CHALMERS
UNIVERSITY OF TECHNOLOGY

Safety assurance of autonomous vehicles using temporal logic in scenario-based simulations

Master's thesis in Master Programme Systems, control and mechatronics

Malin Edviken
Oskar Lundström

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Safety assurance of autonomous vehicles using temporal logic in scenario-based simulations

Malin Edviken & Oskar Lundström

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Automation
Chalmers University of Technology
Gothenburg, Sweden 2021

Safety assurance of autonomous vehicles using temporal logic in scenario based simulations
MALIN EDVIKEN
OSKAR LUNDSTRÖM

Industrial Supervisor: Zhennan Fei, Zenseact; Roozbeh Kianfar, Zenseact
Examiner and Supervisor: Sahar Mohajerani, Department of Electrical Engineering

Master's Thesis 2021
Department of Electrical Engineering
Automation
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A simulated cut-in scenario showing a counterexample found using a search method described in the thesis.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Safety assurance of autonomous vehicles using temporal logic in scenario-based simulations

MALIN EDVIKEN, OSKAR LUNDSTRÖM
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This thesis investigates the use of metric temporal logic specifications in the context of formal simulation as a means to verify functionality in autonomous vehicles. The techniques used are constrained to the capabilities of the chosen toolchain, since the focus of this thesis lies in how metric temporal logic can be used in the context. The toolchain used in the thesis includes the vehicle simulator Carla, a scenario definition language called SCENIC, as well as a verification toolbox called VerifAI. So the thesis includes two examples of safety requirements formulated in metric temporal logic as well as a demonstration of their use in scenario-based simulation. The results have shown that there is utility in using formal specification and verification techniques in the development of complex systems such as autonomous vehicles. Metric temporal logic has the potential to streamline the requirement specification, gives a temporal aspect to the evaluation of requirements and enables traceability from requirement to evaluation.

Keywords: Autonomous vehicle, Carla, Falsification, Metric temporal logic, Safety, SCENIC, Simulation, VerifAI, Verification.

Acknowledgements

First of all, we would like to thank Zhennan Fei, Sahar Mohajerani and Roozbeh Kianfar for giving us the opportunity to work on this master thesis at Zenseact and Chalmers University of Technology. Although it sometimes has been very challenging, it has also been very developing and a tremendous learning experience. Thank to all people at Zenseact for be so welcoming and especially thanks to Zhennan Fei, Jonathan Leiditz, Niels Berger, Tobias Karlsson and Lars Ljungberg for your technical help and useful discussions. Furthermore, we would like to thank our examiner and supervisor Sahar Mohajerani for all support, believing in us and all time you spent on this master thesis.

Finally, we would like to take a moment to thank our families for always being close, despite the physical distance between us. You have always been by our sides during all these years in Gothenburg. Without your cheers, support and love, we would never have made it this far.

Malin Edviken & Oskar Lundström, Gothenburg, June 2021

Contents

List of figures

List of tables

1 Introduction

Self-driving, or autonomous vehicles (AVs) is an emerging solution to several problems in society, such as fatal accidents and traffic congestion [1], [2]. In addition, AV:s could also bring back valuable time to the driver in cases when driving is no longer a pleasant task [2], [3]. Over the last decades, industry and academia have spent enormous effort and resources towards development of AVs, which can serve their intended purposes. One of the key challenges in enabling such technology is to ensure that AV:s are safe and do not cause accidents because of a malfunction or lacking functionality [4]. However, systems such as AV:s are very complex and depend on many different techniques, ranging from machine learning to model-based approaches. For that reason, verification of an AV is not a trivial task [1].

1.1 Background

Some background regarding the functionality of AVs is needed to understand the different aspects of verification of its safety.

1.1.1 Functionality of AVs

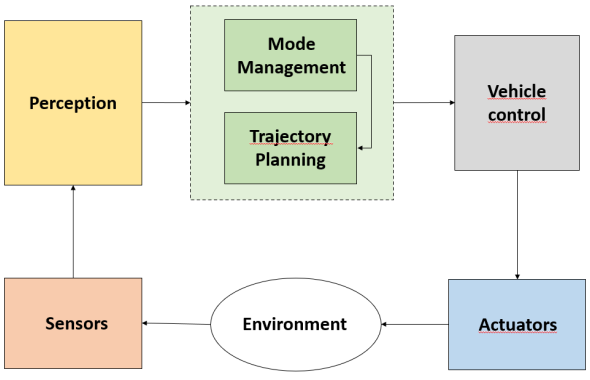

Figure 1.1: A brief overview of the structure for an AV-system.

AV-systems need to solve the complex task of navigating traffic, with other agents, safely. This task can be broken down into general sub-tasks, solved by several sub-modules, see Fig. 1.1. To capture information of the surrounding environment a perception module is needed. The perception module needs to include tracking

functionality to provide information about the environment, such as the locations of other vehicles or traffic signs. The output data from the perception module is passed to a trajectory planning module. The perception module also outputs tracking data so that the mode management module can recognize threats and set appropriate constraints on the trajectory. The mode management module determines a driving mode, such as a lane change, as a response to an event, such as a vehicle appearing in the current lane. The trajectory planning module then computes a safe trajectory in accordance with the current mode of the vehicle [5] [6].

In this thesis an AV-system from the company Zenseact was used. Two components are of interest, the trajectory planning module and the mode management module. The trajectory planning module bases its computations on three main signals. One signal is the tracking data from the perception system, the signal describes identified objects, their size as well as tracking information such as position, velocity and acceleration. The second signal is the dynamic states of the ego vehicle: position, velocity, acceleration, roll rate, yaw rate and pitch rate. The constant parameters of the ego vehicle are also defined to the trajectory planner, since it requires the dimensions of the ego vehicle to compute what paths are safe. The third main signal that the trajectory planner uses is the description of the road, in the form of a road model. The road model describes the shape and size of the current lane as well as any adjacent lanes. The trajectory planning module then computes a path limited by specified comfort constraints and consisting of a requested acceleration as well as a requested lateral path.

The mode management module uses the same inputs as the trajectory planning module to determine whether the vehicle is in a safe situation or not. In normal circumstances the path requests from the trajectory planner are passed to the actuation module directly. If a threat is identified the mode manager has to arbitrate between the trajectory planner request and an emergency request, and choose the appropriate one to pass through to the actuation module.

1.1.2 Safety of AVs

A central aspect of autonomous driving is safety. The standard ISO26262 covers functional safety. Functional safety refers to the absence of unreasonable risk due to hazards caused by system failure [4]. However ISO26262 does not ensure that intended functionality is achieved. Safety of the Intended Functionality (SOTIF), or ISO21448, was crafted with the purpose of covering safety hazards that result without system failure [7]. Riedmaier et al. state that the greatest challenge in safety assessment is that a trafficked road is an open parameter space where an infinite number of different traffic situations can occur. Thus, proof of absolute safety is not possible, the research is made to assess the system's safety with various methods [8].

Many different verification approaches and techniques exist in system verification. The most common verification techniques for AV-systems are simulation, testing,

and formal methods. However, these verification techniques are often used independently of each other [1]. For example, the mode management module makes use of formal verification techniques. Mode management modules have well defined discrete states and can thus be described by a formal model, which in turn can be proven to never reach undesirable states. That option does not exist when verifying the trajectory planning module. Since the trajectory planning module solves problems in continuous domains with complex and non-linear dynamics it can not be modelled formally. Instead it is often evaluated by brute force, letting the module perform in closed-loop simulation and then using the performance as safety assurance [1]. In contrast, verification of the perception module heavily relies on extensive testing and re-simulation of data, evaluating the predictions of inputs to a ground truth [9]. A more general, requirement-based and systematic approach to verification of the different modules would give benefits such as a more standardized process for evaluating the complete autonomous system, with a very explicit connection to the requirements defined by the stake-holders.

1.2 Previous work

The combination of different verification techniques, such as formal methods, scenario-based simulations and real testing has shown promising results. Fremont et al. [9] combined formal verification with scenario-based simulation as an approach for evaluating the safety of an AV in both simulation and on a real-world test track. The aim was to bridge the gap between simulation and real-world testing. The study involved the open-source tools, VerifAI, SCENIC, and a vehicle simulator. SCENIC is a probabilistic language for specifying scenarios, to capture key features of the AV and its environment in an abstract parameter space. VerifAI is a toolkit that can sample scenes from the SCENIC description, run the simulation, and falsify safety specifications. The specifications that VerifAI falsifies can be formulated in the formal language, metric temporal logic (MTL) [10]. The paper in question, concluded that their approach produced qualitatively similar trajectories in simulation and real-world testing for the same scenario. The setup was demonstrated to be effective at finding counterexamples that transfer well to real-world testing. The simulated test cases with safe behavior resulted in 93.3% of the same behavior on the track [9]. With the background of these promising results, this master thesis will utilize a similar setup. The setup used in [9] is shown in Fig. 1.2.

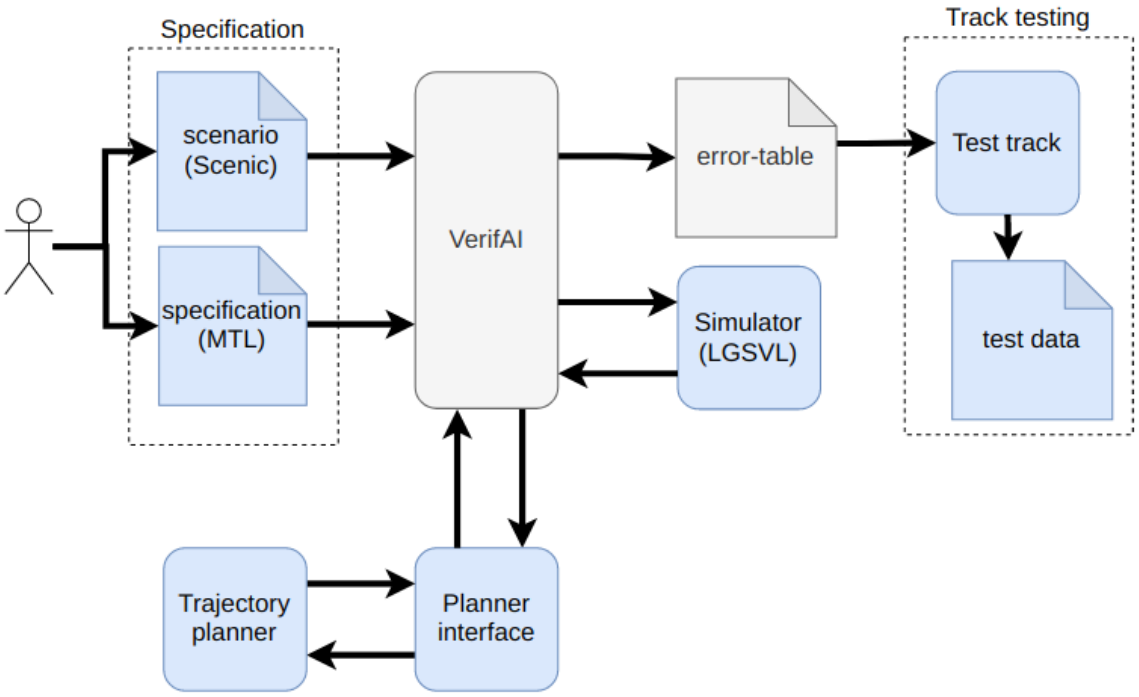

Figure 1.2: Structure of the toolchain and process used by Fremont et al. [9], described in chapter 1.2. The blue sections are the parts of the toolchain that were not kept for use in this thesis.

1.3 Aim and scope

The aim of this master thesis is twofold. First, to find out how formal safety requirements can be formally specified. This aim both involves to translate natural language to formal languages but also to express complex safety requirements in formal language. Secondly, to set up a toolchain and use formal specifications to falsify requirements on a trajectory planner.

Instead of evaluating a complete AV-system a trajectory planner together with a mode management module is evaluated. The implemented toolchain utilizes an existing framework that consists of SCENIC, VerifAI and one of several simulators, in this case Carla. The structure of the thesis toolchain is shown in Fig. 1.3. As you can see, the Figures 1.2 and 1.3 are similar, as the thesis toolchain is based on the toolchain from [9]. Fig. 1.3 shows in green the parts that are different from [9] and in dark green what had to be developed for this thesis work. The dark green square representing the MTL-specification captures a significant part of the aim of this thesis. In Fig. 1.2 the track testing represents the process of testing falsified scenes in a physical simulation. That is beyond the scope of this thesis, and as such is not included in Fig. 1.3. The VerifAI framework is utilized as is and not developed in any significant way in the thesis work, focus is instead on its inputs. Also, only the longitudinal functionality of the trajectory planner is tested.

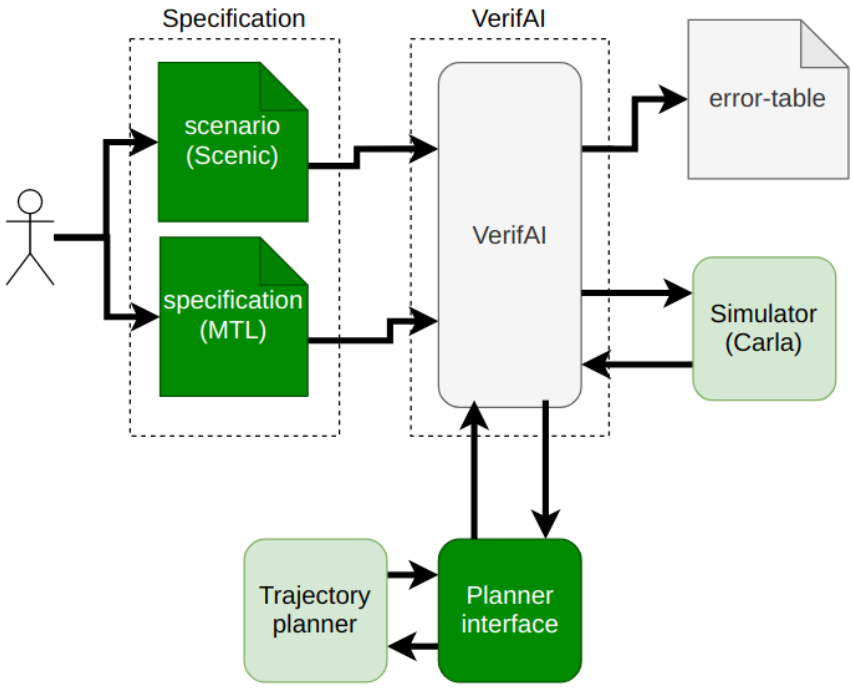

Figure 1.3: Overview of the used software, dark green represents contributions made by the thesis.

1.4 Research questions

With the mentioned aim and scope, the research questions of the master thesis are as follows:

- What are the difficulties when formulating natural language as metric temporal logic, or formal languages in general?
- How does the decisions in formulating natural language as metric temporal logic affect the outcome of formal simulations?
- What are advantages and difficulties when using falsification of a black-box system as a verification method in autonomous driving?
- How can a meaningful robustness metric be formalized in the context of falsification through formal simulation?

1.5 Disposition of the report

The next chapter includes theory about the verification techniques that are used in this thesis, as well as an introduction to MTL and a literature review about safety specifications in temporal logic. In chapter 3, the structure of the toolchain, a description of the simulation process as well as information about the evaluation of the results are presented. Chapter 4, provides definitions of the implemented safety parameters as well as the implemented MTL specifications and scenarios. The simulation results are shown in chapter 5 followed by chapter 6 where the results are discussed and other critical parts for this master thesis. The thesis ends with

chapter chapter 7, where a summary and conclusions are shown, and the research questions revisited.

2
Theory

In this chapter, a presentation of the verification techniques that are used in this project are presented. Followed by an introduction to the formal language, MTL. The last section shows primarily related work where safety specifications have been formalized into temporal logic.

Ensuring absolute safety in a vehicle is impossible, as long as it is subject to other actors who can not be controlled. An example of such a situation can be seen in Fig. 2.1. From the central vehicle's perspective, if one of the surrounding actors performs an unsafe action, the central vehicle can do nothing to avoid it [4]. Instead requirements often specify functionality that is safe enough, i.e. follows a standard. To ensure safe functionality a verification method is needed.

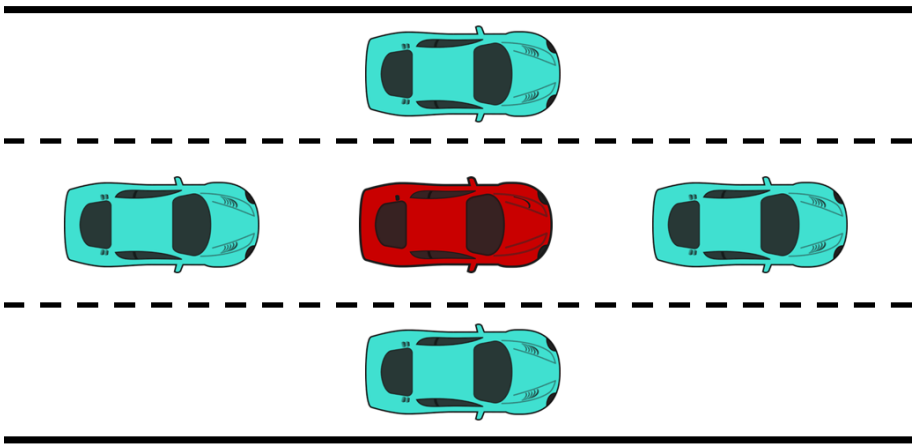

Figure 2.1: From the central vehicle's perspective, there is impossible to ensure absolute safety in this position

2.1 Verification

There exist many different techniques for safety assurance of systems. They can largely be divided into testing and formal methods. While testing can only prove the existence of errors in the system, formal methods can provide complete correctness proof. Fully formal methods such as model checking can search through all state space and either verify the correctness of the system or falsify the system specification. While completeness is attractive, model checking is not feasible for every system. Either the system is too complex, leading to an extreme computational cost to search the state space, or the system is not defined, such as a neural network. In these cases, so called semi-formal techniques can be pursued, such as formal scenario-based simulation [1], [11].

To expose a vehicle to traffic, one approach is to put it in an open environment, such as a real city or a complex simulated environment, to generally evaluate the vehicle's performance. Another approach is to identify a specific safety aspect to be tested and tailor a scenario to expose that aspect to the fullest. When running such a scenario in a simulation, it is commonly called scenario-based simulation. One disadvantage of utilizing scenarios is that the situation is often very predictable and only exposes a small part of the feature space. To mitigate that somewhat, the features of the scenario can be sampled to create perturbations in the initial values of the simulation and provide a more robust exploration of the safety specifications [1], [12].

2.2 Metric temporal logic

Formal languages is an area in mathematics that can be used as a means to express logical statements. In contrast with natural languages, all formal languages are well-formed i.e. they conform to a specific set of rules and relies on math. The formal language MTL is a modal temporal logic language that can describe behavior for well defined intervals of time. It is thus possible to describe bounded intervals for logic specification, further enhancing the use of logic in practical applications [13]. An example of how to use MTL when specifying the behavior of a real-time system now follows as a starting point. If it is desired to stipulate deadlines between environment events and corresponding system responses, for example every *alarm* is followed by a *shutdown* event in 10 time-units unless *allclear* is happening before 10 time-units. This can be formulated as an MTL sentence [14]:

$$\Box(\text{alarm} \rightarrow (\Diamond_{(0,10)}\text{allclear} \vee \Diamond_{[10]}\text{shutdown})) \tag{2.1}$$

As can be seen in this example, the formulation of MTL is built from a set of atomic propositions $\mathcal{P}$ (exemplified by *alarm*, *allclear* and *shutdown*), combined with boolean operators (exemplified by $\vee$) and temporal operators (exemplified by $\Box$ and $\Diamond$). The following sub-chapters explains the syntax and semantics that defines how MTL sentences are formed in this thesis work.

2.2.1 Syntax

The full syntax of MTL is defined as:

$$\varphi, \psi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \psi \mid \varphi \, \mathcal{U}_{\mathbb{I}} \, \psi \tag{2.2}$$

where φ and ψ notates a sentence described by any of the following: $\top$, defined as *True*; $p \in \mathcal{P}$, a atomic proposition; $\neg\psi$, a *negated* sentence; $\varphi \vee \psi$, the *or* operator; the temporal operator *until* $\varphi \, \mathcal{U}_I \, \phi$, defined as true, if φ holds until a time $t \in \mathbb{I}$ after which ψ holds. [14]–[16]. Additionally, other operators can be expressed using the operators above, such as: the *and* operator $\varphi \wedge \psi \equiv \neg(\neg\varphi \vee \neg\psi)$; the *implies* operator $\varphi \rightarrow \psi \equiv \neg\varphi \vee \psi$; *false* is defined as $\bot \equiv \neg\top$. The temporal operators used in this thesis are the *eventually* operator, $\Diamond_I \varphi \equiv \top \, \mathcal{U}_I \, \varphi$ and *always* operator $\Box_I \varphi \equiv \neg\Diamond_I \neg\varphi$ [17]. The temporal operators can be unbounded or bounded. Bounded means that the operator has a time interval $\mathbb{I}$, of real numbers, $\mathbb{R}$. The different temporal operators are exemplified in the Fig. 2.2, with the atomic propositions a and b.

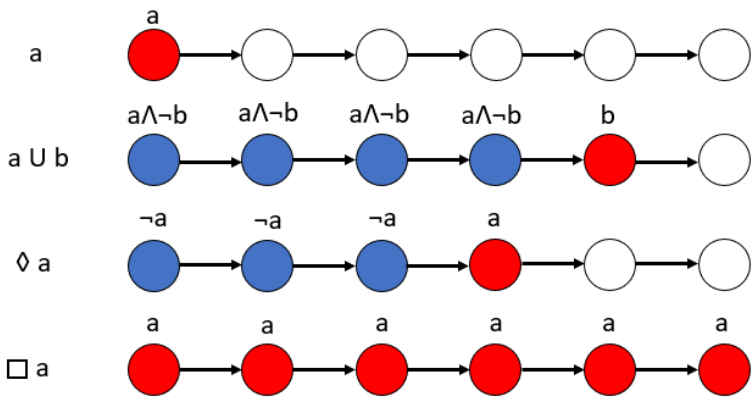

Figure 2.2: Some temporal operators exemplified.

2.2.2 Semantics

The semantics of MTL describes what it means for an MTL sentence to hold at a time t, and in this thesis the semantics are as follows. A timed word, or trace, γ is defined as a finite sequence (r_0, t_0), (r_1, t_1)... (r_m, t_m) where $r_k \in 2^{\mathcal{P}}$ and $t_k \in \mathbb{R}$. The satisfaction of an MTL sentence φ for a trace γ at time t_k is notated as:

$$\rho_\varphi(\gamma, t_k) \equiv \Big(\gamma, t_k \models \varphi\Big) \tag{2.3}$$

Furthermore, MTL supports different kinds of semantics. One, qualitative semantics, supports Boolean expressions and the satisfaction ρ is evaluated as a Boolean value. The second, quantitative semantics returns instead a robustness value and is defined for real-valued traces. The robustness value $\rho_\varphi(\gamma, t_k)$ evaluates to a real value that reveals two things about the trace. The sign of the robustness value shows whether the sentence is satisfied ($\rho_\varphi(\gamma, t_k) > 0$) or violated ($\rho_\varphi(\gamma, t_k) < 0$). Also,

the magnitude $|\rho_\varphi(\gamma, t_k)|$ sets the bound on the perturbation that an execution of the trace γ can resist without changing its true value. So, if $\rho_\varphi(\gamma_1, t_k) > \rho_\varphi(\gamma_2, t_k) > 0$, this means that both executions are satisfied, but γ_1 are more robustly satisfied than γ_2 [14], [18].

2.3 Literature review of safety specification

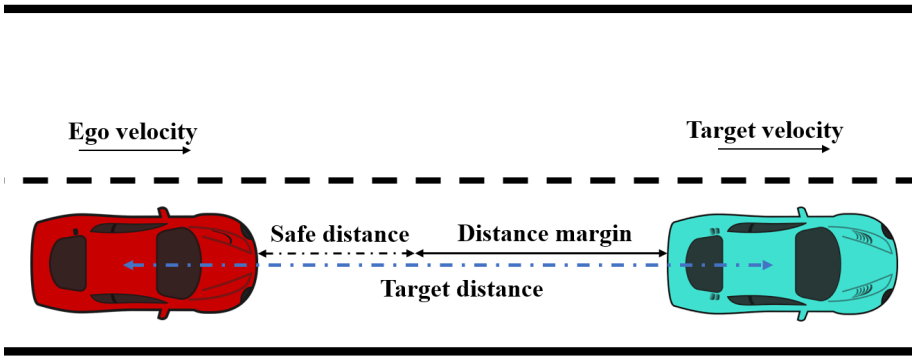

Figure 2.3: Notations for different longitude distance and with the red vehicle as the ego vehicle

There are multiple examples in the existing literature of safety requirements formulated as MTL sentences. This section reviews two definitions of longitudinal safe distance between the ego vehicle and a target, such as another vehicle or a pedestrian, to avoid collisions. Both use the same MTL formulation: $\varphi = \Box d_{\text{margin}}$. What differs is the definition of the safe distance [19]. The complexity of these definitions are various, in Fremont et al.'s [9] article, the MTL specification is quite simple as:

$$d_{\text{margin}} = d_{\text{target}} - 2.5 \tag{2.4}$$

where d_{target} is the distance from the ego vehicle's centre to the target actor's centre and 2.5 meter is the safe distance [9]. For an general illustration of the distance notations, see Fig. 2.3. The safe distance can also be defined with the dynamical properties of the actors as:

$$d_{\text{margin}} = d_{\text{target}} - d_{\text{safe}}^{\text{long}} \tag{2.5}$$

$$d_{\text{safe}}^{\text{long}} = max(v_{\text{ego}} t_r + \frac{1}{2} a_{\max} t_r^2 + \frac{(v_{\text{ego}} + a_{\max} t_r)^2}{2a_{\text{min,brake}}} - \frac{v_{\text{lead}}^2}{2a_{\text{max,brake}}}, 0) \tag{2.6}$$

where v_{ego} is the velocity of ego vehicle, v_{lead} is the velocity of the lead vehicle, t_r is the response time of ego vehicle, $a_{\max}$ is the maximum acceleration of ego vehicle, $a_{\text{min,brake}}$ is the minimum applied braking acceleration and $a_{\text{max,brake}}$ is the maximum braking acceleration of the lead vehicle [4], [18]–[21]. In Table 2.1, values on the parameters in (2.6) have been gathered from different literature sources [18]–[20].

Reference	t_r	$a_{\text{max,acc}}$	$a_{\text{min,brake}}$	$a_{\text{max,brake}}$
[18]	0.5	4.1	4.6	8
[19]	0.1	3	2.5	3
[20]	0.5	5.5	4	10
	s	m/s^2		

Table 2.1: Parameter data to the safe distance in equation 2.6, from different references

The formulation of minimum safe distance can be useful when formalizing MTL specifications. In [22], many different traffic rules are translated into MTL specifications. One example of a general traffic rule is that the vehicle should have *safe distance to preceding vehicle*, which means three things. First, the ego vehicle must maintain a safe distance to ensure collision freedom for all vehicles, even if a sudden stop occurs, with the leading vehicles within the same lane. Second, the ego vehicle must recover the safe distance within a predefined time after another vehicle, else the safe distance violation with each other. Third, this rule must be evaluated with respect to every other vehicles within the sensor range of the ego vehicle. Another general traffic rule is that the ego vehicle should not do *unnecessary braking*, namely brake abruptly without any reasons. These three cases are accepted as necessary braking situations: a) there is an obstacle in front of the ego vehicle and the accelerations difference does not violate a threshold; b) the safe distance to a leading vehicle within the same lane is violated; c) there exist a leading vehicle and the ego vehicle's acceleration does not violate an acceleration threshold. Furthermore, a general guideline regarding formalizing traffic rules into temporal logic are given by Rizaldi et al. [23] and Maierhofer et al. [22] as:

1. Concrete traffic rules
2. Identify relevant atomic propositions, predicates and functions
3. Concretely define each atomic propositions, predicates and functions
4. Create temporal logic formula

The identified propositions in step 2, must be combined into a temporal logic formula that matches the concertised rules from step 1 [22]. This means that it should be possible to monitor the satisfaction of a trace against a temporal logic formula [23].

3

Implementation, simulation and evaluation of toolchain

The approach of this thesis was to utilize a toolchain that previously had shown promising results from verification of AVs [9] and is implemented in this thesis to falsify a trajectory planner that is created by the company Zenseact. This chapter introduces the components of this toolchain and other dependencies, the aspects that factor into the result, how the simulation process was carried out and lastly how the results was evaluated. The chapter starts with an overview of the setup and its components.

3.1 Components

This section provides a brief overview, based on Fig. 3.1, of the components of the toolchain. The components of the toolchain are centered around VerifAI, see section 3.1.4. VerifAI is used to falsify the trajectory planner from Zenseact in accordance to a specification. VerifAI samples the scenario provided in the specification and evaluates the trajectory planner in the simulator Carla, see section 3.1.1. The evaluation is done according to an MTL-specification and produces an error table with an entry for each sample of the scenario. The sampling of the scenario can be done either according to a defined distribution or using a search algorithm, see section 3.1.4.1

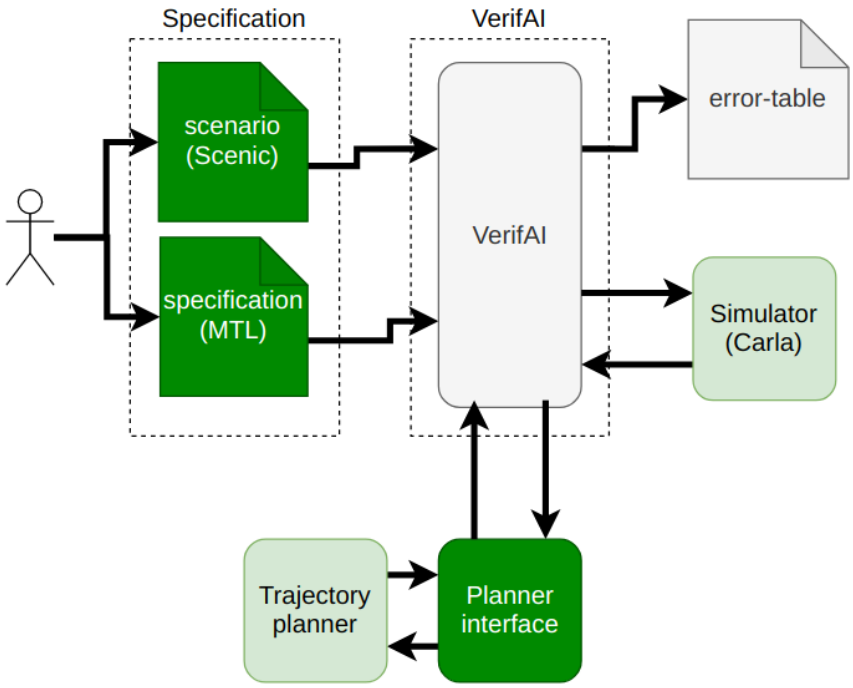

Figure 3.1: The toolchain's structure, where green is differences from Fig. 1.2, dark green being contributions from the thesis work.

3.1.1 Carla

Carla is an open-source autonomous driving simulator based on Unreal Engine for the physics simulation. It was created with the purpose of democratizing autonomous driving research and development, and was designed to be applied in a range of problems related to autonomous driving. Carla provides a Python API where most of the relevant simulation states can be accessed and modified [24]. Therefore, Carla was selected as the simulator in this thesis.

3.1.2 Planner interface

The Planner interface serves the toolchain by allowing the Zenseact trajectory planner to communicate with VerifAI and the simulation environment. The planner interface was developed during the thesis and is a solution for the Zenseact trajectory planner in particular. The trajectory planner needs three main input signals: a model of the lanes, information about the tracked targets as well as the state and parameters of the ego vehicle. The input signals are created by the interface by extracting the needed information from the simulation each timestep. The trajectory planner then returns an output in the form of an acceleration request that is fed in to the controller described in section 3.1.2.1.

3.1.2.1 Controller

To handle the requests received from the planner, a simple controller was implemented based on an inversion of the physics engine's drive-train model. Since, the

planner issues multiple acceleration requests, a simple arbitration algorithm was implemented as well. The model was based on the PhysX vehicle model used in the Unreal Engine [25], and was used to solve for the throttle t from an acceleration request. The engine dynamics was modeled as follows:

$$I_{\text{engine}} \cdot \dot{\omega}_{\text{engine}} = \tau_{\text{throttle}} - \tau_{\text{engine}} \tag{3.1}$$

where I_{engine} is the moment of inertia of the engine, $\dot{\omega}_{\text{engine}}$ is the angular acceleration of the engine, τ_{engine} is the torque applied from the axle connected to the gears and τ_{throttle} is the torque applied from the throttle. The torque applied by the throttle is in turn described by the equation:

$$\tau_{\text{throttle}} = \tau_{\max}(\omega_{\text{engine}}) \cdot t \tag{3.2}$$

where $\tau_{\max}(\omega)$ is an interpolating function provided by the simulator that returns the maximum torque for the given angular velocity of the engine, ω_{engine}. The engine dynamics was connected to the wheel dynamics through three algebraic equations representing the gearing of the vehicle, where n_{wheels} is the number of wheels of the vehicle. Worth noting is that the vehicle used in the simulation was electric and only had one gear.

$$\tau_{\text{engine}} = \frac{\tau_{\text{wheel}}}{i_{\text{gear}}} \cdot n_{\text{wheels}} \tag{3.3}$$

$$\omega_{\text{engine}} = \omega_{\text{wheel}} \cdot i_{\text{gear}} \tag{3.4}$$

$$\dot{\omega}_{\text{engine}} = \dot{\omega}_{\text{wheel}} \cdot i_{\text{gear}} \tag{3.5}$$

where τ denotes torques, ω denotes angular velocity and $\dot{\omega}$ denotes angular acceleration, i_{gear} is the gearing of the vehicle. The wheel dynamics, where r is the radius of the wheel are described by

$$I_{\text{wheel}} \cdot \dot{\omega}_{\text{wheel}} = \tau_{\text{wheel}} - r \times F_{\text{ground}} \tag{3.6}$$

where I_{wheel} is the moment of inertia of the wheel, $\dot{\omega}_{\text{wheel}}$ is the angular acceleration of the wheel, τ_{wheel} is the torque propagating from the gearing and F_{ground} is the friction force between the wheel and the ground. The friction force connects the wheel dynamics to the linear dynamics of the vehicle body. The brakes on the wheels interact with the dynamics as follows:

$$\tau_{\text{wheel}} = \tau_{\text{maxbrake}} \cdot b \tag{3.7}$$

where b is the brake engagement and τ_{maxbrake} is the maximum braking torque. The acceleration request from the planner is input as the acceleration in

$$n_{\text{wheels}} \cdot F_{\text{ground}} = m_{\text{vehicle}} \cdot a \tag{3.8}$$

where n_{wheels} is the number of wheels, m_{vehicle} is the mass of the vehicle and a is the requested acceleration. With the following equations:

$$\omega_{\text{wheel}} = \frac{v}{r} \tag{3.9}$$

$$\dot{\omega}_{\text{wheel}} = \frac{a}{r} \tag{3.10}$$

and v being the velocity of the vehicle. Eq. (3.1)-(3.10) can be solved for throttle t and brake b to be applied to induce the requested acceleration.

When used in the controller, an acceleration request error is propagated through the model to produce a throttle error, and a brake error. The errors are applied to the current throttle and brake, and the parameter that was not equal to or less than zero last time step is applied to the vehicle, while the one that was is set to zero.

3.1.3 SCENIC

SCENIC is a Python-based language for specifying probabilistic scenarios, with a syntax meant to simplify the description of complex traffic scenarios [26]. The purpose SCENIC filled in the toolchain was to parse the description of the scenario, instantiate a given sample of the scenario in simulation, as well as in each step apply the defined behaviors. Modifications to the SCENIC source was limited to the definition of a `Behavior` method for calling the planner interface. The code of the scenarios can be found in subchapter 4.2.

3.1.4 VerifAI

VerifAI is a toolbox that can monitor formal specifications during a series of samples of a SCENIC scenario and use the monitored metric to inform the sampling, with the objective of falsifying the simulated system [27]. The sampling was done from a scenario defined by the SCENIC program and the sampling was informed using one of two cost functions, cross-entropy loss or Bayesian optimization. Modifications were made in the toolkit to enable evaluation of a custom simulation trace, see section 2.2.

3.1.4.1 Falsification

To falsify a specification using VerifAI an optimization method was used. Bayesian optimization is an optimization algorithm implemented in VerifAI with the python package `GPyOpt`. The method is focused on solving problems of the form

$$\max_{x \in A} f(x) \tag{3.11}$$

and is most successful in applications with $x \in \mathbb{R}^d$ for $d \leq 20$. Another assumption in Bayesian optimization is that $f(x)$ lacks known special structures that could improve efficiency, i.e. $f(x)$ is a "black-box" [28]. Bayesian optimization was chosen as the optimization method on this background, since both the mentioned criteria fits our problem. The function $f(x)$ is defined in VerifAI as $\rho_\varphi(\gamma = \mathcal{M}(\pi))$ for a specification φ and, x, the variable to be optimized is the feature space sample π.

3.2 Method

The following section provides a description of the process of using the toolchain; the method used to provide the results in chapter 5. In Algorithm 1, the process of

a simulation run in the toolchain is presented.

First, the toolchain was set up in the framework of VerifAI, SCENIC and Carla and the trajectory planner was integrated. Second, MTL-specification and a scenario described in SCENIC, was selected for each run of simulations. The purpose of investigating whether the trajectory planner was able to be falsified in the scenario using the specification in question. The choice of inputs was based on literature and the company. More about the specific tested formal requirements and scenario descriptions are present in chapter 4.

The next part of the process of using the toolchain was to run simulations, by first starting the vehicle simulator, then initialize the monitor with the chosen sampler. The sampling sequence length as well as the number of time steps in each simulation were chosen. After each simulation in the simulation run, the framework evaluated the trace of the simulation and produced a value, $\rho = \varphi(\gamma)$, evaluated on the sample trace, γ, according to the specified formal specifications. The value was then used in the sampling of the next simulation in the run. This means that simulation, evaluation and sampling were performed for the specified number of simulations. The feature space sample of each counterexample was extracted to examine the falsified behaviour.

Algorithm 1 was specified based on the process described in [10]. The algorithm explains how the toolchain searches for a sample that falsifies the specification. When sampling the feature space $\mathbf{\Pi}$ n times, the simulation is expected to respond differently to different samplings. The simulation model $\mathcal{M}$, maps the sample to a trace γ over m time steps. The trace is evaluated at all time steps using the MTL-specification φ resulting in a metric ρ. The distribution of $\mathbf{\Pi}$ is updated every sample using a cost function $\mathbf{O}$ that minimizes the metric ρ. Running the simulation for a sufficient number of samples results in corner cases with a feature space sample

that minimizes the cost function. [1].

Algorithm 1: Falsify scenario

Result: $\vec{\pi} = (\pi_0, \pi_1, ..., \pi_n)$ and $\vec{\rho} = (\rho_0, \rho_1, ..., \rho_n)$
φ : an MTL specification
p : the relevant states in the simulation at time t_k
Π : the feature-space defined in SCENIC
P : the feature-space initial distributions
O : an optimization algorithm
$\mathcal{M}$: the environment model consisting of Carla and the agents
n : the number of simulations in the sequence
m : the number of time steps in each simulation
while $i < n$ **do**
  sample the feature space $\pi_i = \Pi \sim P$
  while $k < m$ **do**
    $\gamma_k = \mathcal{M}(\pi_i, t_k)$
  end
  evaluate trace $\rho_i = \varphi(\gamma)$
  update distribution $P := O(P, \rho_i)$
end

In the next section the method for evaluating the Algorithm 1 results is specified.

3.3 Evaluation

In each simulation session, the method described in Algorithm 1 was performed for a chosen MTL requirement φ in the scenario. The result was series of samples π_i from the feature space Π, with a resulting robustness metric ρ_i. Negative values on ρ_i indicated that the corresponding sample π_i of the feature space was a counter example. The counter examples were simulated again in Carla and inspected to determine whether the method actually had discovered a situation where the planner did not fulfill the specification. Also, comparisons were done between simulation sessions with similar MTL-requirements.

4

Temporal logic specifications and scenario descriptions

In order to evaluate a trajectory planner an appropriate environment is needed, described by a scenario. In this chapter scenarios are presented that represents possible scenes in traffic. Also, the safety requirements to be tested that are formulated as MTL specifications are shown. To support evaluating the statements, the appropriate atomic propositions must be defined as part of the simulation trace, see section 2.2 and Algorithm 1. First in this chapter is an overview of all the MTL specifications that were tested in this thesis work.

4.1 Metric temporal logic specifications

All tested MTL safety specifications are presented in this section and the definitions of the needed atmic states will be explained below. These are the MTL specifications that were used in the simulation in the different scenarios.

$$\begin{aligned} \varphi_1 &= \square(d^{\mathrm{A}}_{\mathrm{margin,target}}) \\ \varphi_2 &= \square(d^{\mathrm{B}}_{\mathrm{margin,target}}) \\ \varphi_3 &= \square(j_{\mathrm{margin,ego}} \vee \neg\Diamond_{[-0.5,0]} d^{\mathrm{A}}_{\mathrm{margin,target}}) \end{aligned} \tag{4.1}$$

The first two specifications, φ_1 and φ_2 are used in the same scenario. The third assertion, φ_3 was simulated in another scenario and is a formalization of avoiding hard braking for the ego vehicle unless it is motivated by traffic. Note that the evaluation of the MTL statements are assumed to be quantitative, see section 2.3.

4.1.1 MTL requirement: Always keep a safe distance

As mention in section 2.3, a requirement for an AV can be expressed in natural language, such as *always keep safe distance to a vehicle in front of ego vehicle within the same lane*. This requirements translates to MTL as:

$$\varphi = \square(d_{\mathrm{margin,target}}) \tag{4.2}$$

where the always operator expresses that $d_{\mathrm{margin,target}}$ should always be positive. Two definitions of $d_{\mathrm{margin,target}}$ are used in this thesis, both specified in section 2.3.

4.1.2 atomic proposition: Distance margin

There are two definitions of the distance margin used in the tests, or more specifically, two definitions of d_{safe}. The distance margin is defined as

$$d_{\text{margin,target}} = d_{\text{target}} - d_{\text{safe}} \tag{4.3}$$

which is the difference between the relative distance to the target and the safe distance, see Fig. 4.1. While this state is simple so far, there is some complexity in the definition of d_{safe} as well as d_{target}. Algorithm 2 was created for finding the shortest distance between ego and target, given the coordinates of the vertices of imagined rectangles around each vehicle. All distances between two vertices, given that they belong to different vehicles, are collected in the list d. Indices of the sorted lengths are collected, these are then decoded to find the vertices of each vehicle sorted by distance to vertices of the other vehicle. The two shortest vertex pair‘s indices are then picked for each vehicle and placed in a set, eliminating duplicates. From these indices and the input vertices X, p is created containing the vertex or vertices that is part of the two closest pairs from d belonging to the each vehicle. If both p_0 and p_1 is the same length, the two vehicles closest sides are parallel.

Algorithm 2: Shortest distance between rectangles

Result: d_{target}
X_0 : the vertices of the ego vehicle
X_1 : the vertices of the target vehicle
for x_0 *in* X_0 **do**
  for x_1 *in* X_1 **do**
    append $||x_1 - x_0||$ to d
  end
end
indexes = arg_sorted(d)
indexes_0 = set(mod(indexes[0,1], 4))
indexes_1 = set(floor_divide(indexes[0,1], 4))
$p_0 = X_0[\text{indexes}_0]$
$p_1 = X_1[\text{indexes}_1]$
if *len*(p_0) == *len*(p_1) **then**
  $L = p_1[1] - p_1[0]$
  $P = X_0[\text{indexes}_0[2]] - p_1[0]$
end
if *len*(p_1) == 2 **then**
  $L = p_1[1] - p_1[0]$ $P = p_0[0] - p_1[0]$
end
if *len*(p_0) == 2 **then**
  $L = p_1[1] - p_1[0]$ $P = p_0[0] - p_1[0]$
end
$t = P \cdot ||L||$
$d_{\text{target}} = ||L \cdot t - P||$

Furthermore, the distance d_{safe} is a measure of how far away the target needs to be in order for the ego vehicle to react safely. Two definitions are tested in the thesis

work. The first involves the ego vehicle's velocity and the minimum time to reach the target vehicle, such that

$$d^{\mathrm{A}}_{\mathrm{safe}} = v_{\mathrm{ego}} t \tag{4.4}$$

The second expression:

$$d^{\mathrm{B}}_{\mathrm{safe}} = v_{\mathrm{ego}} t_r + \frac{1}{2} a_{\mathrm{max,ego}} t_r^2 + \frac{(v_{\mathrm{ego}} + a_{\mathrm{max,ego}} t_r)^2}{2a_{\mathrm{brake,min}}} - \frac{v^2_{\mathrm{target}}}{2a_{\mathrm{brake,max}}} \tag{4.5}$$

and the equation is introduced in section 2.3 together with a description of the thoughts behind the safe distance.

4.1.3 MTL requirement: Avoid unnecessary hard braking

The second requirement can be expressed in natural language as *the ego vehicle should avoid hard braking unless it is motivated by traffic.* To translate this requirement it was broken down into two parts: *avoid hard braking* and *motivated by traffic.* To *avoid hard braking* was defined as a minimum jerk margin

$$j_{\mathrm{margin}} \tag{4.6}$$

being fulfilled. The part *motivated by traffic* is not well-defined and requires some interpretation. It was interpreted as the vehicle being exposed to risk for a longer time interval than the time the vehicle needs to mitigate the risk. In MTL that can be stated as always being exposed to risk throughout the mentioned time interval. This can be alternatively expressed as to not eventually (i.e. not at least once during the interval), be not exposed to risk

$$\neg \Diamond_I (\neg(\text{exposed to risk})) \equiv \Box_I (\text{exposed to risk}) \tag{4.7}$$

The expression *not exposed to risk* was defined as the atomic proposition d_{margin} being satisfied

$$\neg \Diamond_I (d_{\mathrm{margin}}) \tag{4.8}$$

where the interval I is equal to $[-t, 0]$ with t being the time it should take the vehicle to become safe. The evaluation of the MTL sentence takes place at time 0, where hard braking should be avoided if the vehicle was not exposed to risk during the previous time interval. That is the reason why the time interval is negative. This time t should be considered a function of d_{margin} but is set to a constant since the Python implementation of MTL that is used in the thesis does not support variables as intervals. Together, equations (4.6) and (4.8) becomes

$$\varphi = \Box(j_{\mathrm{margin,ego}} \vee \neg \Diamond_{[-0.5,0]} d^{\mathrm{A}}_{\mathrm{margin,target}}) \tag{4.9}$$

$$\varphi = \mathrm{A}(\mathrm{jerk}_{\mathrm{margin,ego}} \vee \neg \mathrm{E}_I \mathrm{distance}^{\mathrm{A}}_{\mathrm{margin,target}}) \tag{4.10}$$

where $d_{\mathrm{margin,target}}$ was chosen to be defined by $d^{\mathrm{A}}_{\mathrm{margin,target}}$.

4.1.4 atomic proposition: Jerk margin

The atomic proposition jerk margin j_{margin} was defined as the difference between the current vehicle jerk j_{ego} and the most negative jerk allowed j_{min}.

$$j_{\text{margin}} = j_{\text{ego}} - j_{\text{min}} \tag{4.11}$$

where j_{min} is a value interpolated, from a table, with respect to the speed of the ego vehicle.

4.2 Scenarios

As mention earlier, the scenarios was specified in the software program, SCENIC to be simulated the Carla environment. Specifying a scenario with multiple probabilistic parameters makes it possible to explore an arbitrary amount variations of the intended scenario. In the following sections the different scenarios will be presented together with the safety specification in MTL for respectively test and how the scenarios were written in SCENIC.

4.2.1 Safe distance to a leading vehicle

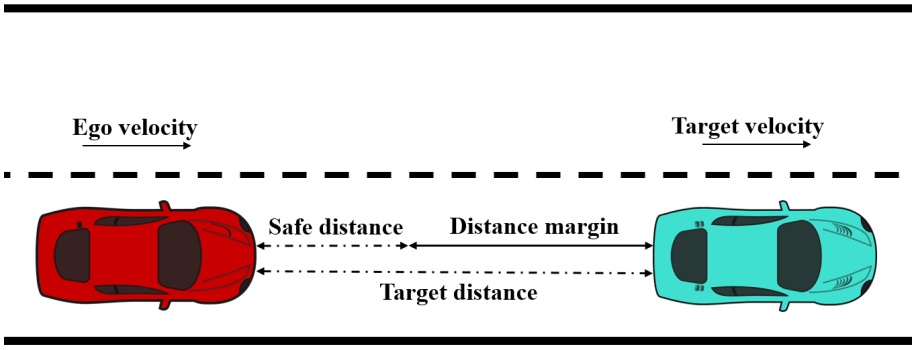

Figure 4.1: The red ego vehicle distance to a target vehicle and the notations of the distance expressions

This scenario simulates two vehicles within the same line with an initial speed and initial positions. After a random time, the leading vehicle brakes with a random force. An pseudo-code in SCENIC language for the scenario is described below as:

Algorithm 3: Braking lead vehicle

```
model scenic.simulator.carla.model

/* BEHAVIORS                                                    */
behavior BrakeBehavior(velocity_target, delay, brake):
    do FollowLaneBehavior(velocity_target) for delay seconds
    take SetBrakeAction(brake)

/* EGO VEHICLE'S BEHAVIOR                                       */
ego = Car following roadDirection from startPoint for 5,
    with behavior PlannerBehavior(globalParameters.set_speed),
    with blueprint 'vehicle.tesla.model3',
    with color (1, 0, 0)

/* TARGET VEHICLE'S BEHAVIOR                                    */
car1 = Car following roadDirection from startPoint for 15,
    with behavior BrakeBehavior(globalParameters.set_speed_target,
    globalParameters.delay, globalParameters.brake_intensity),
    with blueprint 'vehicle.tesla.model3',
    with color (0, 0, 1)
```

The evaluation of this scenario were with quantitative semantics of the MTL specifications because it benefit of telling the safety robustness of the trajectory for the simulated traces. The safety specification is presented in the equation (4.2) with the belonging equations, (4.3)- (4.5).

4.2.2 Cut in back of the vehicle

Algorithm 4: Target vehicle cutting in

```
model scenic.simulator.carla.model

/* BEHAVIORS                                                    */
behavior DelayedLaneChangeBehavior():
    do FollowLaneBehavior(globalParameters.target_speed) for 3 seconds
    try:
      do FollowLaneBehavior(globalParameters.target_speed)
    interrupt when self.distanceTo(ego) < globalParameters.distance:
      do LaneChangeBehavior(startSection)
    terminate

/* EGO VEHICLE'S BEHAVIOR                                       */
ego = Car following roadDirection from startPoint for 10,
    with behavior PlannerBehavior(globalParameters.target_speed + 5),
    with blueprint 'vehicle.tesla.model3',
    with color (1, 0, 0)

/* TARGET VEHICLE'S BEHAVIOR                                    */
car1 = Car following roadDirection from leftPoint for 10 +
 globalParameters.position,
    with behavior DelayedLaneChangeBehavior(),
    with blueprint 'vehicle.tesla.model3',
    with color (0, 0, 1)
```

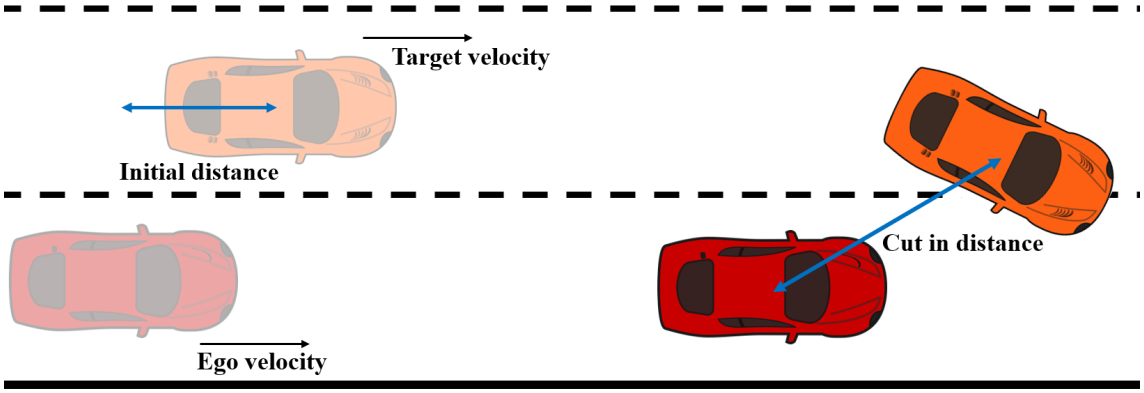

Figure 4.2: A orange target vehicle is cutting in front of the red ego vehicle and the parameter for feature space is illustrate

The second tested scenario scene was together with a ego vehicle and a target vehicle. The design of this scenario was that the ego vehicle drives in the right road with a higher velocity than target. In the left road, the target vehicle was initialized

further down on the roads than the ego. The target vehicle made a right turn when the ego vehicle is in a area besides the target vehicle. The MTL specification in Equation (4.10) was tested in this scenario with the belonging equations (4.3), (4.4) and (4.11). The equations evaluated if the ego vehicle made an unnecessary hard breaking and the specification is violated if the ego vehicle made an hard break and it was not motivated. Below is the psuedo-code in SCENIC for the scenario.

5

Simulation results

The following chapter contains results from running the simulation setup described in chapter 3, with the inputs presented in chapter 4. Table 5.1 specifies the settings and inputs of the simulation runs that were performed. Every test was sampled 50 times each run, and each sample was simulated for 800 time steps of 0.0125 s.

Table 5.1: Test configurations

Test	Scenario	Parameters	MTL
1	Braking target vehicle	initial distance set speed ego set speed lead	$\varphi_1 = \Box(d^{\mathrm{A}}_{\mathrm{margin,target}})$
2	Braking target vehicle	initial distance set speed ego set speed lead	$\varphi_2 = \Box(d^{\mathrm{B}}_{\mathrm{margin,target}})$
3	Target vehicle cutting in	distance when cut in initial distance set speed target	$\varphi_3 = \Box(j_{\mathrm{margin,ego}} \vee \neg\Diamond_{[-0.5,0]} d^{\mathrm{A}}_{\mathrm{margin,target}})$

The first test in the results was performed on the *Braking target vehicle* scenario and with the MTL formulation $\varphi_1 = \Box(d^{\mathrm{A}}_{\mathrm{margin,target}})$. The Bayesian optimization of the sampling process gave rise to 16 counterexamples. The counterexamples‘ location in the feature space can be seen in Fig. 5.1. Each test was a sample from the feature space defined by the relevant scenario, see chapter 4, represented by a point in the figure. The color of each point corresponds to the robustness metric that each sample evaluated to after a simulation. The point at the bottom of Fig. 5.1 visualizes a sample of the Braking target vehicle scenario where the set speed of the ego vehicle is high, the set speed of the target vehicle is low and the initial distance between the vehicles is low. This results in a counter example with a robustness of less than negative one. Changing the view of the feature space to only show the *set speed target* and *set speed ego*, see Fig. 5.2 the relationship between *rho* and *set speed target* is clearer. The test's counterexamples seems to be clustered at medium high speeds or if there is a high difference in the speed between the vehicles.

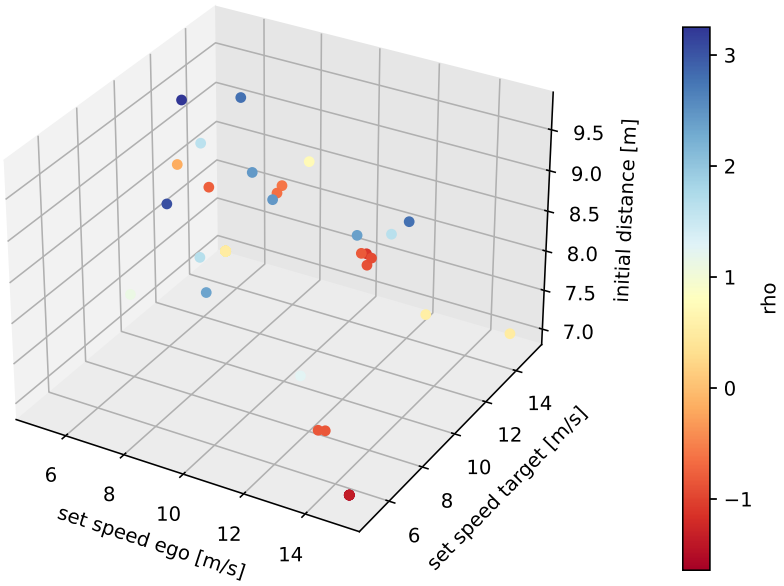

Figure 5.1: Each point is a feature space sample, the color represent the resulting metric and the size of the points correspond to the parameter delay

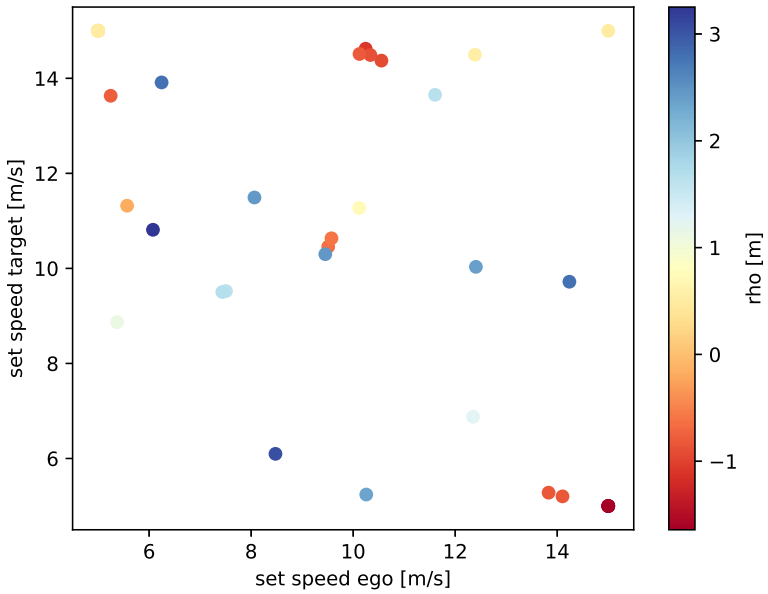

Figure 5.2: Each point is a feature space sample, the color represent the resulting metric and the size of the points correspond to the parameter delay

The second test was also performed in the *Braking target vehicle* scenario but instead with the MTL formulation $\varphi_2 = \square(d^{\mathrm{B}}_{\mathrm{margin,target}})$. The Bayesian optimization of the sampling gave rise to 16 counterexamples in this test also. The counterexample‘s location in the feature space can be seen in Fig. 5.3. It is clear in both figures that the falsification is correlated with high speed in the ego vehicle. In Fig. 5.4, it can be seen that no falsified samples occur when both vehicles have a low *set speed*. Also, the region of high speed for ego vehicle in combination with low speed for the target vehicle is populated with both verified and falsified samples, very close together.

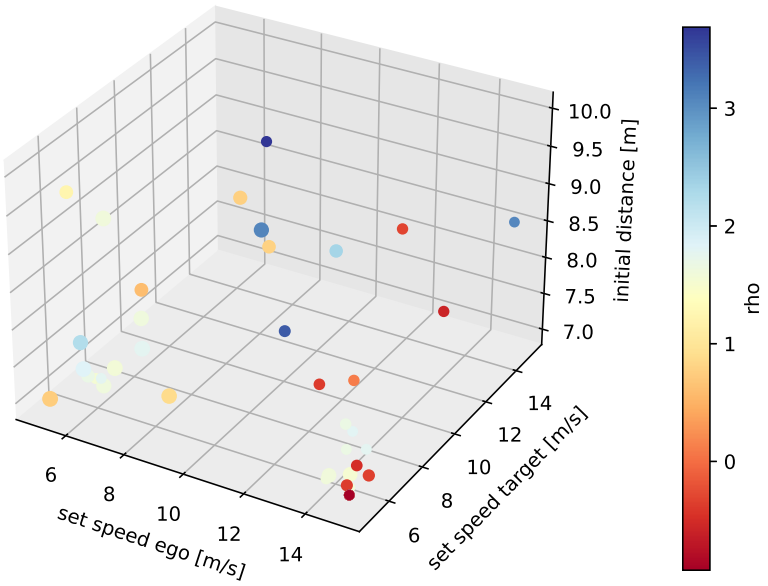

Figure 5.3: Each point is a feature space sample, the color represent the resulting metric and the size of the points correspond to the parameter delay

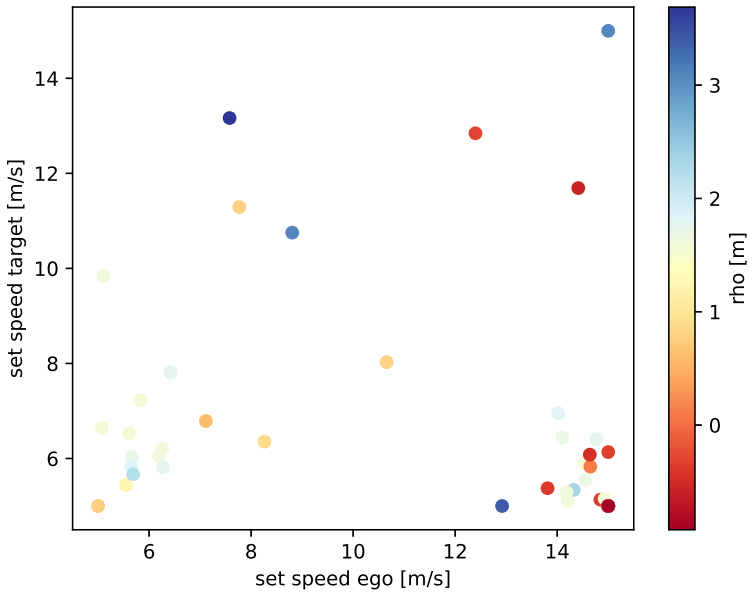

Figure 5.4: Each point is a feature space sample, the color represent the resulting metric and the size of the points correspond to the parameter delay

The third test in the results was performed on the *Target vehicle cutting in* scenario with the MTL formulation $\varphi_3 = \square(j_{\text{margin,ego}} \vee \neg\lozenge_{[-0.5,0]} d^{\text{A}}_{\text{margin,target}})$. The Bayesian optimization of the sampling gave rise to 19 counterexamples. The counterexample's locations in the feature space can be seen in Fig. 5.5. One thing to note is that the set speed of the ego vehicle is not in the figure, but is defined as *set speed target* plus 5 m/s, so they are relatively constant. However, there is a clear correlation between *set speed target* and the distance when cut in starts. High *set target speed* and cut in distance at 12-14 m, leads to falsified samples. Additionally, a correlation between the initial distance and distance when the cut in starts might exist. In the left plot of Fig. 5.6 there seems to be a sharp cut off *rho* with decreasing *distance when cut in*. For explanations of the different parameters in the feature space, see chapter 4.

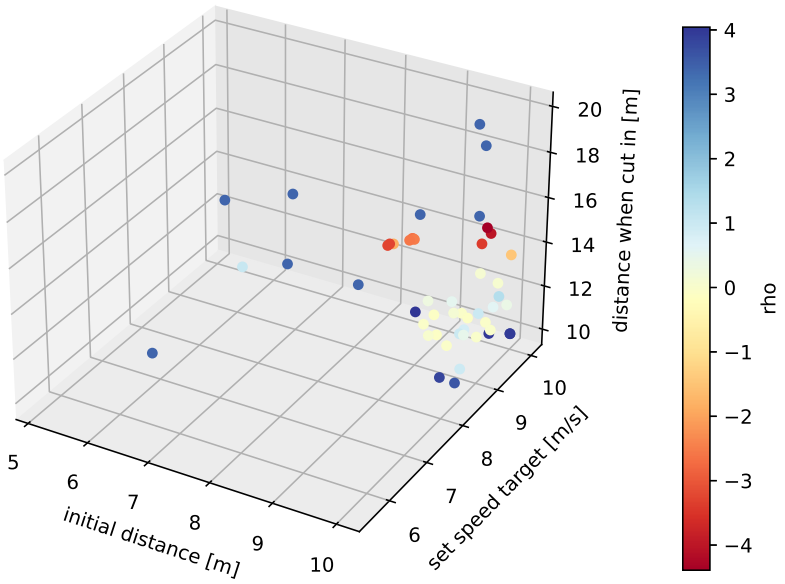

Figure 5.5: Each point is a feature space sample, the color represent the resulting metric and the size of the points correspond to the parameter delay

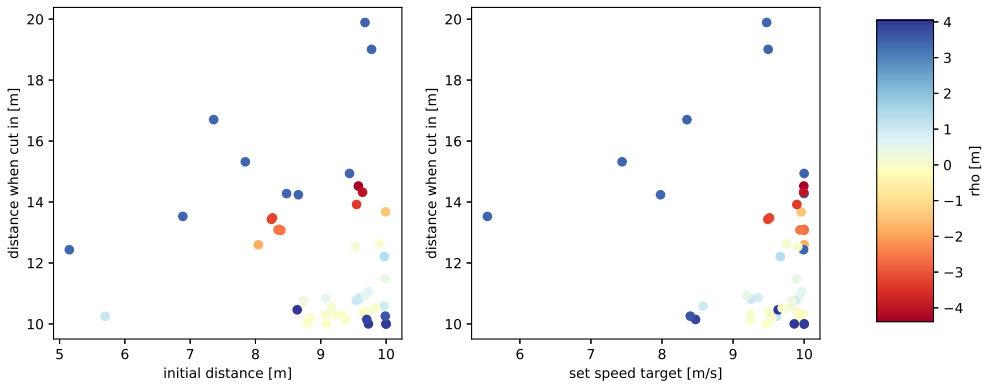

Figure 5.6: Each point is a feature space sample, the color represent the resulting metric and the size of the points correspond to the parameter delay

6
Discussion

This chapter comments on the method of the thesis, the simulation results, as well as the formulated MTL-specifications and scenarios. The method of the thesis, as stated in chapter 3, was to utilize a toolchain to evaluate a trajectory planner as a black-box system and in the process explore what effects MTL had on the process.

6.1 Formulating MTL specifications

In chapter 2.3, a general guideline is listed which states that the natural language can be converted to temporal logic by finding and defining atomic states. However, natural language can expressed ambiguously. The ambiguity could be approached by combining several formal language statements with, although, they should be evaluated separately to ensure that no statements are redundant. So the sense is that formalization for a big complex scenario should be viewed in the whole picture and maybe with small special events an unique requirement can be expressed. Alternatively, define well suitable safety requirements that can be tested in different scenarios to be able to cover the space of real world scenarios. Yet, it is clear that more investigations regarding natural language to formal language is needed. This for maybe be able to tell guidelines to make the formalization effective and which pit falls that should be avoided in this area.

The findings from test three seems to show the nuance of the specification by also having well satisfied samples in vicinity of falsified samples. This indicates that the part *motivated by traffic* of the MTL-statement in test three is was satisfied, in turn verifying the sample. More investigation regarding the best way of define natural language in formal language is needed. Nonetheless, this scenario test with MTL specification shows that MTL can be an effective way to include nuances by using temporal operators.

6.2 MTL in simulations

The results from the simulation shows that the toolchain was able to falsify the trajectory planner for the three selected requirements. The particular decisions that were made in formulating the MTL requirements did show a difference in the resulting falsification process. For the tests in *Braking target vehicle* scenarios, the simpler MTL requirement was falsified for a range of set speeds of the ego vehicle.

Meanwhile, the more complex MTL requirements is not falsified at low speed of the ego vehicle. This seems reasonable, due to $d^{\mathrm{B}}_{\mathrm{margin}}$ taking into consideration the velocity of the target vehicle, effectively compensating for it. However, this could be investigated further, since the exploration of ways to formulate an MTL requirement was not that extensive.

Additionally, from the results in test three, it is a clear correlation between initial distance and the distance when cut in was started. A shorter initial distance between the vehicles requires a cut in performed closer to the ego vehicle to produce a counterexample. This seems reasonable since initial distance affects the time it takes to accelerate enough so that the cut in maneuver is activated, so longer initial distances leads to cut in maneuvers at higher speeds.

6.3 Falsification of the trajectory planner

The aim of the project was to use the method to evaluate a trajectory planner in an isolated manner, to remove the impact other components could have on the performance, and to be able to isolate any unwanted behavior to one module. During the process of interfacing the module with the simulator it was discovered that it was not possible to apply the planner requests directly, but that a controller was needed. One could argue that this takes away somewhat from the aim of an isolated evaluation since the performance of the controller might effect the results. A solution to this could have been to instead of formulating requirements on the resulting behavior in the simulation environment (chapter 4) specify requirements directly on the output signal from the planning module. Since the environment is supposed to emulate a complete traffic experience, maybe this method is more appropriate when evaluating a full stack of vision, planning and control, since the situation would be closer to the actual use case.

The falsification process did produce several counter-examples. It would be interesting to construct these samples in the testing environment at Zenseact to explore if the results would be reproduced. The counter examples are in the case of MTL-requirement ϕ_3 not that interesting in themselves, since the requirement is in part constructed during the thesis work and is not directly a requirement on the product.

6.4 MTL as a robustness metric

In chapter 2, MTL is described and the distinction between quantitative and qualitative semantics is made. Quantitative semantics enables MTL statements to be evaluated to a robustness metric ρ that can be optimized with for example Bayesian optimization.

7
Conclusion

Reflecting on the stated research questions in the introduction the following conclusions were drawn:

- The difficulties with formulating MTL-requirements was to interpret vague natural language in a way that captured its nuance. That difficulty can also be seen as an upside since it forces the requirements to be well defined.
- The decision to use a temporal operator in the second MTL-requirement seems to have provided nuance in the counterexample evaluation.
- The advantages with the toolchain that was shown in this thesis work was that with no need to model the component to be tested, the module could be tested quite quickly when the toolchain was in place. Another positive aspect is the possibility to put requirements on the state of the environment and not just the outputs of the module. The most substantial draw-back on using black box testing on an internal module was that much work was put into creating the specific inputs to it, introducing many opportunities for errors.
- A meaningful robustness metric was utilized, the quantitative semantics of MTL.

Future work should most certainly include more extensive exploration of MTL formulations in simulation. One interesting aspect to implement in the MTL implementation would be to allow the interval limits to be evaluated during execution for even more nuance.

Bibliography

[1] N. Rajabli, F. Flammini, R. Nardone, and V. Vittorini, "Software verification and validation of safe autonomous cars: A systematic literature review," *IEEE Access*, 2020, ISSN: 21693536. DOI: 10.1109/ACCESS.2020.3048047.

[2] S. Kuutti, R. Bowden, and S. Fallah, "Weakly supervised reinforcement learning for autonomous highway driving via virtual safety cages," *Sensors*, vol. 21, pp. 1–16, 6 Mar. 2021, ISSN: 14248220. DOI: 10.3390/s21062032.

[3] J. Wang, L. Zhang, Y. Huang, and J. Zhao, *Safety of autonomous vehicles*, 2020. DOI: 10.1155/2020/8867757.

[4] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," Aug. 2017. [Online]. Available: http://arxiv.org/abs/1708.06374.

[5] J. Suh, B. Kim, and K. Yi, "Design and evaluation of a driving mode decision algorithm for automated driving vehicle on a motorway," *IFAC-PapersOnLine*, vol. 49, no. 11, pp. 115–120, 2016, 8th IFAC Symposium on Advances in Automotive Control AAC 2016, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2016.08.018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896316313398.

[6] A. Serban, E. Poll, and J. Visser, "A standard driven software architecture for fully autonomous vehicles," *Journal of Automotive Software Engineering*, vol. 1, p. 20, 1 2020. DOI: 10.2991/jase.d.200212.001.

[7] M. Khatun, M. Glaß, and R. Jung, "Scenario-based extended hara incorporating functional safety & sotif for autonomous driving," pp. 53–59, Mar. 2021. DOI: 10.3850/978-981-14-8593-0_5225-cd.

[8] S. Riedmaier, T. Ponn, D. Ludwig, B. Schick, and F. Diermeyer, *Survey on scenario-based safety assessment of automated vehicles*, 2020. DOI: 10.1109/ACCESS.2020.2993730.

[9] D. J. Fremont, E. Kim, Y. V. Pant, S. A. Seshia, A. Acharya, X. Bruso, P. Wells, S. Lemke, Q. Lu, and S. Mehta, "Formal scenario-based testing of autonomous vehicles: From simulation to the real world," Mar. 2020. [Online]. Available: http://arxiv.org/abs/2003.07739.

[10] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh, M. Vazquez-Chanlatte, and S. A. Seshia, "Verifai: A toolkit for the design and analysis of artificial intelligence-based systems," Feb. 2019. [Online]. Available: http://arxiv.org/abs/1902.04245.

[11] E. M. Clarke and J. M. Wing, "Formal methods: State of the art and future directions," *ACM Computing Surveys*, vol. 28, pp. 626–643, 4 1996, ISSN: 03600300. DOI: 10.1145/242223.242257.

[12] S. A. Seshia, "Bridging simulation and the real world with verifai and scenic." [Online]. Available: http://vehical.org.

[13] D. D. Leng and F. Heintz, "Approximate stream reasoning with metric temporal logic under uncertainty," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 2760–2767, 2019, ISSN: 2159-5399. DOI: 10.1609/aaai.v33i01.33012760.

[14] J. Ouaknine and J. Worrell, "Some recent results in metric temporal logic," 2008.

[15] Z. Xu, K. Yazdani, M. T. Hale, and U. Topcu, "Differentially private controller synthesis with metric temporal logic specifications," Sep. 2019. [Online]. Available: http://arxiv.org/abs/1909.13294.

[16] R. Koymans, "Specifying real-time properties with metric temporal logic," 1990, pp. 255–299.

[17] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications."

[18] A. Rodionova, I. Alvarez, M. S. Elli, F. Oboril, J. Quast, and R. Mangharam, "How safe is safe enough? automatic safety constraints boundary estimation for decision-making in automated vehicles," Institute of Electrical and Electronics Engineers Inc., 2020, pp. 1457–1464. DOI: 10.1109/IV47402.2020.9304756.

[19] Z. Ramezani, N. Smallbone, M. Fabian, and K. Akesson, "Evaluating two semantics for falsification using an autonomous driving example," vol. 2019-July, Institute of Electrical and Electronics Engineers Inc., Jul. 2019, pp. 386–391, ISBN: 9781728129273. DOI: 10.1109/INDIN41052.2019.8972229.

[20] M. Hekmatnejad, S. Yaghoubi, A. Dokhanchi, H. B. Amor, A. Shrivastava, L. Karam, and G. Fainekos, "Encoding and monitoring responsibility sensitive safety rules for automated vehicles in signal temporal logic," Association for Computing Machinery, Inc, Oct. 2019, ISBN: 9781450369978. DOI: 10.1145/3359986.3361203.

[21] N. Aréchiga, "Specifying safety of autonomous vehicles in signal temporal logic," 2019.

[22] S. Maierhofer, A. K. Rettinger, E. C. Mayer, and M. Althoff, "Formalization of interstate traffic rules in temporal logic," Institute of Electrical and Electronics Engineers Inc., 2020, pp. 752–759. DOI: 10.1109/IV47402.2020.9304549.

[23] A. Rizaldi, J. Keinholz, M. Huber, J. Feldle, F. Immler, M. Althoff, E. Hilgendorf, and T. Nipkow, "Formalising and monitoring traffic rules for autonomous vehicles in isabelle/hol," vol. 10510 LNCS, Springer Verlag, 2017, pp. 50–66, ISBN: 9783319668444. DOI: 10.1007/978-3-319-66845-1_4.

[24] E. Zapridou, E. Bartocci, and P. Katsaros, "Runtime verification of autonomous driving systems in carla," pp. 172–183, 2020. [Online]. Available: http://www.springer.com/series/7408.

[25] *Vehicles*. [Online]. Available: https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/Vehicles.html.

[26] D. J. Fremont, E. Kim, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: A language for scenario specification and data generation," Oct. 2020. [Online]. Available: http://arxiv.org/abs/2010.06580.

[27] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh, M. Vazquez-Chanlatte, and S. A. Seshia, "Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems," *arXiv*, 2019, ISSN: 23318422.

[28] P. I. Frazier, *A tutorial on bayesian optimization*, 2018. arXiv: 1807.02811 [stat.ML].

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY