrency*. Ed. by R. Devillers and A. Valmari. Vol. 9115. Lecture Notes in Computer Science. Springer, 2015, pp. 309–328. ISBN: 978-3-319-19488-2. DOI: 10.1007/978-3-319-19488-2_16

[11] V. Molnár, D. Darvas, A. Vörös, and T. Bartha. "Saturation-Based Incremental LTL Model Checking with Inductive Proofs". In: *Tools and Algorithms for the Construction and Analysis of Systems.* Ed. by C. Baier and C. Tinelli. Vol. 9035. Lecture Notes in Computer Science. Springer, 2015, pp. 643–657. ISBN: 978-3-662-46680-3. DOI: 10.1007/978-3-662-46681-0_58

[12] D. Darvas, B. Fernández Adiego, A. Vörös, T. Bartha, E. Blanco Viñuela, and V. M. González Suárez. "Formal verification of complex properties on PLC programs". In: *Formal Techniques for Distributed Objects, Components, and Systems.* Ed. by E. Ábrahám and C. Palamidessi. Vol. 8461. Lecture Notes in Computer Science. Springer, 2014, pp. 284–299. ISBN: 978-3-662-43612-7. DOI: 10.1007/978-3-662-43613-4_18

[13] Z. Micskei, R.-A. Konnerth, B. Horváth, O. Semeráth, A. Vörös, and D. Varró. "On Open Source Tools for Behavioral Modeling and Analysis with fUML and Alf". In: *Proceedings of the 1st Workshop on Open Source Software for Model Driven Engineering.* Ed. by F. Bordelau, J. Dingel, S. Gerard, and S. Voss. Valencia, Spain, Sept. 2014, pp. 31–41

[14] D. Darvas, A. Vörös, and T. Bartha. "Efficient Saturation-based Bounded Model Checking of Asynchronous Systems". In: *Proceedings of the 13th Symposium on Programming Languages and Software Tools, SPLST'13.* Ed. by Á. Kiss. Szeged, Hungary: University of Szeged, 2013, pp. 259–273. ISBN: 978-963-306-228-9

[15] Á. Hajdu, A. Vörös, T. Bartha, and Z. Mártonka. "Extensions to the CEGAR Approach on Petri Nets". In: *Proceedings of the 13th Symposium on Programming Languages and Software Tools, SPLST'13.* Ed. by Á. Kiss. Szeged, Hungary: University of Szeged, 2013, pp. 274–288. ISBN: 978-963-306-228-9

[16] T. Bartha, A. Vörös, A. Jámbor, and D. 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).* Ed. by E. Ilie-Zudor, Z. Kemény, and L. Monostori. Budapest, Hungary: Hungarian Academy of Sciences, Computer and Automation Research Institute, 2012, pp. 472–485. ISBN: 978-963-311-373-8

[17] A. Vörös, T. Bartha, D. Darvas, T. Szabó, A. Jámbor, and Á. Horváth. "Parallel Saturation Based Model Checking". In: *Proceedings of the 10th International Symposium on Parallel and Distributed Computing (ISPDC).* Cluj Napoca, Romania: IEEE Computer Society, 2011, pp. 94–101. ISBN: 978-1-4577-1536-5. DOI: 10.1109/ISPDC.2011.23

[18] A. Vörös, D. Darvas, and T. Bartha. "Bounded Saturation Based CTL Model Checking". In: *Proceedings of the 12th Symposium on Programming Languages and Software Tools, SPLST'11.* Ed. by J. Penjam. Tallinn, Estonia: Tallinn University of Technology, Institute of Cybernetics, 2011, pp. 149–160. ISBN: 978-9949-23-178-2

Local conference and workshop papers

[19] Á. Hajdu, R. Német, S. Varró-Gyapay, and A. Vörös. "Petri Net Based Trajectory Optimization". In: *ASCONIKK 2014: Extended Abstracts. Future Internet Services.* Veszprém, Hungary: University of Pannonia, 2014, pp. 11–19

[20] V. Molnár and A. Vörös. "Synchronous Product Automaton Generation for Controller Optimization". In: *ASCONIKK 2014: Extended Abstracts. I. Information Technologies for Logistic*

Systems. Veszprém, Hungary: University of Pannonia, 2014, pp. 22–29. ISBN: 978-963-396-046-2

[21] D. Darvas and A. Vörös. "Szaturációalapú tesztbemenet-generálás színezett Petri-hálókkal [in Hungarian]". In: *Mesterpróba 2013. Konferenciakiadvány.* Budapest, Hungary, 2013, pp. 48–51

[22] A. Vörös. "Modellellenőrzés alkalmazása egy biztonságkritikus rendszer védelmi logikájának verifikációjára [in Hungarian]". In: *XVII. Fiatal Műszakiak Tudományos Ülésszaka.* Cluj Napoca, Romania: Erdélyi Múzeum-Egyesület Műszaki Tudományok Szakosztálya, 2012, pp. 383–386

[23] A. Vörös. "Forward Saturation Based Model Checking". In: *Proceedings of the 19th PhD Minisymposium of the Department of Measurement and Information Systems.* Budapest, Hungary, 2012, pp. 38–41

[24] A. Vörös. "Optimizing Saturation Based Model Checking". In: *Proceedings of the 18th PhD Minisymposium of the Department of Measurement and Information Systems.* Budapest, Hungary, 2011, pp. 96–99

References

[Adi+15] B. F. Adiego, D. Darvas, E. B. Viñuela, J. C. Tournier, S. Bliudze, J. O. Blech, and V. M. G. Suárez. "Applying Model Checking to Industrial-Sized PLC Programs". In: *IEEE Transactions on Industrial Informatics* 11.6 (Dec. 2015), pp. 1400–1410. ISSN: 1551-3203. DOI: 10.1109/TII.2015.2489184.

[Ana+13] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, and P. McMinn. "An orchestrated survey of methodologies for automated software test case generation". In: *Journal of Systems and Software* 86.8 (2013), pp. 1978–2001. ISSN: 0164-1212. DOI: 10.1016/j.jss.2013.02.061.

[Bey17] D. Beyer. "Software Verification with Validation of Results". In: *Tools and Algorithms for the Construction and Analysis of Systems.* Ed. by A. Legay and T. Margaria. Berlin, Heidelberg: Springer, 2017, pp. 331–349. ISBN: 978-3-662-54580-5. DOI: 10.1007/978-3-662-54580-5_20.

[BKL08] C. Baier, J.-P. Katoen, and K. G. Larsen. *Principles of model checking.* MIT press, 2008.

[BKP17] H. Bride, O. Kouchnarenko, and F. Peureux. "Reduction of Workflow Nets for Generalised Soundness Verification". In: *Verification, Model Checking, and Abstract Interpretation: 18th International Conference, VMCAI 2017, Paris, France, January 15–17, 2017, Proceedings.* Ed. by A. Bouajjani and D. Monniaux. Cham: Springer, 2017, pp. 91–111. ISBN: 978-3-319-52234-0. DOI: 10.1007/978-3-319-52234-0_6.

[BP12] D. Beyer and A. K. Petrenko. "Linux Driver Verification". In: *Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies: 5th International Symposium, ISoLA 2012, Heraklion, Crete, Greece, October 15-18, 2012, Proceedings, Part II.* Ed. by T. Margaria and B. Steffen. Berlin, Heidelberg: Springer, 2012, pp. 1–6. ISBN: 978-3-642-34032-1. DOI: 10.1007/978-3-642-34032-1_1.

[Cal+15] C. Calcagno, D. Distefano, J. Dubreil, D. Gabi, P. Hooimeijer, M. Luca, P. O'Hearn, I. Papakonstantinou, J. Purbrick, and D. Rodriguez. "Moving Fast with Software Verification". In: *NASA Formal Methods: 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings*. Ed. by K. Havelund, G. Holzmann, and R. Joshi. Cham: Springer, 2015, pp. 3–11. ISBN: 978-3-319-17524-9. DOI: 10.1007/978-3-319-17524-9_1.

[Cav+14] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta. "The nuXmv Symbolic Model Checker". In: *Computer-Aided Verification*. Ed. by A. Biere and R. Bloem. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 334–342. ISBN: 978-3-319-08866-2.

[CES86] E. Clarke, E. A. Emerson, and A. P. Sistla. "Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications". In: *ACM Trans. Program. Lang. Syst.* 8.2 (Apr. 1986), pp. 244–263. ISSN: 0164-0925. DOI: 10.1145/5397.5399.

[CGH97] E. Clarke, O. Grumberg, and K. Hamaguchi. "Another Look at LTL Model Checking". In: *Formal Methods in System Design* 10.1 (1997), pp. 47–71. ISSN: 0925-9856. DOI: 10.1023/A:1008615614281.

[Cia+03] G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceanu. "Logical and stochastic modeling with SMART". In: *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*. Springer. 2003, pp. 78–97.

[CMS03] G. Ciardo, R. Marmorstein, and R. Siminiceanu. "Saturation Unbound". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2003, pp. 379–393.

[CMS05] G. Ciardo, R. Marmorstein, and R. Siminiceanu. "The saturation algorithm for symbolic state-space exploration". In: *International Journal on Software Tools for Technology Transfer* 8.1 (Nov. 2005), p. 4. ISSN: 1433-2787. DOI: 10.1007/s10009-005-0188-7.

[CS03] G. Ciardo and R. Siminiceanu. "Structural symbolic CTL model checking of asynchronous systems". In: *Computer-Aided Verification*. Vol. 3. Springer. 2003, pp. 40–53.

[Cse+02] G. Csertan, G. Huszerl, I. Majzik, Z. Pap, A. Pataricza, and D. Varro. "VIATRA - visual automated transformations for formal verification and validation of UML models". In: *Proceedings 17th IEEE International Conference on Automated Software Engineering*, 2002, pp. 267–270. DOI: 10.1109/ASE.2002.1115027.

[CY05] G. Ciardo and A. Yu. "Saturation-based symbolic reachability analysis using conjunctive and disjunctive partitioning". In: *Correct Hardware Design and Verification Methods* 3725 (2005), pp. 146–161.

[CZJ09] G. Ciardo, Y. Zhao, and X. Jin. "Parallel symbolic state-space exploration is difficult, but what is the alternative?" In: *arXiv preprint arXiv:0912.2785* (2009).

[Dar17] D. Darvas. "Practice-Oriented Formal Methods to Support the Software Development of Industrial Control Systems". PhD thesis. Budapest University of Technology and Economics, 2017. DOI: 10.5281/zenodo.162950.

[DKW08] V. D'Silva, D. Kroening, and G. Weissenbacher. "A Survey of Automated Techniques for Formal Software Verification". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27.7 (July 2008), pp. 1165–1178. ISSN: 0278-0070. DOI: 10.1109/TCAD.2008.923410.

[DMB16] D. Darvas, I. Majzik, and E. Blanco Viñuela. “Formal Verification of Safety PLC Based Control Software”. In: *Integrated Formal Methods: 12th International Conference, IFM 2016, Reykjavik, Iceland, June 1-5, 2016, Proceedings.* Ed. by E. Ábrahám and M. Huisman. Cham: Springer, 2016, pp. 508–522. ISBN: 978-3-319-33693-0. DOI: 10.1007/978-3-319-33693-0_32.

[Dur+11] A. Duret-Lutz, K. Klai, D. Poitrenaud, and Y. Thierry-Mieg. “Self-Loop Aggregation Product – A New Hybrid Approach to On-the-Fly LTL Model Checking”. In: *Automated Technology for Verification and Analysis.* Vol. 6996. Lecture Notes in Computer Science. Springer, 2011, pp. 336–350. DOI: 10.1007/978-3-642-24372-1_24.

[EN94] J. Esparza and M. Nielsen. “Decidability issues for Petri nets”. In: *BRICS Report Series* 1.8 (1994).

[GA14] S. J. Galler and B. K. Aichernig. “Survey on test data generation tools”. In: *International Journal on Software Tools for Technology Transfer* 16.6 (Nov. 2014), pp. 727–751. ISSN: 1433-2787. DOI: 10.1007/s10009-013-0272-3.

[Ger+95] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. “Simple on-the-fly automatic verification of linear temporal logic”. In: *Proc. of the Int. Symp. on Protocol Specification, Testing and Verification.* Chapman & Hall, Ltd., 1995, pp. 3–18. ISBN: 0-412-71620-8.

[Ins10] Institute of Electrical and Electronics Engineers. “Systems and software engineering – Vocabulary”. In: *ISO/IEC/IEEE 24765:2010(E)* (Dec. 2010), pp. 1–418. DOI: 10.1109/IEEESTD.2010.5733835.

[JKW07] K. Jensen, L. M. Kristensen, and L. Wells. “Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems”. In: *International Journal on Software Tools for Technology Transfer* 9.3 (2007), pp. 213–254. ISSN: 1433-2787. DOI: 10.1007/s10009-007-0038-x.

[Kai+09] R. Kaivola et al. “Replacing Testing with Formal Verification in Intel CoreTM i7 Processor Execution Engine Validation”. In: *Computer-Aided Verification.* Ed. by A. Bouajjani and O. Maler. Berlin, Heidelberg: Springer, 2009, pp. 414–429. ISBN: 978-3-642-02658-4. DOI: 10.1007/978-3-642-02658-4_32.

[Kan+15] G. Kant, A. Laarman, J. Meijer, J. van de Pol, S. Blom, and T. van Dijk. “LTSmin: High-Performance Language-Independent Model Checking”. In: *Tools and Algorithms for the Construction and Analysis of Systems.* Ed. by C. Baier and C. Tinelli. Berlin, Heidelberg: Springer, 2015, pp. 692–707. ISBN: 978-3-662-46681-0. DOI: 10.1007/978-3-662-46681-0_61.

[Kle+09] G. Klein et al. “seL4: Formal Verification of an OS Kernel”. In: *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles.* SOSP ’09. Big Sky, Montana, USA: ACM, 2009, pp. 207–220. ISBN: 978-1-60558-752-3. DOI: 10.1145/1629575.1629596.

[LMM99] D. Latella, I. Majzik, and M. Massink. “Automatic Verification of a Behavioural Subset of UML Statechart Diagrams Using the SPIN Model-checker”. In: *Formal Aspects of Computing* 11.6 (Dec. 1999), pp. 637–664. ISSN: 1433-299X. DOI: 10.1007/s001659970003.

[LS09] D. Leinenbach and T. Santen. “Verifying the Microsoft Hyper-V Hypervisor with VCC”. In: *Proceedings of the 2Nd World Congress on Formal Methods.* FM ’09. Eindhoven, The Netherlands: Springer, 2009, pp. 806–809. ISBN: 978-3-642-05088-6. DOI: 10.1007/978-3-642-05089-3_51.

[Mic13] Z. Micskei. “Languages and frameworks for specifying test artifacts”. PhD thesis. Budapest University of Technology and Economics, 2013.

[MSB11] G. J. Myers, C. Sandler, and T. Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[Mur89] T. Murata. “Petri nets: Properties, analysis and applications”. In: *Proceedings of the IEEE* 77.4 (Apr. 1989), pp. 541–580. ISSN: 0018-9219. DOI: 10.1109/5.24143.

[SD10] W. Steiner and B. Dutertre. “SMT-based Formal Verification of a TTEthernet Synchronization Function”. In: *Proceedings of the 15th International Conference on Formal Methods for Industrial Critical Systems*. FMICS’10. Antwerp, Belgium: Springer, 2010, pp. 148–163. ISBN: 3-642-15897-8, 978-3-642-15897-1.

[STV05] R. Sebastiani, S. Tonetta, and M. Y. Vardi. “Symbolic Systems, Explicit Properties: On Hybrid Approaches for LTL Symbolic Model Checking”. In: *Computer Aided Verification*. Ed. by K. Etessami and S. K. Rajamani. Vol. 3576. Lecture Notes in Computer Science. Springer, 2005, pp. 350–363. ISBN: 978-3-540-27231-1.

[Thi15] Y. Thierry-Mieg. “Symbolic Model-Checking Using ITS-Tools”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C. Baier and C. Tinelli. Berlin, Heidelberg: Springer, 2015, pp. 231–237. ISBN: 978-3-662-46681-0. DOI: 10.1007/978-3-662-46681-0_20.

[Wan+01] C. Wang, R. Bloem, G. D. Hachtel, K. Ravi, and F. Somenzi. “Divide and Compose: SCC Refinement for Language Emptiness”. In: *CONCUR 2001 – Concurrency Theory*. Vol. 2154. Lecture Notes in Computer Scienc. Springer, 2001, pp. 456–471. ISBN: 3-540-42497-0.

[WW11] H. Wimmel and K. Wolf. “Applying CEGAR to the Petri Net State Equation”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by P. A. Abdulla and K. R. M. Leino. Vol. 6605. Lecture Notes in Computer Science. Springer, 2011, pp. 224–238. DOI: 10.1007/978-3-642-19835-9_19.

[YCL09] A. Yu, G. Ciardo, and G. Lüttgen. “Decision-diagram-based techniques for bounded reachability checking of asynchronous systems”. In: *International Journal on Software Tools for Technology Transfer* 11 (2 2009), pp. 117–131. ISSN: 1433-2779. DOI: 10.1007/s10009-009-0099-0.

[ZC09] Y. Zhao and G. Ciardo. “Symbolic CTL Model Checking of Asynchronous Systems Using Constrained Saturation”. In: *Automated Technology for Verification and Analysis*. Vol. 5799. Lecture Notes in Computer Science. Springer, 2009, pp. 368–381. ISBN: 978-3-642-04760-2. DOI: 10.1007/978-3-642-04761-9_27.