THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Safety Proofs for Automated Driving using Formal Methods

Yuvaraj Selvaraj

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2022

Safety Proofs for Automated Driving using Formal Methods

Yuvaraj Selvaraj
ISBN 978-91-7905-738-1

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 5204
ISSN 0346-718X

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000

Printed by Chalmers Reproservice
Gothenburg, Sweden, October 2022

To Yathran

தம்மின் தம்மக்கள் அறிவுடைமை மாநிலத்து
மன்னுயிர்க்கு எல்லாம் இனிது

குறள் 68

Abstract

The introduction of driving automation in road vehicles can potentially reduce road traffic crashes and significantly improve road safety. Automation in road vehicles also brings other benefits such as the possibility to provide independent mobility for people who cannot and/or should not drive. Correctness of such automated driving systems (ADSs) is crucial as incorrect behaviour may have catastrophic consequences.

Automated vehicles operate in complex and dynamic environments, which requires decision-making and control at different levels. The aim of such decision-making is for the vehicle to be safe at all times. Verifying safety of these systems is crucial for the commercial deployment of full autonomy in vehicles. Testing for safety is expensive, impractical, and can never guarantee the absence of errors. In contrast, *formal methods*, techniques that use rigorous mathematical models to build hardware and software systems, can provide mathematical proofs of the correctness of the systems.

The focus of this thesis is to address some of the challenges in the safety verification of decision and control systems for automated driving. A central question here is how to establish formal methods as an efficient approach to develop a safe ADS. A key finding is the need for an integrated formal approach to prove correctness of ADS. Several formal methods to model, specify, and verify ADS are evaluated. Insights into how the evaluated methods differ in various aspects and the challenges in the respective methods are discussed. To help developers and safety experts design safe ADSs, the thesis presents modelling guidelines and methods to identify and address subtle modelling errors that might inadvertently result in proving a faulty design to be safe. To address challenges in the manual modelling process, a systematic approach to automatically obtain formal models from ADS software is presented and validated by a proof of concept. Finally, a structured approach on how to use the different formal artifacts to provide evidence for the safety argument of an ADS is shown.

Keywords: Automated driving, safety argument, formal methods, formal verification, supervisory control theory, model checking, theorem proving, automata learning.

List of Publications

This thesis is based on the following publications:

[A] **Yuvaraj Selvaraj**, Wolfgang Ahrendt, Martin Fabian, "Verification of Decision Making Software in an Autonomous Vehicle: An Industrial Case Study". *Formal Methods for Industrial Critical Systems (FMICS)*. LNCS 11687, pp. 143–159, 2019. DOI: 10.1007/978-3-030-27008-7_9.

[B] **Yuvaraj Selvaraj**, Ashfaq Farooqui, Ghazaleh Panahandeh, Wolfgang Ahrendt, Martin Fabian, "Automatically Learning Formal Models from Autonomous Driving Software". *Electronics*, 11(4), 643, 2022. DOI: 10.3390/electronics11040643.

[C] **Yuvaraj Selvaraj**, Wolfgang Ahrendt, Martin Fabian, "Formal Development of Safe Automated Driving using Differential Dynamic Logic". *IEEE Transactions on Intelligent Vehicles*, 2022. DOI: 10.1109/TIV.2022.3204574.

[D] **Yuvaraj Selvaraj**, Jonas Krook, Wolfgang Ahrendt, Martin Fabian, "On How to Not Prove Faulty Controllers Safe in Differential Dynamic Logic". *International Conference on Formal Engineering Methods (ICFEM)*. LNCS 13478, pp. 281–297, 2022. DOI: 10.1007/978-3-031-17244-1_17.

[E] Jonas Krook, **Yuvaraj Selvaraj**, Wolfgang Ahrendt, Martin Fabian, "A Formal Methods Approach to Provide Evidence in Automated Driving Safety Cases". Submitted to *IEEE Transactions on Intelligent Vehicles*, 2022.

Other publications by the author, not included in this thesis, are:

[F] Rajashekar Chandru, **Yuvaraj Selvaraj**, Mattias Brännström, Roozbeh Kianfar, Nikolce Murgovski, "Safe autonomous lane changes in dense traffic". *IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1-6, 2017. DOI: 10.1109/ITSC.2017.8317590.

[G] **Yuvaraj Selvaraj**, Zhennan Fei, Martin Fabian, "Supervisory Control Theory in System Safety Analysis". *Computer Safety, Reliability, and Security. SAFECOMP Workshops*, Lecture Notes in Computer Science vol. 12235, pp. 9–22, 2020. DOI: 10.1007/978-3-030-55583-2_1.

[H] **Yuvaraj Selvaraj**, Ashfaq Farooqui, Ghazaleh Panahandeh, Martin Fabian, “Automatically Learning Formal Models: An Industrial Case from Autonomous Driving Development”. *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS ’20 Companion)*, 2020. DOI: 10.1145/3417990.3421262.

[I] Tom P. Huck, **Yuvaraj Selvaraj**, Constantin Cronrath, Christoph Ledermann, Martin Fabian, Torsten Kröger, “Hazard Analysis of Collaborative Automation Systems: A Two-layer Approach based on Supervisory Control and Simulation”. Submitted to *International Conference on Robotics and Automation (ICRA 2023)*. Available online at arXiv:2209.12560.

Acknowledgments

My biggest thanks go to my supervisors Martin Fabian and Wolfgang Ahrendt. The interdisciplinary discussions with both of them have brought different perspectives and have immensely helped me shape this work. I hope that someday, the terms Martin-safety and Wolfgang-safety become widely adopted. I am thankful for the constant support, feedback, and encouragement they have provided me over the years. Martin has proofread every part of my work and his feedback has helped me grow as a researcher.

I am sincerely thankful for the close collaboration I have had with Jonas Krook and Ashfaq Farooqui. It was a pleasure to work with both of them and all our work together has influenced this thesis to a great extent.

I would like to thank Ali Hedayati for giving me the opportunity to pursue this PhD. Thanks also go to Ghazaleh Panahandeh for taking care of the funding administration and for all the advice. A special gratitude goes to Zhennan Fei for all the insightful discussions. I want to acknowledge the support from my colleagues at Zenseact and Chalmers. They helped me realise that I am not alone in this journey and have made the working atmosphere fun and stimulating.

I would not be where I am today without my parents who have continuously supported me in everything I do. I am indebted to my brother, Bharath, for taking care of things back in India during my absence.

This PhD would not have been possible without my wife, Lakshna. Since the start of this PhD journey, we have got married, pulled through a health scare, and recently have become parents. Thank you for having endured me through all of it, the good and the bad. I am forever grateful for bringing Yathran to our lives. I love you both. Finally, thank you also for helping me with the cover picture. What I tried, but failed to do in six hours using LaTeX, you accomplished in a few minutes using PowerPoint.

Yuvaraj Selvaraj
Gothenburg, October 2022.

This research was supported by FFI-VINNOVA, under grant number 2017-05519, Automatically Assessing Correctness of Autonomous Vehicles—Auto-CAV, and partially by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

Acronyms

ADS:	Automated Driving System
ADAS:	Advanced Driver Assistance Systems
ASIL:	Automotive Safety Integrity Level
DES:	Discrete Event Systems
E/E:	Electrical and/or Electronic
EFSM:	Extended Finite State Machine
FSM:	Finite State Machine
GSN:	Goal Structuring Notation
HP:	Hybrid Program
LSM:	Lateral State Manager
SAE:	Society of Automotive Engineers
SCT:	Supervisory Control Theory
SOTIF:	Safety of the Intended Functionality
TLA:	Temporal Logic of Actions

Contents

Part I

Overview

"*Science can amuse and fascinate us all, but it is engineering that changes the world.*"

Isaac Asimov[1]

CHAPTER 1

Introduction

The World Health Organization reports [1] that approximately 1.35 million people die each year due to road traffic accidents, thereby making traffic injuries a leading cause of human death globally. In addition, road traffic accidents are also a significant source of economic losses costing 3% of the gross domestic product in most countries [2] and amounting to $242 billion in the United States in 2010 [3]. The critical reason for more than 90% of road traffic accidents is attributed to some level of human error [4], [5]. The introduction of driving automation in road vehicles can potentially reduce such accidents and significantly improve road safety [4]. Automating the driving task also brings other benefits, such as reducing drivers' stress, and providing independent mobility for people who cannot and/or should not drive [3], [6].

The Society of Automotive Engineers (SAE) categorizes driving automation into six discrete and mutually exclusive levels based on roles of the human driver and the driving automation system in relation to each other [7]. The amount of human supervision required is a critical part of this taxonomy and the levels extend from Level 0 (no automation) to Level 5 (full automation),

[1]Asimov, I. & Shulman, J. A. (Eds.). (1988). Isaac Asimov's book of science and nature quotations. Weidenfeld & Nicolson.

as shown in Table 1.1. Levels 0 to 2 are driver support systems while Levels 3 to 5 are Automated Driving Systems (ADSs) [7].

Table 1.1: SAE levels of driving automation [8].

SAE Level 0	SAE Level 1	SAE Level 2	SAE Level 3	SAE Level 4	SAE Level 5
driver support features			**automated driving** features		
human driver responsible for driving whenever features are engaged			human driver is not driving when features are engaged		
constant human supervision is needed to maintain safety			human driver must drive on request	automated driving features will not require the human driver to take over	
limited to providing warnings and momentary assistance	provide steering OR brake/acceleration support	provide steering AND brake/acceleration support	these features can drive under limited operating conditions and will not operate until all requirements are met		this feature can drive under all conditions.
automatic emergency braking, lane departure warning	lane centering OR adaptive cruise control	lane centering AND adaptive cruise control	traffic jam chauffeur	local driverless taxi (pedals, steering wheel may or may not be installed)	same as Level 4 but feature can drive everywhere in all conditions

Since the introduction of driver support systems for consumer purchase in the 1990s [9], [10], Level 1 and Level 2 systems have become increasingly available as a standard option and this is expected to grow even further [11], [12]. However, Level 3 systems were made available for purchase only in 2022 [13], [14]. Currently, no vehicle with Level 4 or Level 5 automation is available for consumer purchase though recent advances have been made with Level 4 in the form of ride hailing services [15]. Higher levels of automation increase the complexity of the systems and several technical, business, and regulatory challenges arise [16]. A significant technical challenge is the safety verification and validation of ADSs [17]–[21].

1.1 Industrial Challenges

In general, a driving task can be divided into three levels [22]: *strategic* (e.g. route planning over long time horizon), *tactical* (e.g. maneuvering over a few seconds), and *operational* (e.g. speed control on milliseconds level). SAE J3016 [7] describes an ADS as the hardware and software that are collectively capable of performing the entire *Dynamic Driving Task* (DDT) on a sustained basis within an *Operational Design Domain* (ODD). The DDT includes all the real-time operational and tactical functions required to operate a vehicle and excludes the strategic functions. ODD describes the operating conditions under which the ADS is designed to function. An ADS is realized with the help of several electronic and/or electrical (E/E) subsystems that can be connected in many possible architectural designs. This thesis considers an ADS architecture with three subsystems as shown in Figure 1.1. This architecture follows the *sense-plan-act* paradigm [23] in artificial intelligence and robotics. Note that the DECISION & CONTROL in Figure 1.1 corresponds to the *plan* in *sense-plan-act*.

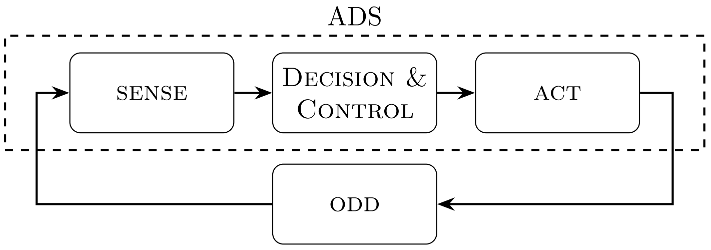

Figure 1.1: A simplified architecture for an ADS.

The SENSE subsystem makes use of sensors such as cameras, radars, gyros, etc., to perceive the environment, here represented by the ODD, and fuses all the information from the different sensors to provide a mathematical model of the traffic situation, such as the vehicle state, road geometry, distance to other vehicles, etc. The DECISION & CONTROL subsystem uses the fused information to decide when and how to act. Typical sub-tasks here are threat assessment, decision-making, path planning, etc. The decisions are then communicated to the ACT subsystem in the form of, e.g., acceleration and steering commands. Finally, these decisions are executed by using, e.g. brake and/or steering actuators to control the vehicle. Each of these subsystems further includes several functional modules that interact together to solve the respective tasks.

The decision-making in an ADS is distinctly challenging because the system must interact with and predict the intentions of other traffic participants in complex and uncertain environments. Furthermore, the sensor limitations and the actuator capabilities have to be considered in the decision-making. The complexity involved due to the interactions with the environment and between the different subsystems is manually intractable. This may lead to subtle but potentially dangerous bugs arising due to unforeseen edge cases, errors in design and/or implementation. Therefore, it is imperative that all safety-critical parts of an ADS are veritably reliable and safe. This is a challenge for the design as well as the verification and validation of ADSs.

Although human error is a major cause of many traffic accidents, the actual failure rate is remarkably low. For instance, in the United States, the fatality rate is 1.34 deaths per 100 million miles driven [17], [24]. This low rate presents a challenge as the failure rate of ADSs has to be at least better than human driving in order to increase overall traffic safety. Such a requirement presents difficulties in the design and verification of these systems as it is crucial to provide sufficient evidence to show that the safety goals are fulfilled. While human supervision is necessary to maintain safety in Level 0–2 driver support features, there needs to be a convincing safety argument for ADS. Ensuring safety of ADS is a multi-disciplinary challenge, and several approaches have been researched and attempted to address this challenge [20], [21].

The shortcomings in current industrial practice are further highlighted by the impact of defective software deployed in production vehicles. Many potentially dangerous software defects have been reported in the past that could result in catastrophes not due to driver error [25]. These defects have resulted in hundreds of thousands of vehicle recalls as they pose an unreasonable safety risk. More than 15 million vehicles were affected by software related safety recalls in the United States in 2019 [26]. In addition, fatalities and accidents involving driving automation systems in Tesla [27]–[29], Uber [30], [31], and Google [32], [33] have warranted stricter safety arguments for successful deployment of automated vehicles at scale.

Formal methods are techniques that use rigorous mathematical models to build hardware and software systems, and can provide a mathematical proof of the correctness of the system. The strength of formal methods lies in the use of unambiguous formal logic and the possibility to provide irrefutable evidence for the safety argument. Formal methods consist of various diverse techniques and a particularly useful approach that aids in providing evidence to demonstrate the safety of ADS is *formal verification*.

Given a *formal model* of a system and a *formal specification* of the intended behaviour of the system, formal verification is the act of proving or disproving the correctness of the system with respect to its specification. Formal verification has been shown to be beneficial in verifying safety-critical systems such as avionics [34]–[36], railway systems [37], [38], nuclear plants [39]. Despite successful applications in automotive systems [40]–[44], formal verification is not a well established approach in the industry. *The focus of this thesis is to explore the barriers that hinder the industrial adoption of formal verification and investigate how formal verification can be used for application in the automotive industry, specifically in the development of safe ADS.*

1.2 Research Questions

This thesis mainly concerns the problem of how to establish formal methods as an efficient approach to develop safe ADS. It is hypothesised that formal methods and in particular formal verification can be used in the industrial development processes to produce formal proofs of correctness of Decision & Control systems and thereby provide rigorous evidence for the safety argument of ADS. Impelled by the limited industrial adoption of formal methods despite potential benefits, the obstacles that impede widespread adoption are investigated. To this end, this thesis considers the following research questions:

RQ 1 What factors affect the application of formal verification to ADS and what are the current challenges in existing methods?

RQ 2 How can the challenges be addressed, and can the solutions be scaled?

RQ 3 How can formal methods be used to provide evidence in the safety argument of ADS?

1.3 Research Methods

The work of this thesis has been carried out as part of an industrial PhD project at Zenseact. As is evident from the research questions, the research is motivated by the need from the industry and therefore the problem statements, research methods, and the results are tightly connected to the industrial research and development of ADS. For the same reason, much of the research carried out, combines aspects of both practical utility and the quest for basic understanding as illustrated through the Pasteur's quadrant model [45] in Figure 1.2.

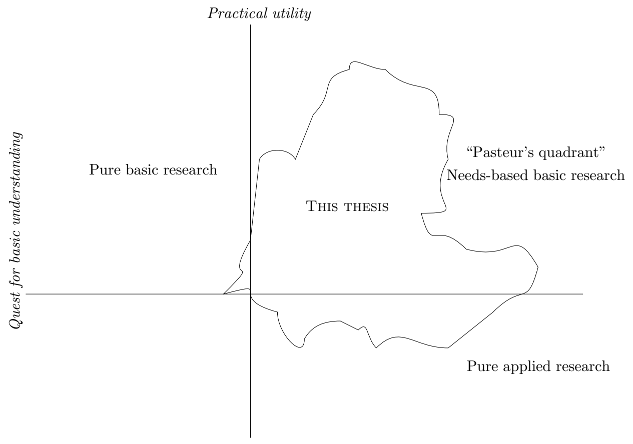

Figure 1.2: This thesis in relation to Pasteur's quadrant model of scientific research.

The choice of a research method depends on various factors such as the nature of the research questions and the kind of knowledge being pursued [46]. **RQ 1** has exploratory characteristics while **RQ 2** and **RQ 3** have problem-solving characteristics. Several research methods are suitable to answer them and have accordingly been applied. The following paragraphs provide a brief summary.

Method 1: It is essential to obtain a good overview of the current state of knowledge in an area before research is carried out to advance the knowledge in that area. *Critical analysis of literature* is "an appraisal of relevant published material based on careful analytical evaluation" [47]. This research method is employed in all the papers included in this thesis.

Method 2: *Case study* is an examination of a system in detail that is exploratory in nature [48]. As suggested by the name, a case study focuses on one or a number of cases for an in-depth study. Case studies are suitable to answer questions that are exploratory as well as questions that require problem-solving [46]. This method is primarily used in Paper A and Paper B.

Method 3: *Proof of concept* is a kind of research that aims to answer a question with an idea that may have wide applicability to a class of phenomena [49]. This method is particularly efficient to demonstrate the usefulness of a concept/solution and is employed in Paper B and Paper C.

Method 4: *Mathematical modelling and mathematical proofs* as a research method provides mathematical models to explore and make logical arguments in the form of proofs that a certain proposition is true [47]. Unsurprisingly, this method is used in all papers included in this thesis and is the predominant method in Paper D.

1.4 Scientific Contributions

The nature of the research questions not only affects the choice of the research method, but also the knowledge that is generated as the outcome of the research. Due to the exploratory nature of **RQ 1**, this work has led to descriptive knowledge in the form of evaluation of characteristics of the researched phenomenon, identification of further research gaps, etc. Whereas, the problem-solving nature of **RQ 2** and **RQ 3** has led to prescriptive knowledge in the form of guidelines, conceptual solutions, etc. The scientific contributions are documented in the included papers and Table 1.2 maps how the included papers relate to the research questions. The main contributions of this thesis are:

- It evaluates several logical formalisms and methods to model, specify, and verify ADS. In doing so, it identifies various challenges in applying formal verification to argue safety of ADS (Paper A and Paper C).

- It identifies the need for an integrated approach to argue safety of ADS. To help developers and safety engineers in that regard, it provides modelling guidelines and insights into how the evaluated methods differ with respect to the formalism, verification objective, and their applicability based on the different levels of abstraction. (Paper A, Paper C and Paper D).
- It presents methods to identify and address subtle modelling errors that might inadvertently result in proving a faulty design to be safe. This helps to avoid a fallacious safety argument of the ADS, which could potentially be catastrophic in practice (Paper D).
- It presents a systematic approach to automatically obtain formal models from ADS software to overcome the obstacle of manual effort needed to apply formal verification. The approach is validated by a proof of concept in which two algorithms are comparatively evaluated through an industrial case study (Paper B).
- It presents a structured approach and demonstrates how to use the different formal artifacts in the development and in the safety argument of an ADS (Paper C and Paper E).

Table 1.2: Relation between the research questions and the included papers. ✓denotes that the corresponding RQ is addressed.

RQ	Paper A	Paper B	Paper C	Paper D	Paper E
RQ 1	✓		✓		
RQ 2		✓		✓	
RQ 3			✓		✓

1.5 Thesis Structure

The thesis is divided into two parts. Part I gives an introduction to automated driving, the challenges in providing evidence for the safety of an ADS, the necessary fundamentals, and a summary of scientific contributions from the included papers. Part II contains the papers. Part I consists of the following chapters:

Chapter 1 – Introduction
This current chapter provides a brief introduction to automated driving, the research questions and the contributions of this thesis.

Chapter 2 – Safety Evidence
The second chapter puts into perspective how formal methods compare to the other current industrial approaches to provide credible safety argument for automated driving.

Chapter 3 – Formal Methods
The third chapter presents the necessary preliminaries and the fundamental concepts of different formal methods discussed in this thesis.

Chapter 4 – Safety Proofs for Automated Driving
This chapter presents insights obtained from the included papers towards provable correctness for Decision & Control systems and connects them in the context of this thesis.

Chapter 5 – Summary of Included Papers
This chapter provides a brief summary of the included papers and their contributions.

Chapter 6 – Concluding Remarks and Future Work
The final chapter concludes the first part of the thesis and presents ideas for future research.

“*There is nothing more deceptive than an obvious fact.*”
Sir Arthur Conan Doyle[2]

CHAPTER 2

Safety Evidence

The problem of providing sufficient evidence to demonstrate safety of a system is important in several safety-critical industries. The terms *safety* and *safety case* have been defined in many (but not too different) ways in various industry-specific safety standards [50]–[52]. This thesis adopts the following definition for those terms.

Safety is the absence of an unreasonable risk of harm. A *safety case* is a structured argument supported by evidence that provides a compelling, comprehensible, and valid case that a system is safe in a given operational environment.

The terms safety case and safety argument are sometimes used interchangeably in this thesis due to their almost identical meaning. While several approaches [17], [20], [21] have been researched and adopted to provide sufficient evidence for safety of ADSs, they do have their own limitations. In that context, this chapter presents a brief overview of how formal methods compare to some of the other industrial approaches.

[2]The Boscombe Valley Mystery - A Sherlock Holmes Short Story

2.1 Conformance to Safety Standards

A familiar strategy for safety argumentation is to show evidence for conformance to industry-specific safety standards. For instance, many safety standards exist in the transportation industry to aid in the development of safety-critical software. The DO-178C standard [53] is used by various certification authorities in the aerospace industry to certify software for aerospace systems. Similarly, the railway industry uses the EN50128 (IEC 622279) [54] as one of the safety standards. For the automotive industry, the ISO 26262 functional safety standard [52] is the primary standard to address the safety of electrical and/or electronic (E/E) systems within road vehicles.

ISO 26262

ISO 26262 describes a functional safety framework for the development of safe E/E systems. Functional safety, as defined by the standard is *the absence of unreasonable risks that could be caused by the malfunctioning behaviour of the E/E systems*. The standard provides a reference for the automotive safety lifecycle and takes a risk-based approach to mitigate and/or avoid risks. ISO 26262 gives appropriate guidelines to achieve functional safety throughout the development process, which includes activities like requirement specification, design, implementation, verification and validation.

The initiating task of the safety lifecycle is the *item* definition. The objective of the item definition phase is to develop a description of the system, or combination of systems, for which functional safety needs to be achieved and subsequently the item's functionality, boundaries, interfaces, environmental conditions, assumptions, etc. are determined. The item definition phase is then followed by a hazard analysis and risk assessment phase where potential hazardous events of the item are identified and classified based on three factors:

- the *severity* of the potential harm caused by the hazardous event,
- the probability of *exposure* of the hazardous event, and
- the *controllability*, which estimates the ability to avoid the specific harm.

Together, the three parameters determine the Automotive Safety Integrity Level (ASIL) [52]. Four possible levels: ASIL A, ASIL B, ASIL C, and ASIL D are defined to categorize the hazardous events. ASIL A is the lowest safety integrity level and ASIL D is the highest. These levels are then used to obtain the safety goals that form the top-level safety requirements for the system. These safety goals are the key to achieving functional safety and the standard recommends best practices for the design, implementation, and verification of the item such that the safety goals are met.

In subsequent phases of the safety lifecycle, safety goals are broken down to obtain detailed low-level safety requirements. These detailed requirements, derived as a result of the requirement refinement process, correspond to different phases in the design. For instance, the safety goals lead to implementation-independent functional safety requirements, which in turn are refined to obtain implementation specific technical safety requirements. Since the ASIL allotted to a safety goal is inherited throughout the process, sufficient verification evidence for all the requirements is needed to comply with the safety goal.

Evidently, a higher ASIL demands more rigorous development processes compared to a lower ASIL. This fact is also reflected in the recommended best practices for the development of high ASIL components in the standard. For instance, while the use of semi-formal and formal methods (in addition to other methods) are recommended for higher ASIL components, the standard has no recommendation for or against their usage for lower ASIL components.

Several arguments [18], [20], [55] have been presented to show that ISO 26262 presents unique challenges and is sometimes even inadequate to completely demonstrate safety of ADSs. Specifying unambiguous safety requirements at each phase of the development process, and providing evidence that such requirements at each level in conjunction implies the overall safety goal, are significant challenges. Another specific concern is the impact of the inherent assumptions that are made in the standard. As an example, in the ASIL analysis, especially the controllability parameter assumes a human driver to react and mitigate/avoid risks. While such an assumption might be relevant in driver support features, it is fatal for ADSs. Yet another arguable point is the assumption of the V model [52], [56] in the reference safety lifecycle. The relevance of ASIL analysis and the recommended best practices in a development approach other than the V model is debatable. Such challenges have initiated the development of new standards [57], [58] specific to autonomous systems.

Other Relevant Standards

The recently published UL 4600 [57] standard addresses the safety processes for the evaluation of automated vehicles and other products without human supervision. The standard intends to provide additional elements to address the needs for safety of full autonomy and is therefore expected to work with other existing standards. UL 4600 takes a claim-based safety case approach that includes structured set of goals, arguments, and evidence to support the fact that an item is safe for deployment. While UL 4600 provides guidance and recommended best practices to improve the consistency of the safety case, it does not mandate any specific development process (e.g., the V model). Most importantly, UL 4600 highly recommends the use of formal methods as a verification and validation approach to provide sufficient evidence of safety.

The ISO/PAS 21448 [58] standard addresses the safety of systems that rely on sensing the environment, which is essential for an ADS. The ISO/PAS 21448 standard concerns the safety of the intended functionality (SOTIF) and is complementary to the functional safety aspect of ISO 26262. SOTIF provides guidelines to achieve absence of unreasonable risks due to potential hazards caused by unintended behaviour in a system that is free from faults addressed in ISO 26262. Limitations that arise due to the inability of a function to have correct situational awareness and performance issues due to sensor variations are typical examples of risks dealt with in this standard.

Though these standards exist, conformance to any particular standard alone is not a guarantee of a fully safe autonomous vehicle. However, conformance to such standards provides the necessary rigorous evidence to support the safety argument. Formal methods, by definition requires rigorous mathematical proofs and therefore meets the objective of such safety standards, wherever applied.

2.2 Testing, Simulation, and Miles Driven

At present, the most widely used industrial approach to assure automotive software safety is *testing*. Requirements based testing is an important aspect of the V model development framework in the reference safety lifecycle of ISO 26262. Testing can be done at various levels, such as software unit testing, model-based testing, complete vehicle on-road testing, and scenario-based virtual simulations. However, there are several challenges associated with a testing based approach to prove safety [18]. The most prominent challenge is that testing can only find errors, but can never guarantee their absence.

Testing safety by driving under real world conditions is expensive and impractical. An important question here is how many test miles need to be driven to demonstrate safety? Studies have shown [17] that it would be necessary to drive hundreds of thousands of miles, and sometimes even billions of miles to statistically demonstrate the reliability of ADSs in terms of fatalities and injuries compared to human driving. In addition to being expensive and potentially dangerous to public safety, such an approach does not scale well.

One way to reduce the cost and scale of vehicle-level testing is to run the tests virtually by simulating a wide range of scenarios representing real world conditions. While this approach overcomes some of the disadvantages of field testing, it brings a different set of unique challenges. For instance, the reliability of the simulation framework, selection of statistically significant simulations, and the need for massive amounts of data to accurately represent real world environment are some of the challenges that have to be addressed [20], [21].

2.3 Formal Methods

Formal methods, especially formal verification can provide mathematical proofs of correctness and complement other methods to provide sufficient evidence for the safety argument. Though formal methods, in contrast to testing can guarantee the absence of errors, they do not remove the need for testing. In comparison to non-exhaustive methods like testing and simulation, which require *dynamic* analysis (e.g. executing code, running software in a vehicle), formal verification techniques are typically *static* in nature. This makes them best suited to detect subtle errors in the early stages of development and potentially avoid catastrophic consequences associated with vehicle recalls [25], [26]. Formal verification can indeed provide sufficient rigour in the safety argument as required by different safety standards. Despite the advantages, the industrial adoption of formal verification in the automotive industry for ADS development is below par. In order to ease the industrial adoption, associated challenges need to be explored and possible solutions investigated. In the subsequent chapters, this thesis throws some light on these aspects.

"*Modelling in science remains, partly at least, an art.*
Some principles do exist, however, to guide the modeller.
A first, is that all models are wrong."

P. McCullagh and J.A. Nelder[3]

CHAPTER 3

Formal Methods

Formal methods are a category of mathematically rigorous techniques and tools that can be used to specify, design, and verify hardware and software systems. Admittedly, the term *formal methods* is quite broad and a variety of tools and techniques are available. Currently, an online resource [60] lists more than 100 different formal notations, methods, and tools for describing and reasoning about computer-based systems.

Formal methods can be classified in many ways depending on the level of formality, modelling formalism, ease of use, etc. Almeida *et al.* [61] argue that the characteristic aspect of formal methods is the ability to guarantee the behaviour of a given system with some rigorous approach. Therefore, the *specification*, which is a description of the desired behaviour, is at the core of formal methods. In this context, the different groups of formal methods can be classified into four types as shown in Table 3.1. The main focus of this thesis is on the safety verification of ADS, and therefore the scope is limited to the "Specify and Prove", i.e., the *formal verification* category of formal methods.

[3]McCullagh, P., & Nelder, J. A. (1983). Generalized linear models. Routledge. A similar expression "All models are wrong, but some are useful" is attributed to George Box [59]

Table 3.1: Classification of formal methods [61]

Classification	Approach	Example
Specify and Analyse	specification is formally defined and analysed using animation, execution, etc.	ASM, Z, VDM
Specify and Prove	specifications and models are formally defined to prove properties about them (**formal verification**)	model checking, SPARK, KeY, KeYmaera X
Specify and Derive	given a formal specification, obtain an implementation whose behaviour matches the specification (**correct-by-construction**)	supervisory control theory, program refinement
Specify and Transform	transform a specification to either approximate or enrich it	abstract interpretation

Given a formal model of a system and a formal specification of its desired behaviour, *formal verification* is the act of proving or disproving the correctness of the system with respect to the specification.

At the mention of formal verification, a natural question is: how to specify, model, and prove properties of systems? Of course, there are multiple ways to specify and prove depending on the modelling formalism and the proof technique used. Typically, the choice of verification method depends on the nature of the system. For instance, to verify distributed asynchronous systems, model checking using SPIN [62] can be used; and to verify Java programs, the deductive verification tool KeY [63] can be used.

In the case of an ADS, such a straightforward choice of the verification method is no longer possible due to the complexity involved. An ADS must interact with and adapt its behaviour to the surrounding environment in a feedback loop. The Decision & Control subsystem of an ADS as shown in Fig. 1.1 must consider the uncertainty of the environment in the decision-making, which increases the complexity. Furthermore, the interactions between the different subsystems and the interaction between the different functions within a subsystem also affect the complexity involved. Such factors highlight the need to identify appropriate verification methods and their suitability to provide sufficient evidence for the safety argument of ADS.

A variety of formal verification techniques have previously been used to

Table 3.2: Formal verification methods investigated, see Paper A and Paper C.

Method	Tool
Control theoretic	SUPREMICA
Model checking	TLA^+
Deductive verification	SPARK, KeYmaera X

reason about autonomous systems [64], [65]. This thesis focuses on three general methods as shown in Table 3.2. The choice of these methods is primarily motivated by the following important factors:

- characteristic of the DECISION & CONTROL subsystem in an ADS,
- established proof of concept for verification of safety-critical systems,
- capability to handle industrial sized systems.

The rest of this chapter introduces the different methods investigated in this thesis to specify and prove properties of ADS. Section 3.1 presents the formalism of the methods to *specify* the requirements. This is followed in Section 3.2 by brief explanations of the features available in the respective methods to *model* the behaviours of an ADS. Finally, Section 3.3 discusses the *verification* approach used in the different methods. This structure is aimed to help making a qualitative comparison of the different aspects of these methods in Chapter 4 based on the included papers.

3.1 Formal Specification

In the context of this thesis, the term *formal specification* refers to a formal representation of the ADS requirements to be verified. Three types of formal specifications are considered: state machine based, logic based, and contract based.

State Machines

Often, tactical decision-making in an ADS can be viewed as an *event-driven* process. For instance, a decision to emergency brake can be triggered on the event of an obstacle unexpectedly appearing in front. This view leads to the class of *Discrete Event Systems* (DES) [66], which are discrete-state, event-driven systems that occupy at each time instance a single *state* out of many possible ones, and transits to another state on the occurrence of an *event*. Thus, a characteristic feature of DES is the notion of instantaneous and discrete events that may be associated with specific phenomena in an ADS, such as detecting an object or switching on the turn indicator.

A typical way to reason about DES is to analyse its *logical behaviour*, that is, analysing whether a sequence of events satisfies a specification (e.g. reaching a desired state). A straightforward way to specify such behaviours is through the *language* of the DES. Formally, the finite nonempty set of events is called an *alphabet*, denoted by Σ. A *string* is a finite sequence of events chosen from Σ. The set of all strings over Σ is denoted by Σ^*. A language $\mathcal{L} \subseteq \Sigma^*$ is a set of strings over Σ.

Though languages are suitable to specify logical behaviours of DES, they are not convenient to describe and analyse a system. In this regard, state machines [66] provide an alternative way to represent languages and specify DES behaviours. The central idea behind formally specifying DES behaviours as state machines is the possibility to systematically analyse them for the acceptance of a specified property.

Typically, the specification is expressed as a state machine with *marked* states [66], [67] that express the desired behaviour. The intention is that at least one of these marked states should always be reachable.

In Paper A and Paper B, two kinds of state machines are used: Finite State Machine (FSM) [66] and Extended State Machine (EFSM) [67]. An EFSM extends an FSM with bounded discrete variables, guards (logical expressions) over the variables, and actions that assign values to the variables on the transitions. The current state of an EFSM is given by its current location together with the current values of the variables. In either case the marked states specify desired behaviours and can be checked for reachability.

Remark: In the context of this thesis, the terms finite state machine and extended finite state machine are synonymous to finite automata and extended finite automata, respectively.

Temporal Logic of Actions

The Temporal Logic of Actions (TLA) [68] is a variant of temporal logic [69]. TLA, as a logical formalism provides the expressive power to reason about systems using assertions on states and pairs of states (actions). The reasoning mechanism in TLA is built around TLA formulas. A TLA *formula* is true or false on a behaviour. A *behaviour* in TLA is an infinite sequence of states. A *state* in TLA is an assignment of values to variables. A *state predicate* is a boolean valued expression on states. A *step* is a pair of states. Steps of a behaviour denote successive pairs of states.

Actions are predicates that relate two consecutive states and are used to capture how the system is allowed to evolve. An action, $\mathcal{A}$, is a boolean valued expression on steps and represents a relation between old states and new states. A step is an $\mathcal{A}$-step if it satisfies $\mathcal{A}$. An action is valid, $\models \mathcal{A}$, iff every step is an $\mathcal{A}$-step. In TLA, atomic operations are represented by actions.

The elementary building blocks of a TLA formula include state predicates, actions, logical operators (such as $\wedge$,$\neg$, etc.), the temporal operator $\Box$ (always) and the existential quantifier $\exists$. A summary of some formulas of TLA and their meaning is given in Table 3.3

Table 3.3: Formulas of TLA [68], [70]. P is a state predicate, $\langle vars \rangle$ is a tuple of variables, $\mathcal{A}$ is an action

Formula	Meaning
P	true of a behaviour iff P is true in the first state of the behaviour
$\Box P$	true of a behaviour iff P is true in every state of the behaviour
$\Box[\mathcal{A}]_{\langle vars \rangle}$	true of a behaviour iff every step is an $\mathcal{A}$-step or leaves values of *vars* unchanged

One way to specify properties in TLA is to use *invariants* of the form $\Box\, P$. Invariants are state predicates that should be valid in all reachable states of the model.

Differential Dynamic Logic

Differential dynamic logic (dL) [71], [72] is a specification and verification language for *hybrid systems*, which are mathematical models of systems that combine discrete dynamics with continuous dynamics.

The formulas of dL include formulas of first-order logic of real arithmetic and the modal operators $[\alpha]$ and $\langle\alpha\rangle$ for any *hybrid program* α (see Section 3.2 for hybrid programs). The dL formula $[\alpha]\,\phi$ expresses that *all* non-aborting executions of α reach a state in which the dL formula ϕ is true. The dL formula

$\langle\alpha\rangle\,\phi$ means that there exists *some* non-aborting execution leading to a state where ϕ is true. $\langle\alpha\rangle\,\phi$ is the dual to $[\alpha]\,\phi$, defined as $\langle\alpha\rangle\,\phi \equiv \neg[\alpha]\neg\phi$.

To specify the correctness of an ADS (modelled as a hybrid program α) with respect to a requirement described by *guarantee*, a dL formula of the form

$$(\mathit{init}) \rightarrow [\alpha]\,(\mathit{guarantee}) \tag{3.1}$$

can be used. If the formula is valid, then it expresses that for every state in which the formula *init* is true, all (non-aborting) executions of α only lead to states where formula *guarantee* is true.

Program Contracts

The types of specifications discussed in the previous sections are used to describe properties that can be verified on an abstract model of a system. This thesis also investigates formalisms that can be used to specify properties at the source code level. In this context, this section describes how specifications can be written in the programming language SPARK [73]. SPARK is a subset of Ada [74], an imperative programming language targeted for the development of large scale safety critical software.

The intended behaviour of a SPARK program can be specified using a *contract* attached to a *subprogram* (e.g. a function). A subprogram contract may be made of various parts including pre- and post-conditions. A *precondition* specifies constraints on the caller that must be satisfied upon entry of the subprogram and a *postcondition* specifies the conditions that must be adhered to by the subprogram upon exit. In addition, properties in SPARK can also be specified using loop invariants and data dependencies. The specifications in SPARK are expressed in the form of program annotations written inline with the source code as shown in Listing 3.1 using an example of a contract with pre- and post-conditions on the subprogram `foo`.

Listing 3.1: An example of a contract in SPARK.

```
procedure foo
  (mode : in out Integer)
with
  Pre      => 2 < mode and mode < 5,
  Post     => mode = 0;
```

Here, the precondition requires the `mode` parameter to have a value largerthan 2 and less than 5, and the postcondition ensures that the value is set to 0 on return.

Remark: The dL specification in (3.1) can be considered as a contract on a hybrid program where *init* corresponds to the precondition and *guarantee* corresponds to the postcondition.

3.2 Formal Models

In the context of this thesis, the term *formal model* is a description of the possible behaviours of the ADS. This section briefly discusses the various formalisms used in the included papers to model an ADS. Each specification formalism is associated with its respective modelling formalism and hence the modelling formalisms are introduced in the same order as their corresponding specification formalism in the previous section.

State Machines

FSMs and EFSMs described in Section 3.1 are not only suitable to formally specify properties of a DES, but also to model controllers designed to control a DES. A controller may interact with several different components, such as other subsystems that are part of the DES, or the environment that the DES is to react on. Such interactions between state machines are typically modelled by *synchronous composition* [66], [67].

For two FSMs F_1 and F_2, the synchronous composition denoted as $F_1 \parallel F_2$ models the interaction through shared events. The shared events occur simultaneously in the interacting FSMs when possible, while non-shared events occur independently. In the case of EFSMs, in addition to the synchronous composition, interaction can also be modelled through shared variables [67]. Thus, a complex ADS can efficiently be modelled as a set of interacting state machines in a *modular* way to reason about their overall behaviour.

TLA$^+$

TLA$^+$ [70], [75] is a specification and modelling language for concurrent and reactive systems. TLA$^+$ is based on formal set theory, first order logic, and TLA. TLA$^+$ models describe the possible behaviours of a system and are typically referred to as *specifications*. However, here this terminology is slightly abused to be consistent with the rest of this chapter. Thus, a TLA$^+$ model is a description of the behaviours of a system and a TLA$^+$ specification is a desired property of the system. TLA$^+$ describes a system as a set of behaviours with an initial condition and a next state relation. The initial condition describes the possible initial states and the next state relation describes the possible steps. A TLA$^+$ model, here denoted by *Spec* is a temporal formula of the form

$$\mathit{Spec} \triangleq \mathit{Init} \land \Box\, [\mathit{Next}]_{\langle \mathit{vars} \rangle} \land \mathit{Temporal}, \tag{3.2}$$

where *Init* is a state predicate corresponding to the initial condition, *Next* is an action corresponding to the next state relation, *vars* is a tuple of all variables in the model, and *Temporal* is a temporal formula usually specifying liveness conditions. Formula *Spec* can be seen as a predicate on behaviours. *Spec* is true for a behaviour σ, iff *Init* is true in the first state of σ and every step in σ is either a step that satisfies *Next* or is a *stuttering* step. A *stuttering* step is one in which none of the variables are changed.

Remark: Typically, TLA$^+$ models are referred to as *specifications*, which is why (3.2) is denoted *Spec*. Some aspects of modelling and specification are blended in formalisms like TLA$^+$ and state machines. Moreover, TLA$^+$ models of the form (3.2) describe a set of behaviours with a condition on the initial state and a next-state relation on pairs of successive states. Thus, such TLA$^+$ models closely resemble state machine based models.

Hybrid Program

The logic dL uses *hybrid programs* (HP) [71], [72] to model hybrid systems. Each HP α is semantically interpreted as a reachability relation $[\![\alpha]\!] \subseteq \mathcal{S} \times \mathcal{S}$, where $\mathcal{S}$ is the set of all states. If $\mathcal{V}$ is the set of all variables, a state $\omega \in \mathcal{S}$ is defined as a mapping from $\mathcal{V}$ to $\mathbb{R}$, i.e., $\omega \colon \mathcal{V} \to \mathbb{R}$.

Discrete changes are modelled using *discrete assignment* $x := e$, which assigns the value of term e to variable x, and *nondeterministic assignment* $x := *$, which assigns an arbitrary real number to x. HPs model continuous changes as $x' = f(x) \,\&\, Q$, which describes the *continuous evolution* of x along the differential equation system $x' = f(x)$ for an arbitrary duration (including zero) within the *evolution domain constraint* described by the first-order formula Q. HPs use *test* action $?P$ to model decision conditions. The test action $?P$

of a first-order formula P has no effect in a state where P is true, i.e., the next state is the same as the current state. If, however, P is false when $?P$ is executed, then the current execution of the HP *aborts*, meaning that no transition is possible and the entire current execution is removed from the set of possible behaviours of the HP.

HPs also include *sequential composition* (;), *nondeterministic choice* ($\cup$), and *nondeterministic repetition* (*) to model hybrid systems. The sequential composition $\alpha; \beta$ expresses that HP β starts executing after HP α has finished. The nondeterministic choice operation expresses that the HP $\alpha \cup \beta$ can nondeterministically choose to follow either α or β. The nondeterministic repetition α^* expresses that α repeats n times for any $n \in \mathbb{N}_0$.

The nondeterministic modelling features in a HP are particularly useful to describe an ADS and a discussion on this is presented in Section 4.2. To model and specify the ADS system in Figure 1.1, the hybrid program α in (3.1) can be refined as a nondeterministic repetition of three sequentially composed HPs, each describing the respective subsystems as:

$$\mathit{init} \rightarrow [(\,\mathit{env};\, \mathit{ctrl};\, \mathit{plant}\,)^*]\,(\mathit{guarantee}) \tag{3.3}$$

where *env* corresponds to the environment as perceived by SENSE, *ctrl* corresponds to the DECISION & CONTROL, and *plant* corresponds to the continuous evolution of the physical system as a result of the execution of the actuation commands by ACT.

SPARK Program

A SPARK program [73], [76], [77] is made up of one or more program units. Subprograms and packages are two examples of SPARK program units. A subprogram execution is invoked by a call and subprograms express a sequence of actions. Procedures and functions are the two types of subprograms in SPARK. Procedure calls are standalone statements, whereas function calls occur in an expression and return a value. Packages group together entities like data types, subprograms, etc. A program unit consists of two structures, a specification and a body. The specification contains the variables, types and the subprogram declarations with their annotations. The body of a program unit contains the details of the implementation, i.e., the source code.

In addition to user defined program contracts as described in Section 3.1, SPARK has a set of core annotations as predefined rules that can be checked without user defined contracts such as checking for divisions by zero, or out of bounds array indexing. The contracts are used by the analysis tools to generate verification conditions, which are mathematical expressions relating a number of hypotheses (obtained from preconditons) and conclusions (from postconditions). Providing a correctness proof of the program then boils down to showing that the conclusions always follow from the hypotheses. Detailed information on the the analysis tools is available in [73], [76].

3.3 Formal Verification

As shown in Table 3.2, three different formal verification methods and their associated tools are investigated in Paper A and Paper C. Though each method has its own theoretical foundation, the three methods can be classified into two categories based on the fundamental strategy used to verify the correctness of the system. The first two methods investigated, namely Supervisory Control Theory and model checking, belong to the *enumeration* category, while deductive verification belongs to the *deduction* category. The basic idea in the enumeration category is to provide a proof of correctness by enumeration, that is, enumerate the finitely-many possible behaviours of the system and check each of them. In the deduction category, however, the basic idea is to provide a proof by mathematical deduction. In this case, deductive reasoning is used to derive conclusions from a set of assumptions. In the following sections, each of the three methods are briefly examined.

Supervisory Control Theory

The Supervisory Control Theory (SCT) [78] provides a framework for modelling, synthesis, and verification of reactive control functions for DES. Given a DES model of a system to control, the *plant* G, and a *specification* K of the desired controlled behaviour, SCT provides means to synthesise a *supervisor* that, interacting with the plant in a closed-loop system, dynamically restricts the event generation of the plant such that the specification is satisfied.

Though the original SCT focused on synthesising supervisors that by construction fulfill the desired properties, a dual problem of interest here is to, given a model of a plant and a specification, verify whether the spec-

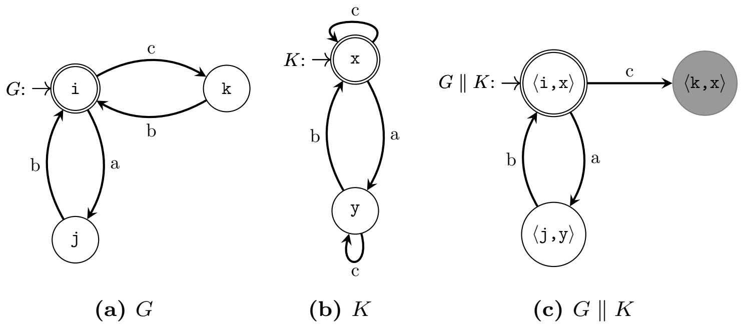

Figure 3.1: Illustration of nonblocking verification. The marked states are indicated by double circles. Though all three states in $G \parallel K$ are reachable, the grey state $\langle k, x \rangle$ is not coreachable and hence a blocking state.

ification is fulfilled or not. Paper A exploits one aspect of this duality, namely the *nonblocking verification* problem. Given a set of state machines $\mathcal{SM} = \{G_1, \ldots, G_n, K_1, \ldots, K_m\}$ where the components G_i $(i = 1, \ldots, n)$ represent the plant, and K_j $(j = 1, \ldots, m)$ represent the specification, nonblocking verification aims to determine whether the synchronous composition over all the components can from any reachable state always reach some marked state.

The straightforward way to perform nonblocking verification is to show that a state machine has no blocking state, which is a state from where no marked state is reachable. Blocking states are determined by comparing the reachable states and the *coreachable* states, i.e., states from where a marked state can be reached. Any reachable but not coreachable state is a blocking state as illustrated in Figure 3.1. Several algorithms [79], [80] have been developed to tackle the state-space explosion problem and perform efficient verification. The software tool Supremica [81] implements such verification and synthesis algorithms, as well as other techniques for modelling and analysis of DES.

Model Checking

Model checking [82] is an automated verification method for finite-state models. Given a finite-state model of a system and a formal property expressed using temporal logic as the specification formalism, the model checking problem systematically checks whether the property is satisfied in the model. If a state that violates the property is encountered, the model checking algorithm provides a counterexample as an evidence for the violation.

The TLC model checker [70], [83] is explicitly designed to check TLA$^+$ models of the form (3.2). The input to TLC is the TLA$^+$ model and a description of the properties to be checked. A typical way to specify properties of TLA$^+$ models is through the use of *invariants* of the form $\Box P$. Invariants are state predicates that should be true in all reachable states.

To check an invariant, TLC explores all reachable states, which are explicitly represented, and evaluates the invariant in each reachable state. If a state is found that does not satisfy the invariant, TLC generates a minimal-length trace [83] that leads to the violated state from the initial state. The TLA$^+$ language accompanied by an IDE consisting of TLC and other useful tools can be downloaded from [75].

Deductive Verification

Nonblocking verification and model checking employ exhaustive state space exploration to verify the correctness of a system. In contrast, deductive verification [84] uses a deductive mechanism of reasoning based on axioms and inference rules to produce a proof of correctness of the system. Often, deductive approaches are supported by theorem provers with varying levels of automation for the development of the proofs. In this thesis, deductive verification is used to prove correctness of both hybrid programs and SPARK programs and the overall approach can be summarised using Figure 3.2.

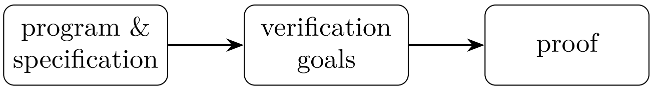

Figure 3.2: Simplified deductive verification workflow

KeYmaera X [85] is an interactive theorem prover that can do deductive verification on hybrid systems. The input to KeYmaera X is a single dL formula such as (3.3) with a description of the hybrid system in the form of a HP and the specification to be verified. KeYmaera X successively decomposes such a dL formula into several verification goals according to the sound proof calculus [71], [72] of dL to prove the formula. The proof calculus of dL uses *sequents* to structure the proofs. A sequent is of the form $\Gamma \vdash \Delta$ where antecedent Γ and succedent Δ are dL formulas. A sequent $\Gamma \vdash \Delta$ is semantically equivalent to the dL formula $\bigwedge_{\phi\in\Gamma}\phi \rightarrow \bigvee_{\psi\in\Delta}\psi$. So, to prove a sequent $\Gamma \vdash \Delta$, all of Γ is assumed to be true and from these one of Δ is shown to be true.

To prove a dL formula of the form (3.3), KeYmaera X first converts it into a sequent (goal) of the form

$$\vdash \textit{init} \rightarrow [(\textit{env};\ \textit{ctrl};\ \textit{plant})^*]\,(\textit{guarantee})$$

and then successively decomposes it into several smaller logical formulas (sub-goals) based on the proof rules of sequent calculus. To prove properties of loops as in (3.3), the loop invariant proof rule [72] can be used. HP statements in the dL formula are symbolically executed to describe their effect based on the respective proof rules in such a decomposition. A proof of the desired goal is finished when all the sub-goals are proven to be valid.

SPARK refers to both a programming language and an associated set of tools to perform formal verification on that language. Therefore, from the deductive verification workflow in Figure 3.2, program and specification correspond to the SPARK source code. The formal verification toolset in SPARK can analyse the source code on two different levels, flow analysis and proofs.

Flow analysis capabilities ensure the program correctness with respect to data flow and information flow. Errors arising due to uninitialized variables, data dependencies between inputs and outputs of subprograms, well-formedness of programs, etc., are checked by this level of analysis. The second level of analysis is to perform automated proofs to check for runtime errors and conformance of the program with respect to user-defined specifications. As described in Section 3.1, user-defined specifications in SPARK are provided in the form of program annotations such as pre- and post-conditions. The program annotations specified are used to generate verification conditions, which can then be discharged using the proof tools to show program correctness. The verification conditions are generated in the form of a list of hypotheses $H_1, H_2, \ldots, H_n$ and conclusions $C_1, C_2, \ldots, C_m$. The goal is to infer the conclusions from the hypotheses, i.e., to prove $(H_1 \wedge H_2 \wedge \cdots \wedge H_n) \rightarrow (C_1 \wedge C_2 \wedge \cdots \wedge C_m)$.

“*What a mathematical proof actually does is show that certain conclusions... follow from certain premises... The validity of these premises is an entirely independent matter which can safely be left to philosophers.*”

Timothy Gowers[4]

CHAPTER 4

Safety Proofs for Automated Driving

The aim of this work is to investigate how formal methods can be used to formally verify DECISION & CONTROL subsystems and how formal proofs can be used as evidence in the safety argument of an ADS. The DECISION & CONTROL can be functionally decomposed based on many different architectural designs [86]–[88]; generally three kinds of functions:

- System management functions (henceforth referred as *state manager*) that are responsible for monitoring the health and status of the system like the ODD conditions, the different driving modes of the ADS feature, etc., and for making decisions like activation and deactivation of the ADS feature, appropriate response to a fault, etc.;
- Nominal planning and control functions (henceforth referred as *nominal controller*) that are responsible for performing the DDT during nominal driving conditions; and
- Safe planning and control functions (henceforth referred as *safety controller*) that are required to guarantee safety in critical driving situations (e.g. when a collision must be avoided).

[4]Gowers, T. (2002). Mathematics: A very short introduction. OUP Oxford.

Depending on the architectural design, there might be one or many state managers, the nominal and safety functions may be performed by the same functional element or by separate elements. Regardless of the structure, the functions included in the Decision & Control have certain characteristics. They make decisions at both the tactical and the operational level and the decisions are based on information from the sense subsystem (see Figure 1.1) that perceives the environment; they have to account for uncertainties in the prediction and assessment of the traffic situation; and they include a reactive feedback control mechanism to perform the DDT and guarantee safety. The following sections describe the insights obtained from the included papers with respect to the research questions. Paper A and Paper B deal with the state manager part, Paper C and Paper D deal with the nominal and safety controller part, and Paper E deals with the Decision & Control subsystem as a whole.

4.1 Specifying in the Large and in the Small

As mentioned in Chapter 3, formal specifications are at the heart of the formal verification process. A formal specification describes a requirement, and particularly in this context, a safety requirement.

A *safety requirement* (SR) is a requirement defined in order to avoid or reduce risk.

Paper A studies the Lateral State Manager (*LSM*), a system management component responsible for managing modes during an automated lane change. The correctness of *LSM* is verified with respect to SR 1 (*Req.1* from Paper A):

SR 1. *If changing lane, the lane change shall always be to the same side as indicated.*

In Paper C, a safety controller design is proved to guarantee SR 2 (FSR 1 from Paper C):

SR 2. *Decision & Control shall at all times output a safe acceleration request to avoid collision with any object in front.*

Safety requirements of an ADS typically describe (un)desired behaviour over time. A key insight obtained from the case studies in Paper A, Paper B, and

Paper C, is that unique challenges are associated with how a given ADS safety requirement can be formally specified in the different approaches. In Paper A, the safety requirement SR 1 was modelled as an EFSM, encoding the unsafe state as a blocking state. Verifying SR 1 is then done by checking whether the blocking state can be reached or not. The possibility to use guards and variables in an EFSM provided an efficient way to express SR 1 with just 2 locations, as shown in Figure 3 of Paper A.

In Paper B, on the other hand, where an FSM model of *LSM* was automatically learnt, it was not as easy to express the same specification. To efficiently learn the FSM, several reductions, such as merging variables in the original code to reduce the alphabet, had to be made. This improved the efficiency of the learning. However, expressing SR 1 turned out to be difficult, since it was no longer obvious how to model the unsafe state as a blocking behaviour in relation to the learned FSM.

In the model checking approach using TLA$^+$, SR 1 was expressed as an invariant property of the model using TLA constructs as shown in Paper A (A.2). Recall from Chapter 3 that invariants, which are safety properties in TLA$^+$, are linear-time properties and are expressed of the form

$$\Box P$$

where P is a state predicate. TLC checks an invariant by exploring all reachable states. Expressing SR 1 as an invariant property in TLA$^+$ is notably different from expressing it as a (non)blocking property in SCT. The nonblocking property can be translated into the branching-time temporal logic [82] formula:

$$\mathbf{AG}\,\mathbf{EF}\ \mathit{marked}$$

which expresses that across **A**ll computational paths **G**lobally, there **E**xists at least one path where **F**inally a marked state is reached. Admittedly, the possibility to specify and verify SR 1 through the nonblocking property and the invariant property does not provide enough evidence to suggest they are suitable to specify all types of safety requirements for a state manager function of Decision & Control. More empirical studies are needed in this regard.

With the deductive verification framework in SPARK, Paper A shows that using pre- and post-conditions to express SR 1 is significantly difficult. A primary reason for this is the presence of requirement gaps, i.e., gaps between the requirements at different levels of the development process. While SR 1 in its natural language form proved to be sufficient to work with model-based approaches like SCT and model checking, it had to be further broken down in the form of program annotations for meaningful analysis using the deductive framework in SPARK. This was inefficient because SR 1 puts a requirement on the *LSM* at the function design level, while the program annotations have to be specified at the source-code implementation level. The implementation independent SR 1 of the *LSM* had to be broken-down into implementation specific requirements in order to be efficiently formalized and verified using SPARK. However, it should not be overlooked that verification of implementation specific safety-critical properties such as division-by-zero, overflow, etc., is automatically done in SPARK with no additional effort.

The challenge with the presence of requirement gaps was also encountered in Paper C. Similar to SR 1, SR 2 also puts a requirement on the DECISION & CONTROL at the function design level. To prove that a safety controller guarantees the fulfilment of SR 2, a safety constraint given by the critical position-velocity pair $\langle x_c, v_c \rangle$ (see Paper C) was formulated. It was then proved using KeYmaera X that the safety controller always guarantees a safe acceleration request such that the automated vehicle does not have a velocity higher than v_c at or beyond x_c. Though the correctness of the design with respect to the safety constraint was formalized and proved in dL, the fact that fulfilling the safety constraint $\langle x_c, v_c \rangle$ implies that a collision is indeed avoided, as required by SR 2, was assumed and not proved in Paper C.

In Paper C, SR 2 was expressed as a dL formula of the form (3.1) where the box modality

$$[\alpha]\,(\mathit{guarantee}) \tag{4.1}$$

is used to express that *guarantee* is true in all final states reached by all executions of the hybrid program α. However, as discussed in Remark 4.3 of Paper C, care must be taken to avoid situations where *guarantee* is true in the final state but violated by some intermediate states in an execution of α. Such situations can be avoided either by making use of the modelling features in a hybrid program as done in Paper C, or by using differential temporal logic [89] which extends dL with temporal operators.

A safety argument of an ADS depends on the verification of the safety requirements at all levels, and also on the traceability of the lower level safety requirements to the higher level safety goals. Bergenhem *et al.* [55] advocate the importance of having formal proofs at each step of the requirement refinement process for the safety argument. Experience from Paper A and Paper C further strengthens this argument. To help tackle this challenge, Section 6 of Paper C shows how formal analysis, especially using dL, is used to systematically refine SR 2 into subsequent low-level requirements depending on the assumptions and invariant conditions required to guarantee safety. This introduces more rigour in the requirement refinement process with the help of formal analysis, and such a model-based approach can be used to achieve traceability in the safety argument. In that vein, Paper E proposes a safety argument approach that shows how the different artifacts obtained during a formal analysis can be used to close the gap between the ADS system requirements and the individual component requirements. An interesting future work would be to investigate whether dL and SPARK can be used together to show traceability from the design level to the implementation level.

4.2 Models – The Good, the Bad, and the Useful

The validity of a formal proof directly corresponds to the validity of the formal model. So, what would make a formal model and thereby the proof invalid? Insights from Paper A and Paper C reveal two main problems. First, manual errors introduced during the modelling process could lead to invalid conclusions. Second, modelling errors due to incorrect assumptions and abstractions in the formal model could result in proving a faulty controller safe. In both these cases, an ADS will be incorrectly verified to be safe, which could be catastrophic in practice. Unsurprisingly, these concerns are not specific to the case studies in Paper A and Paper C, but rather a general threat to safety arguments based on formal approaches [20], [65]. To address the first challenge, Paper B presents a systematic method to automatically learn formal models (discussed in Section 4.3); and to address the second challenge, Paper D considers two kinds of modeling errors and defines and proves conditions that when fulfilled ensure that these modelling errors are not present.

In Paper A and Paper B, a formal model of the *LSM* was obtained from the source code. Since the *LSM* is a state manager function, the hypothesis was that state-based modelling formalisms like EFSM and TLA$^+$ were naturally suitable. While it was sufficient to model and analyse the decision logic in the *LSM*, some shortcomings were identified in both Paper A and Paper B. For instance, it was observed that, to reason about certain decisions in the *LSM*, it was also necessary to sufficiently model the interactions of the *LSM* with other components in the DECISION & CONTROL (Figure 1 in Paper A). These interactions typically involve state variables that vary continuously with time, and in Paper A such interactions were abstracted as discrete decision variables that affect the state transitions (see Section 3 and Section 4 of Paper A). In Paper B, significantly more abstractions were needed in order to efficiently learn a model of the *LSM*.

Of course, different techniques [90], [91] have been investigated to deal with continuous change in the formalisms investigated in Paper A. However, it entails from the case studies in both Paper A and Paper B that adding finer abstractions of those aspects would likely result in the formal analysis becoming intractable due to the well known state-space explosion problem associated with finite-state methods like SCT and model checking. On the other hand, experience from Paper A suggests that such finite-state methods are useful in the concept and the early design phases of the ISO 26262 safety lifecycle [52].

Based on insights from Paper A, Paper C evaluates a modelling formalism that allows reasoning about both discrete changes as well as the continuous dynamics of the system. While the risk of modelling errors due improper abstractions were reduced in comparison to the methods in Paper A, modelling errors due to improper assumptions surfaced as a crucial challenge.

Paper C uses hybrid programs to model DECISION & CONTROL as a hybrid system. Hybrid programs include program statements to describe nondeterminism in many ways (see Section 3 of Paper C). The nondeterministic statements are particularly helpful in two aspects: (i) they can describe unknown behaviour of the environment and other components interacting with the DECISION & CONTROL; and (ii) they can abstract away implementation specific details and thus reduce the dependency of the proof on such details. Test actions are often used together with non-deterministic assignment, like $a_n := *;\ ?\left(-a_n^{min} \leq a_n \leq a_n^{max}\right)$. This expresses that a_n is assigned an arbitrary value, which is then tested to be within the bounds $-a_n^{min}$ and a_n^{max}. Such a model could, for instance, be used to describe the behaviour of a nominal controller which outputs a nominal acceleration a_n within the bounds. Thus, the behaviour of any nominal controller could be modelled independent of the implementation, which provides the flexibility to analyse and reason about DECISION & CONTROL even if learning-based algorithms are used in the nominal controller.

While it is undoubtedly a strength that hybrid programs can describe and reason about the typically infinite set of behaviours, it is also a weakness. The weakness stems from the fact that assumptions are used to limit the operational domain and remove behaviours that are too hostile for any ADS to act in a safe manner. As an example, consider an ADS designed to autonomously drive a vehicle in urban highways. Of course, it is possible to model an environment where obstacles are allowed to nondeterministically appear directly in front of the vehicle while driving. Admittedly, such a model is neither useful nor realistic to argue safety of the ADS. Therefore assumptions are necessary to restrict the operational domain, but, if care is not exercised, they might inadvertently result in a situation where the formal proof provides less assurance than what is assumed.

Theorem 1 of Paper D formulates and proves conditions to identify and avoid situations where the environment assumptions and the controller actions interact in such a way that the environment behaves in a friendly way to adapt to the actions of a controller that *exploits* that friendliness. An example of this is a faulty ADS controller that never brakes, together with an environment that reacts by always moving obstacles to accommodate the faulty actions. In the same way, Theorem 2 of Paper D addresses modelling errors that arise when the assumptions about the environment and/or other interacting components remove all behaviours in which any action by the controller is needed. A worst-case situation of this error is when the model is so over-constrained that all possible behaviours are removed, thereby resulting in a vacuous proof for the correctness of an *unchallenged* controller.

Such modelling errors also highlight the implicit trade-off between false alarms and vacuous truth in these model-based methods. While abstracting too many details increases the risk of false alarms, having a model with restrictive constraints could potentially end up with a proof that is vacuously true. In contrast, formal verification in SPARK is applied at the source-code level. Hence, such modelling errors are not a concern. However, other kinds of challenges are manifested as discussed in the next section.

4.3 Verification – Effort vs. Gain

As described in Chapter 3, the different methods and tools evaluated are grouped into two categories: enumeration and deduction. The verification effort and the expertise required in the two categories are noticeably different.

In Paper A, an existing implementation of the *LSM* was manually translated into an EFSM model (for SCT), and a TLA$^+$ model. Though there was considerable manual effort involved in the translation, it was not a major obstacle in this case. Both Supremica and TLC provided a counterexample to SR 1. It was then confirmed by reviews and simulation that the behaviour described by the counterexample was due to a bug in the implementation. Though the verification objective was met, it is foreseeable from the experience of Paper A that manually translating the models from source code does not scale well. This highlights the fact that these methods may not be best suited for verification at the implementation level.

Apart from the effort needed to translate the models, another significant challenge with the manual approach is the possibility to introduce modelling errors in the process. To overcome the problem of potentially error-prone manual modelling, Paper B presents one possible solution that automatically learns the formal models directly from the implemented code. Two learning algorithms were investigated in the case study: L^*, a language-based learning algorithm [92] and the Modular Plant Learner (MPL), a state-based learning algorithm [93]. While L^* did not manage to learn a model (despite several abstractions and experiments on two different tools), MPL did learn a model of the *LSM*. Although the learnt model was not formally verified (see Section 6 in Paper B), results from the case study make strong arguments for the applicability of automata learning as a method to address the challenges in manual modelling. However, as discussed in Paper B, more empirical studies are needed to further explore the limits of the learning approach.

In the case of SPARK, though, the *LSM* code in MATLAB was re-implemented using Ada to make use of the SPARK formal verification framework. The Ada implementation of the *LSM* was straightforward since the original MATLAB implementation was available for reference. However, specifying SR 1 in the form of program contracts required several iterations starting from a single global contract at the package level to incrementally adding many specific contracts at the subprogram level. Since SPARK could not prove some contracts, it was difficult to make a concrete conclusion as the proof checks only report that the unprovable post-conditions might fail but little feedback on why so. When a proof fails, SPARK may provide a counterexample with a path which highlights the subprogram statements and the assignment of values to variables that appear on that path (Section 7.2 in [94]). Since this was observed with every incremental addition of contracts, detailed analysis and domain knowledge were required to determine whether the failed proof attempts were due to an error in the code or due to an error in the program annotations. This further emphasises the need to address the gaps in formalizing requirements at all levels of the development process.

The need for domain knowledge and expertise in the proof process was also observed in the case of dL and KeYmaera X in Paper B. The effort in this case was targeted towards finding the correct assumptions and the loop invariants required to close the proof goals. This was also an incremental process, however, KeYmaera X allows user interactions [85] during the intermediate steps of the proof process. Analysing a particular proof branch in case of a failed proof attempt was helpful in strengthening the initial loop invariant candidates as well as for identifying the required assumptions in the model. In contrast, though SPARK supports user interaction by the addition of manual proofs and lemmas (Section 7.9 in [94]), the level of interaction is quite involved and may require significant expertise to analyse the verification conditions as they are expressed in an intermediate language different from the SPARK programming language. On the other hand, the feedback from KeYmaera X in the form of a counterexample was very much similar to SPARK. Wherever available, the counterexample only provides variable assignments that cause the formula to be invalid.

A fundamental trade-off exists between the level of automation and the expressiveness of the underlying logical formalism. dL and KeYmaera X allow modelling of and reasoning about infinite state systems over infinite sets of behaviours that include both discrete and continuous dynamics. The price to pay here is the user expertise and domain knowledge needed to prove the requirements. Whereas, SCT and TLA$^+$ provide full automation but at the price of discrete and finite-state abstractions of the system. The counterexamples provided by SUPREMICA and TLC include information about the variable assignments as well as the trace leading to the property violation. This is considerably different from what KeYmaera X and SPARK provides as counterexamples. Though manual inspection is needed in all cases, in the case of Supremica and TLC, the objective is to validate that the counterexample is not due to false alarms, for which help can be obtained from other techniques like testing and simulation. This distinction could be valuable when the type of evidence and verification feedback desired during the development process is important.

4.4 Towards an Integrated Approach

An important take-away message from this thesis is the need for an integrated approach to aid the development of safe ADS. Papers A, B, C, and D give insights into the advantages and limitations of the different approaches. It is also clear that no method can on its own provide sufficient evidence for the safety argument of an ADS.

From Paper A, Paper B, and Paper C, there is strong evidence to suggest that SCT (Supremica), TLA$^+$ (TLC), and dL (KeYmaera X) are best suited to model, specify, and verify Decision & Control of an ADS in the conceptual stages of the development process. Typical activities include feature design specification, evaluation of different conceptual designs, requirement refinement, hazard analysis (for related work, see Paper G and Paper I), design and verification of the functional safety concept [52]. Paper A indicates that SCT and TLA$^+$ include modelling features suitable to reason about the system management functions, and Paper C indicates that dL is preferable for the nominal and safety controller functions of Decision & Control. These three methods and their associated tools also seem to support an iterative design process in which small design modifications do not add any significant overhead in the effort to use the methods and tools.

On the other hand, the use of such methods in the early stages of the development does not preclude the need for rigour in the implementation stages. As seen in Paper A, the deductive framework in SPARK is best suited to verify source code at the implementation level. Similarly, the automata learning approach from Paper B could also be suitable to learn formal models from the source code. Certainly, there are other ways, like strict coding guidelines and standards, that can be combined with traditional software testing to assess the quality of the implemented source code. In the end, the question of whether evidence from such arguments meets the rigour provided by a formal proof will be decisive. Such a distinction between the various approaches clearly highlights the importance of the verification objective as a factor that affects the choice of the formal method. Though not investigated in this thesis, an interesting approach is to combine several methods to design a safe ADS, that is, to use SCT with SPARK for a state manger and dL with SPARK for a safety controller for development end-to-end from design to implementation.

Gleirscher *et al.* [95] report that a "striking finding" in their experience is that, in various safety-critical standards, the normative parts for development activities like requirements engineering (both specifying and avoiding errors), and for hazard and risk analysis, is below par. This, despite observations that specification errors is a significant cause of safety-critical incidents. Furthermore, several studies [65], [95], [96] agree that choosing an appropriate modelling and specification method for a particular system is difficult, and there is a lack of guidelines to aid engineers and developers in that task. Insights from this thesis help narrow the gap in both the above aspects.

Another factor that affects the industrial adoption of the investigated methods is their ease of integration with the existing software development processes. With recent advances in continuous software development [97], [98], the need for automated development tools that are seamlessly integrated into the software development tool chain is increased. Such new developments in the processes have raised concerns for the development of safety-critical systems [99], [100]. Any progress in developing tool support for continuous reasoning [101] for these methods will likely increase the industrial adoption. In addition, the compatibility of the formal tools with other conventional development tools is equally important. SPARK provides features to combine formal verification and testing (Section 8 in [94]). The automata learning approach in Paper B has the possibility to be used in a learning based test environment [102]. In the case of model-based tools, promising results are shown along the lines of combining SUPREMICA (SCT) with simulation-based analysis (Paper I) and also through the use of runtime monitoring [103] based on KeYmaera X models.

4.5 Safe Automated Driving – The Big Picture

The previous sections discussed the insights obtained from formally verifying parts of the DECISION & CONTROL subsystem in an ADS. However, **RQ 3** still remains: how do the evaluated methods, or rather, formal methods in general aid in the overall safety argument of an ADS? In this regard, Paper E presents one way to structure a safety argument for an ADS based on formal methods. Recall from Chapter 2 that a safety case is a structured argument supported by evidence to show that a system is safe in a given operational environment.

The *Goal Structuring Notation* (GSN) is a standardized graphical argument notation [104], [105], which can be used to structure a safety case. It explicitly documents the individual elements of a safety argument and their relationships to the gathered evidence. Figure 3 in Paper E shows the GSN structure of the proposed safety argument approach using the different formal artifacts.

In a nutshell, Paper E proposes to split the safety argument using two strategies that taken together demonstrate that an ADS fulfills a safety goal (e.g. does not cause a collision). The first strategy argues that a formal model of the ADS is correct with respect to the specification, i.e., the formalized safety goal, using the formal proof of correctness as evidence. The formal model is composed of the different subsystems of the ADS as shown in Figure 1.1. The formal models are then used to obtain specifications for the realization of the individual subsystems. The second strategy then argues from evidence that the different subsystems are correct with respect to the specifications resulting from the first strategy. Thus, through the evidence from these two strategies together, the safety goal is shown to be fulfilled.

The artifacts produced from model-based approaches using SCT, TLA$^+$, and dL are well-suited to be a part of the first strategy. For example, Paper C proves that a Decision & Control model fulfills SR 2. This proof is indeed the evidence needed for the first strategy. The formal model includes a description of the Decision & Control subsystem as well as the invariant conditions and assumptions on the other subsystems of the ADS. These artifacts are then used, as part of the second strategy, to obtain requirements on the individual components. An example of the safety argument evidence in this strategy, is a formal proof that a SPARK implementation of the safety controller is correct with respect to the requirements resulting from the proven model.

As discussed in Paper E, the proposed approach is just one way of structuring a safety argument using the different formal artifacts. A potential benefit, however, is that this approach not only ensures that unambiguous requirements are obtained on the different components, but also lessens the effort of verifying the safety requirement of the ADS. This is primarily due to the separation of concerns in the proposed argument. A safety requirement that initially required closed-loop verification of the whole ADS feature can now be shown to be fulfilled by verifying the individual component requirements, since the evidence that the individual component requirements imply the original requirement is already available as a by-product from the proof of correctness of the formal model from the first strategy. A critical aspect of the proposed strategy, not considered in this thesis, is an argument to show the correctness of the formal tools used to produce the evidence. Habli and Kelly [106] present a systematic approach in that regard.

“*Whatever you say may fade away*
Whatever you write might come back and bite”

David V. Thiel[5]

CHAPTER 5

Summary of Included Papers

This chapter provides a summary of the included papers.

5.1 Paper A

Yuvaraj Selvaraj, Wolfgang Ahrendt, Martin Fabian
Verification of Decision Making Software in an Autonomous Vehicle: An Industrial Case Study
Larsen K., Willemse T. (eds) Formal Methods for Industrial Critical Systems. FMICS 2019. Lecture Notes in Computer Science, vol 11687, pp. 143–159, Jul. 2019.
©Springer, Cham DOI: 10.1007/978-3-030-27008-7_9.

In Paper A, three different formal verification approaches, namely supervisory control theory, model checking, and deductive verification are used to formally verify an existing decision-making software in an autonomous vehicle. The three approaches are evaluated to identify (i) the challenges in applying formal verification to automated driving, and (ii) the factors that affect the choice of

[5]Thiel, D. V. (2014). Research methods for engineers. Cambridge University Press.

the verification method. Paper A discusses how the verification objective differs in the three approaches and presents the challenges in formally modelling and specifying the decision-making software. Insights from Paper A show the need for an integrated formal approach to prove correctness.

5.2 Paper B

> **Yuvaraj Selvaraj**, Ashfaq Farooqui, Ghazaleh Panahandeh, Wolfgang Ahrendt, Martin Fabian
> Automatically Learning Formal Models from Autonomous Driving Software
> *Electronics 2022; vol. 11, no. 4: 643.*
> ©The Authors DOI: 10.3390/electronics11040643.

One of the challenges identified in Paper A is the manual effort required in obtaining formal models. Manual construction of formal models is expensive, error-prone, and intractable for large systems. As one possible solution to this problem, Paper B applies active automata learning techniques to obtain formal models of the decision-making software studied in Paper A. Results from Paper B demonstrate the feasibility of such automated techniques for automotive industrial use. Two learning algorithms are evaluated and practical challenges in their application are presented. Furthermore, insights from Paper B show that such techniques could potentially pave way for the widespread adoption of formal methods to guarantee correctness of automated driving systems.

5.3 Paper C

> **Yuvaraj Selvaraj**, Wolfgang Ahrendt, Martin Fabian
> Formal Development of Safe Automated Driving using Differential Dynamic Logic
> *IEEE Transactions on Intelligent Vehicles*, 2022
> ©IEEE DOI: 10.1109/TIV.2022.3204574.

Paper C investigates the use of differential dynamic logic and the deductive verification tool KeYmaera X in the development of an automated driving feature. Specifically, this paper demonstrates how formal models and safety proofs of different design variants of a Decision & Control module can be used in the safety argument of an in-lane automated driving feature. In doing so, the assumptions and invariant conditions necessary to guarantee safety are identified, and this paper shows how such an analysis helps in the requirement refinement and in the formulation of the operational design domain during the development process. Furthermore, Paper C also illustrates how the performance of the different models is formally analyzed exhaustively, in *all* their possible behaviors.

5.4 Paper D

Yuvaraj Selvaraj, Jonas Krook, Wolfgang Ahrendt, Martin Fabian
On How to Not Prove Faulty Controllers Safe in Differential Dynamic Logic
Riesco, A., Zhang, M. (eds) Formal Methods and Software Engineering. ICFEM 2022. Lecture Notes in Computer Science, vol 13478, pp. 281–297, Oct. 2022. ©Springer, Cham DOI: 10.1007/978-3-031-17244-1_17.

Though formal methods have shown their usefulness, care must be taken as modelling errors might result in proving a faulty controller safe, which is potentially catastrophic in practice. Paper D deals with two such modelling errors in *differential dynamic logic.* The main contribution is to prove that, when certain conditions hold, these two modeling errors cannot cause a faulty controller to be proven safe. The problems are illustrated with a real world example of a safety controller for automated driving, and it is shown that the formulated conditions have the intended effect both for a faulty and a correct controller. It is also shown how the formulated conditions aid in finding a *loop invariant* candidate to prove properties of hybrid systems with feedback loops. The results are proven using the interactive theorem prover KeYmaera X.

5.5 Paper E

Jonas Krook, **Yuvaraj Selvaraj**, Wolfgang Ahrendt, Martin Fabian
A Formal Methods Approach to Provide Evidence in Automated Driving Safety Cases
Submitted to IEEE Transactions on Intelligent Vehicles, 2022.

Paper E proposes a formal methods based approach to structure a safety argument for an ADS feature. The main contribution is to show how formal methods can aid in providing evidence to the safety argument. Furthermore, if a proof is obtained, Paper E demonstrates how the proven formal models can help to close the gap between the ADS system requirements and the broken-down subsystem requirements. This structure of the safety argumentation can be used to alleviate the need for reviews and tests to ensure that the break-down is correct, thereby saving effort both in data collection and verification time.

“*That’s all folks!*”

Porky Pig[6]

CHAPTER 6

Concluding Remarks and Future Work

This thesis started with the hypothesis that formal methods, especially formal verification, can be used to prove correctness of DECISION & CONTROL systems and thereby provide rigorous evidence for the safety argument of ADS. The results presented in the thesis strongly support that hypothesis. To achieve the overall goal of establishing formal verification as an efficient tool in the development of a safe ADS, three research questions are considered. Based on the insights from the included papers and the discussion in Chapter 4, the research questions are answered as follows.

RQ 1 What factors affect the application of formal verification to ADS and what are the current challenges in existing methods?

To answer **RQ 1**, three aspects are identified. First, deciding the right level of abstraction and the choice of the formal method for that level is a critical aspect but also a *vague* one with no clear answer. The formal methods investigated in this thesis are fundamentally targeted at certain types of systems. SCT towards DES, TLA$^+$ towards concurrent and reactive systems, dL towards

[6] Looney Tunes

cyber-physical systems, and SPARK towards safety-critical software. The Decision & Control system in an ADS includes all the above characteristics and more. The right level of abstraction needed to describe the behaviour of the Decision & Control system with each method have to be balanced with the kind of modelling errors that might arise as an unintended consequence.

Second, the verification objective is a crucial aspect that affects the practical applicability of the different methods and their associated tools. Similar to how the different methods are intended to model specific kinds of systems, they also have fundamental differences in how a proof of correctness is produced. A notable difference between the methods in the verification process is where human expertise and domain knowledge is required. The kind of evidence (if successful) and the level of feedback (if failed) provided by the methods are also closely connected to the verification objective. The question to consider here is whether quick automated feedback is required or a comprehensive proof of a wide variety (typically infinite set) of behaviours of the system. Also, the interoperability of the tools with other conventional development methods such as testing and simulation also affects the applicability of these methods.

Third, the need for sufficiently detailed requirements at all levels of the development process is identified as an important factor but also a barrier for practical verification. An insight from different case studies in this thesis is that there might not be a straightforward way to formalize a high level safety requirement (e.g. ADS does not cause a collision) in all the methods. The introduction of more rigour in the requirement refinement process to correctly break down the high level safety requirement into requirements that can be formalized and verified in the method of choice is vital.

RQ 2 How can the challenges be addressed, and can the solutions be scaled?

The second research question, **RQ 2**, concerns how to address the challenges identified in **RQ 1**. As regards the challenge with the formal modelling process, this thesis proposes solutions in two different directions. To address the potentially error-prone manual modelling process, Paper B presents a systematic method to automatically learn formal models directly from source code. Issues with scalability are identified and possible future investigations are also discussed. In the other direction, Paper D formulates and proves two theorems that address two kinds of modelling errors introduced due to improper assumptions in the models. This reduces the risk of fallible proofs in the safety argument.

Based on the included papers, this thesis identifies the need for an integrated approach to develop a safe ADS. The case studies of Paper A and Paper C provide guidelines on how to model and verify certain parts of Decision & Control using the different methods. Since some methods are well-suited at a specific phase of the development process, the case studies can be used to identify the right method and tool based on the level of abstraction and the verification objective at that level. Though the possibility to combine the different methods to develop a safe ADS is promising, much work is needed in that direction to establish a suitable framework that supports such development.

Paper C and Paper E discuss how formal methods can be used to break down a high-level system safety requirement into low-level subsystem requirements that can be formalized and verified. More empirical case studies are required to understand the limitations of this approach. However, it should be noted that the process of formalization of requirements can itself be beneficial.

RQ 3 How can formal methods be used to provide evidence in the safety argument of ADS?

Paper E presents a safety argument approach that uses formal methods to provide the necessary evidence. Of course, Paper E in isolation is not enough to answer **RQ 3**. The suitability of a formal method and the type of evidence generated is crucial for the argument, and Chapter 4 discusses how the evaluated methods fit into such an approach. It is not surprising that answering **RQ 3** is tightly connected to **RQ 1** and **RQ 2** and perhaps not surprising that the important factors that affect the formal verification of an ADS also have associated challenges. The strength of each method comes with a weakness. This, together with the heterogeneous and complex nature of ADS makes it a strenuous problem to solve. Efforts have to be targeted towards clearly defining the boundaries of the problem so that the strength of each method can be maximised and the weakness minimised, and this thesis provides valuable insights in that direction.

6.1 Future Work

The goal of developing a safe ADS and demonstrating that it is safe is a huge challenge and this thesis is just one step towards that goal. The scope for possible future research is wide. The case studies in this thesis are broad in the sense that different methods are evaluated on representative parts of Decision & Control in ADS. It is definitely possible to advance research by aiming for more studies that focus on deeper understanding. The research question in this direction, for instance, would be to investigate whether a particular method (e.g. SCT) is suitable to reason about all kinds of state manager functions in an ADS. In the same vein, more studies can be made in regard to the work in Paper D. A relevant research question is: what other modelling errors may cause a formal proof to fall apart and how to address them?

The possibilities to integrate different methods evaluated in this thesis, with each other and with other methods, to develop a safe ADS is a relatively less explored area. Invariants play a crucial role in safety verification and this was also observed in multiple methods studied in this thesis. Identifying invariants of a system is a challenging problem researched (independently) within the formal methods community as well as the control engineering community. Investigating whether results from those communities can be integrated in the context of ADS safety verification will be interesting.

In Paper B, a learning-based approach is studied to reduce the effort required in the verification of ADS where finite-state methods are used. Similarly, it is equally interesting to investigate whether learning-based approaches could reduce the effort in the deductive methods. Chapter 4 briefly discusses how the approach in Paper C can be used to guarantee safety even if learning-based algorithms are used for nominal planning in Decision & Control. However, this thesis does not investigate how formal methods can be used to reason about specific learning algorithms used for decision-making in an ADS. Any future work in this direction is definitely valuable since safety verification of learning-based algorithms in general is an open problem.

The safety argument approach to answer **RQ 3** is proposed based on the experience from the papers included in the thesis. Though Paper E presents arguments on how formal methods can provide compelling evidence, it is but an argument. Investigating the limits of the argument through concrete examples from the industry will likely remove some of the barriers for the industrial adoption of formal methods.

References

[1] “Global status report on road safety 2018.” (2018), [Online]. Available: https://www.who.int/publications/i/item/9789241565684.

[2] World Health Organization (WHO). “Road traffic injuries.” (2022), [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries (visited on Aug. 2, 2022).

[3] National Highway Traffic Safety Administration (NHTSA). “Automated Vehicles for Safety.” (2022), [Online]. Available: https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety (visited on Aug. 2, 2022).

[4] *Regulation (EU) 2019/2144 of the European Parliament and of the Council on type-approval requirements for motor vehicles and their trailers, and systems, components and separate technical units intended for such vehicles, as regards their general safety and the protection of vehicle occupants and vulnerable road users*, URL: http://data.europa.eu/eli/reg/2019/2144/oj, Dec. 2019.

[5] S. Singh, *Critical reasons for crashes investigated in the national motor vehicle crash causation survey*, https://crashstats.nhtsa.dot.gov/Api/Public/Publication/812506, Traffic Safety Facts Crash • Stats. Report No. DOT HS 812 506, Washington, DC:National Highway Traffic Safety Administration, Mar. 2018.

[6] T. Litman, *Autonomous vehicle implementation predictions*. Victoria, Canada: Victoria Transport Policy Institute, 2022, https://www.vtpi.org/avip.pdf.

[7] SAE J3016_202104, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," SAE Int., Tech. Rep., Apr. 2021.

[8] SAE International. "SAE J3016 Visual Chart." (2019), [Online]. Available: https://www.sae.org/binaries/content/assets/cm/content/blog/sae-j3016-visual-chart_5.3.21.pdf (visited on Aug. 2, 2022).

[9] Mitsubishi Motors. "Mitsubishi Motors Develops "New Driver Support System"." (1998), [Online]. Available: https://www.mitsubishi-motors.com/en/corporate/pressrelease/corporate/detail429.html (visited on Aug. 2, 2022).

[10] P. Bhatia, *Vehicle technologies to improve performance and safety*, https://escholarship.org/uc/item/4zw4m05k, University of California Transportation Center, UC Berkley, 2003.

[11] J. Jeffs. "A History of ADAS: Emergence to Essential." (2022), [Online]. Available: https://www.idtechex.com/it/research-article/a-history-of-adas-emergence-to-essential/25592 (visited on Aug. 2, 2022).

[12] K. Shirokinskiy, W. Bernhart, and S. Keese. "Adanced driver-assistance systems: A ubiquitous technology for the future of vehicles." (2021), [Online]. Available: https://www.rolandberger.com/en/Insights/Publications/Advanced-Driver-Assistance-Systems-A-ubiquitous-technology-for-the-future-of.html (visited on Aug. 2, 2022).

[13] Mercedes-Benz Group. "First internationally valid system approval for conditionally automated driving." (2021), [Online]. Available: https://group.mercedes-benz.com/innovation/product-innovation/autonomous-driving/system-approval-for-conditionally-automated-driving.html (visited on Aug. 2, 2022).

[14] M. Harley. "Testing (And Trusting) Mercedes-Benz Level 3 Drive Pilot In Germany." (2022), [Online]. Available: https://www.forbes.com/sites/michaelharley/2022/08/02/testing-and-trusting-mercedes-benz-level-3-drive-pilot-in-germany/?sh=2ec90cda366e (visited on Aug. 2, 2022).

[15] "WAYMO ONE," [Online]. Available: https://waymo.com/waymo-one/ (visited on Aug. 2, 2022).

[16] MIT Technology Review. "The three challenges keeping cars from being fully autonomous." (2019), [Online]. Available: https://www.technologyreview.com/2019/04/23/103181/the-three-challenges-keeping-cars-from-being-fully-autonomous/ (visited on Aug. 2, 2022).

[17] N. Kalra and S. M. Paddock, "Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?" *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016.

[18] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016.

[19] P. Koopman and M. Wagner, "Autonomous vehicle safety: An interdisciplinary challenge," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 1, pp. 90–96, 2017.

[20] P. Koopman, A. Kane, and J. Black, "Credible autonomy safety argumentation," in *27th Safety-Critical Sys. Symp. Safety-Critical Systems Club, Bristol, UK*, 2019.

[21] S. Riedmaier, T. Ponn, D. Ludwig, B. Schick, and F. Diermeyer, "Survey on scenario-based safety assessment of automated vehicles," *IEEE access*, vol. 8, pp. 87 456–87 477, 2020.

[22] J. A. Michon, "A critical view of driver behavior models: What do we know, what should we do?" In *Human behavior and traffic safety*, Springer, 1985, pp. 485–524.

[23] E. Gat, R. P. Bonnasso, R. Murphy, *et al.*, "On three-layer architectures," *Artificial intelligence and mobile robots*, vol. 195, p. 210, 1998.

[24] The Insurance Institute for Highway Safety. “Fatality Facts 2020.” (2022), [Online]. Available: https://www.iihs.org/topics/fatality-statistics/detail/state-by-state (visited on Aug. 2, 2022).

[25] Philip Koopman. “Potentially deadly automotive software defects.” (2018), [Online]. Available: https://betterembsw.blogspot.com/p/potentially-deadly-automotive-software.html (visited on Oct. 23, 2022).

[26] Sibros. “The current state of automotive software related recalls.” (2020), [Online]. Available: https://www.sibros.com/post/the-current-state-of-automotive-software-related-recalls (visited on Aug. 2, 2022).

[27] T. Krisher. “Tesla’s Autopilot system faces increased scrutiny after 3 crashes, 3 deaths.” (2020), [Online]. Available: https://globalnews.ca/news/6363342/tesla-autopilot-crashes-deaths/ (visited on Aug. 6, 2022).

[28] The NewYork Times. “Tesla Autopilot system found probably at fault in 2018 crash.” (2020), [Online]. Available: https://www.nytimes.com/2020/02/25/business/tesla-autopilot-ntsb.html (visited on Aug. 6, 2022).

[29] T. Krisher and O. R. Rodriguez. “NTSB releases details in 2 crashes involving Tesla Autopilot.” (2020), [Online]. Available: https://apnews.com/article/e2eabac31019a5a750dc8ef67791f62b (visited on Aug. 6, 2022).

[30] D. Shepardson. “Uber self-driving cars were involved in 37 crashes before a fatal incident.” (2019), [Online]. Available: https://www.businessinsider.com/uber-test-vehicles-involved-in-37-crashes-before-fatal-self-driving-incident-2019-11?r=US&IR=T (visited on Aug. 6, 2022).

[31] A. J. Hawkins. “Serious safety lapses led to Uber’s fatal self-driving crash, new documents suggest.” (2019), [Online]. Available: https://www.theverge.com/2019/11/6/20951385/uber-self-driving-crash-death-reason-ntsb-dcouments (visited on Aug. 6, 2022).

[32] M. Blanco, J. Atwood, S. M. Russell, T. Trimble, J. A. McClafferty, and M. A. Perez, "Automated vehicle crash rate comparison using naturalistic data," Virginia Tech Transportation Institute, Tech. Rep., 2016, https://vtechworks.lib.vt.edu/handle/10919/64420.

[33] WIRED. "Google's Self-Driving Car Caused Its First Crash." (2016), [Online]. Available: https://www.wired.com/2016/02/googles-self-driving-car-may-caused-first-crash/ (visited on Oct. 23, 2022).

[34] J. Souyris, V. Wiels, D. Delmas, and H. Delseny, "Formal verification of avionics software products," in *International symposium on formal methods*, Springer, 2009, pp. 532–546.

[35] A. Platzer and E. M. Clarke, "Formal verification of curved flight collision avoidance maneuvers: A case study," in *International Symposium on Formal Methods*, Springer, 2009, pp. 547–562.

[36] J.-B. Jeannin, K. Ghorbal, Y. Kouskoulas, *et al.*, "Formal verification of ACAS X, an industrial airborne collision avoidance system," in *2015 International Conference on Embedded Software (EMSOFT)*, IEEE, 2015, pp. 127–136.

[37] C. Bernardeschi, A. Fantechi, S. Gnesi, S. Larosa, G. Mongardi, and D. Romano, "A formal verification environment for railway signaling system design," *Formal Methods in System Design*, vol. 12, no. 2, pp. 139–161, 1998.

[38] A. Platzer and J.-D. Quesel, "European train control system: A case study in formal verification," in *International Conference on Formal Engineering Methods*, Springer, 2009, pp. 246–265.

[39] M. Lawford and A. Wassyng, "Formal verification of nuclear systems: Past, present, and future," in *1st International Workshop on Critical Infrastructure Safety and Security (CrISS-DESSERT'11)*, vol. 1, 2011, pp. 43–51.

[40] G. Bahig and A. El-Kadi, "Formal verification of automotive design in compliance with ISO 26262 design verification guidelines," *IEEE Access*, vol. 5, pp. 4505–4516, 2017.

[41] V. Todorov, F. Boulanger, and S. Taha, “Formal verification of automotive embedded software,” in *Proceedings of the 6th Conference on Formal Methods in Software Engineering*, 2018, pp. 84–87.

[42] A. Zita, S. Mohajerani, and M. Fabian, “Application of formal verification to the lane change module of an autonomous vehicle,” in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, IEEE, 2017, pp. 932–937.

[43] J. Krook, L. Svensson, Y. Li, L. Feng, and M. Fabian, “Design and formal verification of a safe stop supervisor for an automated vehicle,” in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 5607–5613.

[44] S. M. Loos, A. Platzer, and L. Nistor, “Adaptive cruise control: Hybrid, distributed, and now formally verified,” in *International Symposium on Formal Methods*, Springer, 2011, pp. 42–56.

[45] D. E. Stokes, *Pasteur’s quadrant: Basic science and technological innovation.* Brookings Institution Press, 2011.

[46] K. Säfsten and M. Gustavsson, *Research methodology: For engineers and other problem-solvers*, 2020.

[47] H. J. Holz, A. Applin, B. Haberman, D. Joyce, H. Purchase, and C. Reed, “Research methods in computing: What are they, and how should we teach them?” In *Working group reports on ITiCSE on Innovation and technology in computer science education*, 2006, pp. 96–114.

[48] M. Alavi and P. Carlson, “A review of MIS research and disciplinary development,” *Journal of management information systems*, vol. 8, no. 4, pp. 45–62, 1992.

[49] C. E. Kendig, “What is proof of concept research and how does it generate epistemic and ethical categories for future scientific practice?” *Science and Engineering Ethics*, vol. 22, no. 3, pp. 735–753, 2016.

[50] UK Ministry of Defence, “Safety management requirements for defence systems,” *Defence Standard 00-56 Part 1 Issue 7*, 2017.

[51] IEC, *IEC 61508-4:2010-1 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 4: Definitions and abbreviations*, International Electrotechnical Commission, 2006.

[52] ISO 26262:2018, "Road vehicles – functional safety," International Organization for Standardization, Tech. Rep., Dec. 2018.

[53] RTCA, *Software considerations in airborne systems and equipment certification*. RTCA, Incorporated, 2012, https://en.wikipedia.org/wiki/DO-178C.

[54] International Electrotechnical Commission, "IEC 62279: Railway applications—Communications, signalling and processing systems—Software for railway control and protection systems," *International Electrotechnical Commission: Geneva, Switzerland*, 2002.

[55] C. Bergenhem, R. Johansson, A. Söderberg, *et al.*, "How to reach complete safety requirement refinement for autonomous vehicles," in *CARS 2015-Critical Automotive applications: Robustness & Safety*, 2015.

[56] N. B. Ruparelia, "Software development lifecycle models," *ACM SIGSOFT Software Engineering Notes*, vol. 35, no. 3, pp. 8–13, 2010.

[57] ANSI/UL, *ANSI/UL 4600 - Standard for Evaluation of Autonomous Products*, https://ul.org/UL4600, 2020.

[58] ISO/PAS 21448:2019, "Road vehicles – safety of the intended functionality," International Organization for Standardization, Tech. Rep., Jan. 2019.

[59] G. E. Box, "Robustness in the strategy of scientific model building," in *Robustness in statistics*, Elsevier, 1979, pp. 201–236.

[60] J. Bowen. "Formal methods: Individual notations, methods and tools," [Online]. Available: https://formalmethods.wikia.org/wiki/Formal_methods (visited on Sep. 23, 2022).

[61] J. B. Almeida, M. J. Frade, J. S. Pinto, and S. M. De Sousa, *Rigorous software development: an introduction to program verification*. Springer Science & Business Media, 2011.

[62] G. J. Holzmann, "The model checker spin," *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.

[63] W. Ahrendt, B. Beckert, R. Bubel, R. Hähnle, P. H. Schmitt, and M. Ulbrich, "Deductive software verification–the KeY book," *Lecture notes in computer science*, vol. 10001, 2016.

[64] J. Guiochet, M. Machin, and H. Waeselynck, "Safety-critical advanced robots: A survey," *Robotics and Autonomous Systems*, vol. 94, pp. 43–52, 2017.

[65] M. Luckcuck, M. Farrell, L. A. Dennis, C. Dixon, and M. Fisher, "Formal specification and verification of autonomous robotic systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–41, 2019.

[66] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. New York, NY: Springer Science & Business Media, 2009.

[67] M. Skoldstam, K. Akesson, and M. Fabian, "Modeling of discrete event systems using finite automata with variables," in *2007 46th IEEE Conference on Decision and Control*, IEEE, 2007, pp. 3387–3392.

[68] L. Lamport, "The temporal logic of actions," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 16, no. 3, pp. 872–923, 1994.

[69] A. Pnueli, "The temporal logic of programs," in *18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, IEEE, 1977, pp. 46–57.

[70] L. Lamport, *Specifying systems: the TLA$^+$ language and tools for hardware and software engineers*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[71] A. Platzer, "Logics of dynamical systems," in *2012 27th Annual IEEE Symposium on Logic in Computer Science*, IEEE, 2012, pp. 13–24.

[72] A. Platzer, *Logical foundations of cyber-physical systems*. Cham, Switzerland: Springer, 2018, vol. 662.

[73] J. Barnes, *SPARK: The Proven Approach to High Integrity Software*. Altran Praxis, 2012.

[74] Barnes, John, *Programming in Ada 2012*. Cambridge University Press, 2014.

[75] L. Lamport, *The TLA$^+$ home page*, `https://lamport.azurewebsites.net/tla/tla.html`, Accessed: 2019-04-22.

[76] C. Dross and Y. Moy. "Introduction to SPARK." (2022), [Online]. Available: `https://learn.adacore.com/courses/intro-to-spark/index.html` (visited on Oct. 8, 2022).

[77] "SPARK 2014 reference manual," [Online]. Available: https://docs.adacore.com/spark2014-docs/html/lrm/index.html (visited on Oct. 8, 2022).

[78] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[79] R. Malik, "Programming a fast explicit conflict checker," in *2016 13th International Workshop on Discrete Event Systems (WODES)*, IEEE, 2016, pp. 438–443.

[80] S. Mohajerani, R. Malik, and M. Fabian, "A framework for compositional nonblocking verification of extended finite-state machines," *Discrete Event Dynamic Systems*, vol. 26, no. 1, pp. 33–84, 2016.

[81] R. Malik, K. Akesson, H. Flordal, and M. Fabian, "Supremica–An efficient tool for large-scale discrete event systems," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 5794–5799, 2017, 20th IFAC World Congress, ISSN: 2405-8963.

[82] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[83] Y. Yu, P. Manolios, and L. Lamport, "Model checking TLA$^+$ specifications," in *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, Springer, 1999, pp. 54–66.

[84] C. A. R. Hoare, "An axiomatic basis for computer programming," *Communications of the ACM*, vol. 12, no. 10, pp. 576–580, 583, Oct. 1969.

[85] N. Fulton, S. Mitsch, J.-D. Quesel, M. Völp, and A. Platzer, "KeYmaera X: An axiomatic tactical theorem prover for hybrid systems," in *International Conference on Automated Deduction*, Springer, 2015.

[86] S. Ulbrich, T. Menzel, A. Reschka, F. Schuldt, and M. Maurer, "Defining and substantiating the terms scene, situation, and scenario for automated driving," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, IEEE, 2015, pp. 982–988.

[87] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on intelligent transportation systems*, vol. 17, no. 4, pp. 1135–1145, 2015.

[88] S. Thrun, M. Montemerlo, H. Dahlkamp, *et al.*, “Stanley: The robot that won the DARPA grand challenge,” *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[89] A. Platzer, *Logical analysis of hybrid systems: proving theorems for complex dynamics*. Springer Science & Business Media, 2010.

[90] L. Lamport, “Hybrid systems in TLA$^+$,” in *Hybrid systems*, Springer, 1992, pp. 77–102.

[91] X. D. Koutsoukos, P. J. Antsaklis, J. A. Stiver, and M. D. Lemmon, “Supervisory control of hybrid systems,” *Proceedings of the IEEE*, vol. 88, no. 7, pp. 1026–1049, 2000.

[92] D. Angluin, “Learning regular sets from queries and counterexamples,” *Information and Computation*, vol. 75, no. 2, pp. 87–106, 1987.

[93] A. Farooqui, F. Hagebring, and M. Fabian, “Active learning of modular plant models,” *IFAC-PapersOnLine*, vol. 53, no. 4, pp. 296–302, 2020, 15th IFAC Workshop on Discrete Event Systems WODES 2020 — Rio de Janeiro, Brazil, 11-13 November 2020, ISSN: 2405-8963.

[94] “SPARK User’s guide,” [Online]. Available: https://docs.adacore.com/spark2014-docs/html/ug/index.html# (visited on Oct. 23, 2022).

[95] M. Gleirscher, S. Foster, and J. Woodcock, “New opportunities for integrated formal methods,” *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–36, 2019.

[96] E. A. Lee and M. Sirjani, “What good are models?” In *International Conference on Formal Aspects of Component Software*, Springer, 2018, pp. 3–31.

[97] J. Bosch, “Continuous software engineering: An introduction,” in *Continuous software engineering*, Springer, 2014, pp. 3–13.

[98] S. Vöst and S. Wagner, “Towards continuous integration and continuous delivery in the automotive industry,” *arXiv.1612.04139*, 2016.

[99] S. Vost and S. Wagner, “Keeping continuous deliveries safe,” in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, IEEE, 2017, pp. 259–261.

[100] R. Kasauli, E. Knauss, B. Kanagwa, A. Nilsson, and G. Calikli, “Safety-critical systems and agile development: A mapping study,” in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE, 2018.

[101] P. W. O’Hearn, “Continuous reasoning: Scaling the impact of formal methods,” in *Proceedings of the 33rd annual ACM/IEEE symposium on logic in computer science*, 2018, pp. 13–25.

[102] K. Meinke, F. Niu, and M. Sindhu, “Learning-based software testing: A tutorial,” in *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, Springer, 2011, pp. 200–219.

[103] S. Mitsch and A. Platzer, “ModelPlex: Verified runtime validation of verified cyber-physical system models,” *Formal Methods in System Design*, vol. 49, no. 1, pp. 33–74, 2016.

[104] T. Kelly and R. Weaver, “The goal structuring notation – a safety argument notation,” in *Int. Conf. Depend. Syst. Netw.*, Workshop Assur. Cases, Jul. 2004.

[105] ACWG, *Goal structuring notation community standard (version 3)*, `https://scsc.uk/r141C:1`, May 2021.

[106] I. Habli and T. Kelly, “A generic goal-based certification argument for the justification of formal analysis,” *Electron. Notes Theor. Comput. Sci.*, vol. 238, no. 4, pp. 27–39, Sep. 2009, First Workshop Certif. Saf.-Crit. Softw. Control. Syst. (SafeCert 2008).