Leveraging Answer Set Programming for Continuous Monitoring, Fault Detection, and Explanation of Automated and Autonomous Driving Systems

Lorenz Klampfl[1]
CD Laboratory QAMCAS, Institute of Software Technology, Graz University of Technology, Austria

Franz Wotawa
Institute of Software Technology, Graz University of Technology, Austria

Abstract

Recent advancements in automated and autonomous driving systems have facilitated their integration into modern vehicles, enabling them to accurately perceive their surroundings and support or even fully undertake complex driving tasks. Given the complexity and unpredictable nature of driving environments and traffic situations, ensuring the correct behavior of such systems is essential to prevent hazardous situations, increase user acceptance, and avoid human harm. However, the increased complexity of these systems and the extensive search space of possible scenarios introduce significant challenges to testing and real-time fault management. Hence, besides rigorous testing during the development phase, there is a need for additional validation and verification during operation. This paper proposes utilizing Answer Set Programming (ASP), a form of declarative programming, for continuous real-time monitoring, fault detection, and explanation to ensure the correct functioning of automated and autonomous driving systems. Our approach aims to enhance the reliability and safety of such systems by detecting violations and providing explanations that can support fault-adaptive control or mitigation strategies. We demonstrate the effectiveness of our methodology across diverse scenarios executed within a simulation environment, discuss the main challenges encountered, and outline future research directions.

2012 ACM Subject Classification General and reference → Validation

Keywords and phrases Autonomous Driving, Answer Set Programming, Continuous Monitoring

Digital Object Identifier 10.4230/OASIcs.DX.2024.10

Funding The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology, and Development, and the Christian Doppler Research Association is gratefully acknowledged.

1 Introduction

In recent years, newly developed algorithms and software components utilizing artificial intelligence (AI) have become central components integrated into modern vehicles. Smart sensors reliably perceiving the vicinity of autonomous vehicles, as well as advanced algorithms for path planning and handling complex driving situations, successfully demonstrate how Advanced Driver-Assistance Systems (ADAS) and Autonomous Driving (AD) technologies can contribute to the vision of safe, efficient, and comfortable transportation. However, guaranteeing safety and robustness under all circumstances during driving remains an open challenge. In both engineering and research, the necessity for comprehensive testing and validation to comply with established standards and regulations, such as UN Regulation No. 157 [3], and

[1] Corresponding Author

to ensure the reliability and trustworthiness of these systems has been widely recognized, e.g., Wotawa and colleagues [20], Wotawa [17], and Koopman et al. [11]. It is essential to consider that autonomous vehicles operate within domains characterized by unpredictability and uncertainty, necessitating robust safety measures and testing methodologies. It is evident that traditional mileage validation is impractical, as expressed by Kalra and Paddock [7], stating that autonomous vehicles must operate 275 million miles for verification. As a result, new methodologies have emerged over the last few years, emphasizing the advantages of using virtual validation approaches and simulation frameworks to test ADAS/AD effectively, e.g., Klampfl et al. [8], Schuldt et al. [13] or Klück and colleagues [10]. Nevertheless, despite the considerable advancements virtual validation methods have introduced compared to on-road testing, these approaches also come with limitations. Considering the range of parameters involved, e.g., environmental conditions, traffic signs, or road characteristics, alongside the complex dynamics between other road users and the autonomous vehicle, verifying the correct behavior for all scenarios within reasonable time seems unfeasible. Furthermore, there may be unknown factors or a combination of factors that could lead to ADAS or AD system malfunctioning. Unfortunately, numerous incidents in recent years have underscored the complexities and challenges in testing and verifying autonomous driving systems. This highlights the need for innovative solutions that ensure the continuous safe operation of autonomous vehicles and their correct responses to unforeseen or unknown events.

Our Approach

In this paper, we aim to enhance the safety and reliability of ADAS/AD by introducing an additional layer of quality assurance that can be deployed during vehicle operation and is responsible for continuously monitoring and validating the behavior of autonomous vehicles during operation, complementary to strict testing during the development phase of automated and autonomous driving systems. Figure 1 depicts a high-level overview of our proposed approach, combining vehicle sensors from an autonomous car or simulation data from a simulation environment and a continuous validation setup (CVS) comprising formalized logic-based expectations for vehicle behavior and solving mechanisms for monitoring, identifying, and explaining potential ADAS/AD misbehavior. Through repeated interactions between the autonomous vehicle or a simulation environment and the CVS, we aim to find discrepancies between expected and actual behavior. It is worth noting that the general principles of the CVS can be similarly implemented across different layers of abstraction, e.g., raising and diagnosing sensor faults or recognizing physical impossibilities of objects detected by the perception system.

Our Contributions

Within this paper, we contribute to the topic of safe and robust behaviors of deployed automated and autonomous driving systems based on the actions they perform. Specifically, we investigate four different prototypical Automated Lane Keeping Systems (ALKS). In addition, we assume that vehicle sensors work as expected and that the perception system correctly detects and classifies all objects in the vicinity of the autonomous vehicle. This assumption is ensured by generating virtual scenarios executed within a simulation environment, i.e., Esmini[2], where ground truth data about the vehicle's state, as well as positions and states of other road users and objects, is available. Our primary contribution includes the development

[2] `https://github.com/esmini/esmini`

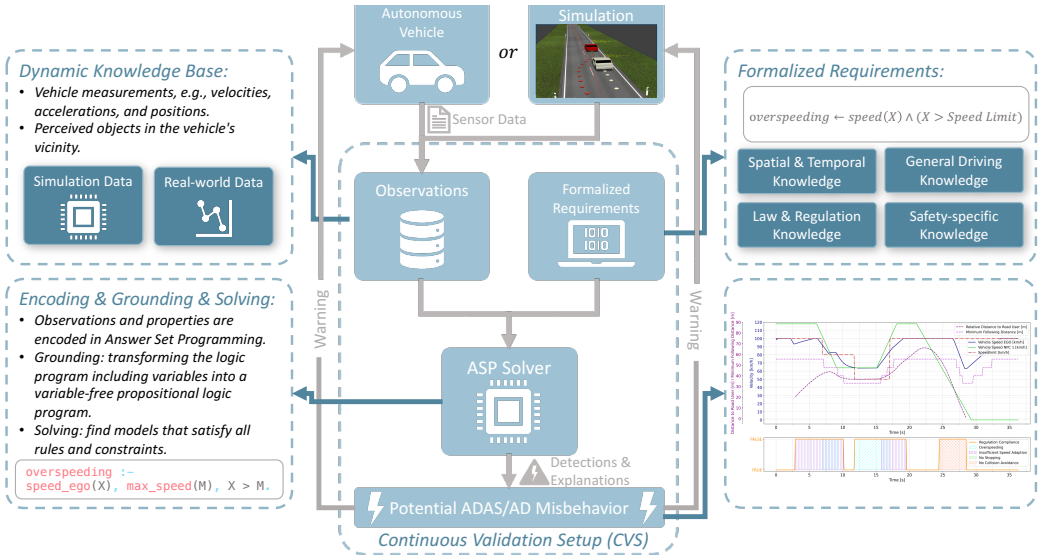


Figure 1 High-level schematic showing the general concept of our approach and how the framework's different components interact.

of a model incorporating formalized logic-based properties that an ALKS has to comply with to ensure correct behavior. In more detail, we provide a model covering parts of the requirements stated in regulation UN R157 [3] for Automated Lane Keeping Systems. We leverage Answer Set Programming (ASP) [2] and the ASP solver Clingo [3] [4], which is using an extended version of Prolog [1] as input language, to encode the regulations an ALKS has to comply with. Using a declarative programming approach like ASP, combined with advanced solvers like Clingo, allows us to efficiently monitor, detect faults, explain possible misbehavior, and extend our model with additional or new rules without implementing an entirely new model. Furthermore, we demonstrate the effectiveness of our model by developing a framework and corresponding interfaces to integrate our requirement compliance model into a simulation framework, enabling continuous interaction between our model and the simulation.

Outline

The structure of our paper is as follows: First, we give a brief introduction to ASP, followed by a description of our proposed model, detailing its design, implementation, and evaluation. Finally, we discuss the implications of our findings and highlight the potential of ASP to enhance the safety and reliability of Advanced Driver Assistance Systems (ADAS) and autonomous driving (AD).

2 Background

The following paragraphs outline the theoretical and technical foundation for our research and experimental evaluation, including a brief overview of ASP, our simulation framework, and continuous validation setup.

[3] `https://potassco.org`

2.1 Answer Set Programming (ASP)

Answer Set Programming is a programming approach that emphasizes specifying the problem in a declarative way rather than in an imperative. ASP originates from several fields, including deductive databases, logic programming, and knowledge representation and reasoning. It is applied in various domains, such as planning, configuration, decision support, model checking, robotics, and autonomous driving. In recent years, ASP has gained particular attention in the automotive domain. For instance, Suchan and colleagues introduced a modular framework for visual sensemaking using ASP and a qualitative model for tracking objects in traffic, including occlusion handling, as detailed in their works [15] and [16]. Similarly, Wotawa and Klampfl utilized logic models and ASP to identify critical situations in autonomous driving during operation [9]. Additionally, Gilpin proposed methods for monitoring autonomous driving systems and comparing them with driving knowledge, regulatory knowledge, and safety requirements [5].

ASP is based on the stable model (or answer set) semantics of logic programming. Given a program P, its answer sets can be understood as the intended models of P, each representing a distinct scenario or world described by P. An ASP program is a collection of rules of the form:

$$a_0 \leftarrow a_1, ..., a_m, not\ a_{m+1}, ..., not\ a_n. \tag{1}$$

where $(a_0, ..., a_n)$ are atoms and *not* represents negation as failure. The semantics of ASP facilitate the representation of default assumptions and exceptions, which is crucial for modeling the dynamic and uncertain environments encountered by autonomous vehicles.

Let us consider the following example of approaching a traffic light. We can express a rule stating that we are only allowed to drive when the traffic light is green as follows:

$$colorDrive(green). \tag{2}$$

$$stop \leftarrow trafficLightColor(X), not\ colorDrive(X). \tag{3}$$

$$noStoppingTrafficLight \leftarrow stop, not\ stoppingInitiated. \tag{4}$$

Here, (2) encodes the fact that we are only allowed to drive if the traffic light color is green. (3) is the formalization of the rule that stopping is required using ASP syntax with *not* representing negation as failure, i.e., *not* $colorDrive(X)$ is $TRUE$ if all attempts to prove $colorDrive(X)$ fail. Therefore, for all other traffic light colors X, we would derive the atom *stop* indicating that our autonomous vehicle needs to stop at the present traffic light. Furthermore, with rule (4), where *not stoppingInitiated* represents, for instance, an observation delivered by a sensor of an autonomous vehicle, we can derive a faulty behavior of the system under test and its corresponding explanation, i.e., the autonomous vehicle has not stopped at a traffic light although the traffic light color was not green.

It is worth mentioning that ASP has been already used to implement Model-based Diagnosis (see [18, 19]). In the context of this paper, we utilize MBD for implementing fault detection based on models, i.e., the formalized regulations, and reasoning.

Solving Procedure

The solving procedure for ASP can be divided into two main phases: grounding and solving.

1. **Grounding:** The grounding phase translates a high-level ASP program, which may contain variables and complex expressions, into a variable-free propositional logic program. This process involves generating all possible instances of the rules by substituting variables

with constants from the domain. The challenge in grounding is to manage the exponential growth in the size of the program, which can lead to what is known as the "grounding bottleneck." To mitigate this, grounding tools included in solvers like Clingo apply intelligent techniques to reduce the size of the grounded program, such as eliminating irrelevant rules and facts.

2. **Solving:** Once the program is grounded, the solving phase involves searching for answer sets of the grounded program that satisfy all rules and constraints included in the logic program. Therefore, in our case, our answer sets represent potential misbehaviors of the system under test and its corresponding explanations.

For a more detailed explanation of the grounding and solving procedure, we refer the interested reader to the respective literature, e.g., [4].

2.2 Simulation Framework and Continuous Validation Setup

Figure 1 illustrates a high-level overview of our approach's concept, including the different artifacts implemented and their interactions. Our methodology comprises two main components: the simulation framework or autonomous vehicle and the continuous validation setup (CVS). The simulation framework, representing the autonomous vehicle in our use cases, is responsible for providing sensor data. This data is used to generate a dynamic knowledge base during runtime, which includes all necessary information to determine whether the ALKS under test complies with all implemented regulations and expected behaviors. Measurements of particular importance include accelerations and the positions of all objects in the simulation environment. It is important to note that during our experiments, we only used data gathered from simulation to evaluate our approach. This ensures that all additional road users are correctly identified, particularly regarding their positions, allowing us to evaluate the behavior of the four investigated ALKSs without interference from detection and classification algorithms. As already mentioned, we decided to use Esmini as our simulator. Esmini is an environment simulator for replaying ASAM OpenSCENARIO [4] files, which has become a widely adopted standard for virtual development, testing, and validation in the autonomous driving domain. We utilize Esmini to load, execute, and collect simulation data during scenario execution. Detailed descriptions of the scenarios used in our experiments, as well as the integrated ALKS controllers, are provided in subsequent sections. The CVS comprises the dynamic knowledge base, the formalized requirements (see Table 1), the interface to our ASP solver Clingo, and a result post-processing pipeline. At each simulation time step, i.e., every 0.1 seconds, the CVS retrieves simulation data, encodes the provided observations to ASP syntax, grounds our logic program, and solves it by finding all answer sets that satisfy our rules and constraints. Since simulation data cannot be directly utilized by the CVS, an encoding approach similar to that proposed by Jahaj et al. [6] has been implemented. Specifically, within each time step, a set of adaptors is used to convert the incoming sensor data into ASP facts. For example, an adaptor may take the input from the speed sensor and convert the received data into the fact `speed(130)`. Therefore, for each adaptor, we have a function that implements the conversion from the sensor domains to ASP facts.

[4] `https://www.asam.net/`

3 UN R157 Requirement Formalization

Table 1 showcases the requirements we implemented to test our methodology. As seen in the table, only a subset of requirements was included, namely those dealing with system activation, general traffic rules, and the dynamic driving task. Table 1 depicts from which chapter the requirement was extracted, a short description of the requirement, as well as a general formalization of the requirement demonstrating how we implemented it within our logic model. It is worth noting that all requirements were implemented within one model. With this, we ensure that at every time step, all requirements are checked and possible faults are detected and explained appropriately. As an example, let us consider the first requirement in the table, i.e., *System Activation*. Within regulation UNR157, it is stated that an ALKS system can only be activated under certain conditions, i.e., the automated or autonomous vehicle is driving on a specific road characterized by a physical separation dividing the traffic moving in opposite directions and preventing traffic from cutting across the street. Furthermore, road types where pedestrians and cyclists are prohibited are a prerequisite for successfully activating the ALKS. Concerning vehicle speed, the ALKS is only allowed to be activated up to operational speeds of 60 km/h. However, if the ALKS includes the functionality of performing Minimum Risk Maneuvers (MRM), the maximum operational speed increases to 130 km/h. Those requirements are implemented with three dedicated rules as stated in Table 1. Each rule would infer *prohAct* in case of a prohibited activation of the ALKS system. The first rule evaluates if the road type we are actually driving on is suitable for activating the ALKS, which is encoded with the fact *alksRoad*, and if the ALKS system is activated, i.e., *alksActive*. In case *alksActive* is true, and *alksRoad* is false, the term *not alksRoad* will evaluate to true, resulting in deriving the fact *prohAct*. Similarly, the second and third rule evaluate if *alksActive* is true and check if the vehicle speed is within the allowed range. Furthermore, the second rule includes the term *not alksMRM*, which evaluates to true in case the ALSK system is not capable of performing a Minimum Risk Maneuver. All other requirements are implemented similarly. For a detailed description of the UN R157 requirements, we refer the interested reader to [3].

4 Unit Testing Evaluation

In order to demonstrate the correct implementation and outcomes of our CVS, we conducted two separate evaluation approaches: a unit testing approach as well as an experimental evaluation using a dedicated simulation framework (see Chapter 5). We implemented a total of 41 unit tests, covering all requirements outlined in Table 1. To leverage Clingo's dedicated interfaces to Python, we used Python[5], version 3.12.2, and the PyTest[6] framework, version 7.4.3, to write and execute our unit tests. PyTest is a simple yet powerful framework that simplifies the process of writing and running tests by providing a clean and readable syntax. It also supports parameterized testing, allowing us to test, for instance, multiple vehicle speed values within a single unit test.

Table 2 presents three example unit tests, describing their parameters and values. Each unit test comprises different parameters and their respective values, meaning that each test evaluates the model based on specific parameters and their combinations. For the unit testing framework, we implemented dedicated Python classes and functions to encode these

[5] `https://www.python.org`
[6] `https://docs.pytest.org/en/8.2.x/index.html`

Table 1 Overview depicting the implemented requirements included in UN Regulation No. 157 [3].

UN R157 Requirements		
Reference	**Name**	**Short Description**
Formalization		
Intro 5.2.3.1.	System Activation	The system can be activated only under certain conditions, e.g., suitable roads or speeds up to 60 km/h. If the ALKS is equipped with a Minimum Risk Maneuver(MRM) functionality, the maximum speed limit for activation increases to 130 km/h.
$prohAct \leftarrow alksActive \wedge not\ alksRoad.$ $prohAct \leftarrow alksActive \wedge speed(X) \wedge (X > 60) \wedge not\ alksMRM.$ $prohAct \leftarrow alksActive \wedge speed(X) \wedge (X > 130).$		
5.1.2	Traffic Rules	The system shall comply with traffic rules, for instance, obey speed limits. Additional regulations, such as the requirement to stop at stop signs or traffic lights, are encoded similarly.
$overspeeding \leftarrow speed(X) \wedge speedSign(Limit) \wedge (X \geq Limit).$ $overspeeding \leftarrow maxSpeed(Country, Streettype, Limit) \wedge speed(X) \wedge (X \geq Limit).$		
5.2.1	Dynamic Driving Task	The system shall ensure the vehicle's lateral stability and position within its lane, avoiding unintentional lane departures. Longitudinal rules are implemented similarly considering accelerations. $Thres$ defines at which distance to a lane marking it is considered as crossing.
$unintLaneChane \leftarrow distLeft(DistLeft) \wedge (DistLeft < Thres) \wedge not\ laneChangeInitiated.$ $unintLaneChane \leftarrow distRight(DistRight) \wedge (DistRight < Thres) \wedge not\ laneChangeInitiated.$		
5.2.2	Dynamic Driving Task	The system shall appropriately adjust the speed and the lateral position within its lane in case another vehicle is detected in an adjacent lane. The system shall not accelerate and ensure that the vehicle is not too close to the lane boundaries.
$noAdjustment \leftarrow nextToDrivingLane(Vehicle) \wedge not\ keepingSpeed.$ $noAdjustment \leftarrow nextToDrivingLane(Vehicle) \wedge not\ centricPosition.$		
5.2.3	Dynamic Driving Task	The system shall control the vehicle's speed and adapt it when certain conditions are present, e.g., environmental conditions or a vehicle in front. $sufficientSpeedAdaption$ is considered $True$ when the vehicle speed is lower than the allowed speed concerning properties like environmental conditions (speed should be reduced appropriately) as well as if the distance to the detected vehicle in front is greater than the minimum following distance to the vehicle based on the table depicted in [3], paragraph 5.2.3.3.
$noSpeedAdjustment \leftarrow envCondition(Env) \wedge not\ sufficientSpeedAdaption(Env).$ $noSpeedAdjustment \leftarrow roadUserOnLane(Vehicle) \wedge not\ sufficientSpeedAdaption(Vehicle).$		
5.2.4	Dynamic Driving Task	The system shall bring the vehicle to a complete stop behind a stationary object. $collisionAvoidable$ is considered $True$ if the relative distance to the object is greater than the minimum following distance.
$noStoppingManeuver \leftarrow objectOnLane(Obj) \wedge collisionAvoidable \wedge not\ stoppingInitiated.$		
5.2.5	Dynamic Driving Task	The system shall detect the risk of a collision and shall perform appropriate maneuvers, e.g., lane change or emergency maneuver. $collisionAvoidable$ is considered $True$ if the relative distance to the object is greater than the minimum following distance.
$noCollisionAvoidance \leftarrow$ $objectOnLane(Obj) \wedge not\ collisionAvoidable \wedge not\ collisionAvoidanceInitiated.$		

Table 2 Overview of some of the executed unit tests based on the requirements stated within Table 1.

UN R157 Requirements - Unit Tests		
Reference	**Name**	**Description**
Inputs		
Intro 5.2.3.1.	System Activation	Three dedicated unit tests were implemented covering the testing of a prohibited ALKS activation for unsuitable road types as well as for a prohibited activation at certain speeds dependent on the availability of Minimum Risk Maneuver (MRM) functionality.
$RoadTypes = [0, ruralRoad, city, motorway]$ $VehicleSpeeds = [0, 50, 59.999, 60, 60.001, 100, 129.999, 130, 130.001, 150, 200]$ $MRMFunctionality = [True, False]$		
5.1.2	Traffic Rules	Three unit tests were performed to check if overspeeding is correctly identified. This includes a differentiation between the situation where a speed sign is present and not as well as signs that restrict the speed to a lower or higher maximum allowed speed.
$RoadTypes = [motorway]$ $VehicleSpeeds =$ $[0, 95, 99.999, 100, 100.001, 120, 125, 129.999, 130, 130.001, 135, 139.999, 140, 140.001, 145, 200]$ $SpeedSignLimit = [notavailable, 100, 140]$		
5.2.3	Dynamic Driving Task	The system shall control and adapt the speed of the vehicle when certain conditions are present, e.g., environmental conditions or a vehicle in front. A dedicated unit test was implemented for both. In case of environmental conditions like rain, we assume that a sufficient speed reduction is achieved when the vehicle speed is equal to or lower than 75% of the maximum allowed speed, i.e., for motorways in Austria where the speed limit is 130km/h, this would result in 97.5 km/h. For a detected road user, sufficient speed reduction is achieved when the vehicle obeys the minimum following distance based on the table depicted in [3], paragraph 5.2.3.3.
$RoadTypes = [motorway], EnvironmentCondition = [rain]$ $VehicleSpeeds =$ $[0, 50, 69.999, 70, 70.001, 79.999, 80, 80.001, 97.499, 97.500, 97.501, 100, 129.999, 130, 130.001, 150, 200]$ $DistanceToRoadUser = [notavailable, 40]$		

parameters and values into the appropriate ASP syntax for passing them to the Clingo solver and for post-processing the results. As shown in the provided examples, we included boundary values within the allowable ranges to identify potential errors and defects in our model. This approach ensures that any issues arising from incorrect assumptions are detected.

All 41 unit tests were successfully passed, providing a solid foundation for further evaluation within a simulation environment that includes an actual ALKS controller and test scenarios.

5 Experimental Evaluation

Building on the results obtained from unit testing, we conducted an extensive experimental evaluation to demonstrate the applicability of our proposed methodology within a simulation environment, including a vehicle equipped with a prototypical ALKS. As previously mentioned, we established a simulation framework using Esmini and developed dedicated interfaces to integrate our logic model. These interfaces facilitate continuous communication between the simulation environment and the CVS, ensuring that sensor data is transmitted seamlessly.

The continuous validation setup (CVS) processes the received simulation data every 0.1 seconds. This involves encoding the data into the appropriate syntax and determining whether potential misbehavior is occurring. To test our model on actual ALKS controllers, we integrated four different controllers available within Esmini. Each controller was subjected to 977 automatically executed and evaluated scenarios, totaling in 3,908 test scenarios.

The following sections provide an overview of several example scenarios, detailing the maneuvers involved, the integrated ALKS controllers, and the evaluation outcomes. This thorough experimental evaluation underscores the practical applicability of our methodology, showcasing its effectiveness in ensuring safe and reliable behavior in automated and autonomous driving systems.

5.1 ALKS Controllers

Advanced Lane Keeping Systems are designed to control the lateral and longitudinal movement of a vehicle for extended periods without human intervention. Since obtaining sophisticated controllers for automated or autonomous driving from companies is challenging due to their proprietary nature, it is fortunate that Esmini includes several pre-built controllers for vehicle control within their simulations framework. According to Esmini's user guide, various controllers are available, such as an interactive control using a simple vehicle model that can be operated via keyboard inputs and more advanced controllers based on a ghost concept, where a ghost-twin performs planned events in a few seconds ahead, and the vehicle under control mimics its behavior. In addition, Esmini provides a pre-built ALKS controller named *ALKS_R157SM*, inspired by the safety models described in UN Regulation No. 157 [3]. This controller offers four different modes (safety models), though it is important to note that these implementations are preliminary and experimental and not officially commissioned ALKS controllers. The four safety models for the controller are described as follows:

1. **Regulation:** This model includes the characteristics of the expected system performance, which is stated within paragraph 5.2.5.2 in [3].
2. **ReferenceDriver:** Within Annex 4 - Appendix 3 of [3], a reference model of a "typical" human driver is specified, which is represented by this safety model.
3. **RSS:** Responsibility-Sensitive Safety: This mode includes the proposed performance model described by Shalev-Shwartz and colleagues [14].
4. **FSM:** Fuzzy Safety Model was introduced and described by Konstantinos et al. [12].

Since the objective of this paper is not to describe the inner workings of each safety model and the integrated ALKS controller, we refer interested readers to the respective literature. However, all these controllers share the capability of maintaining longitudinal and lateral stability and reacting appropriately to other road users. However, none of the models has advanced steering and emergency maneuver capabilities. In the next section, an overview of four example use cases and a description of the scenario properties will be given.

5.2 Test Scenarios

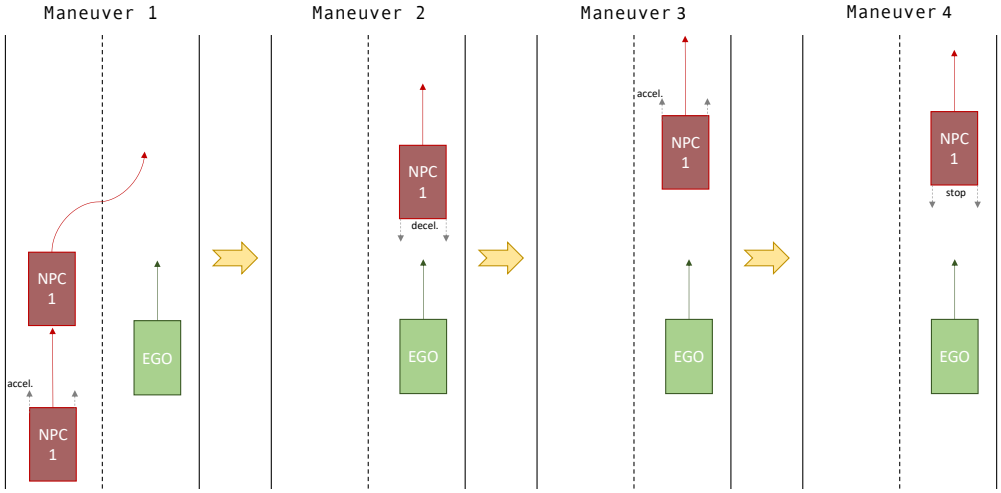
All of the executed and evaluated test scenarios are based on the ASAM OpenSCENARIO and ASAM OpenDRIVE [7] standard. OpenSCENARIO is a widely adopted standard for defining the dynamic part of complex traffic scenarios in a detailed and structured manner. Furthermore, it provides an extensible framework for specifying the behavior of all entities in

[7] https://www.asam.net/

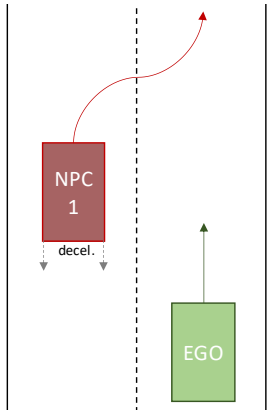
the simulation, e.g., additional road users, pedestrians, and environmental factors. This allows us the precise modeling of various driving situations, enabling thorough testing and validation of automated driving systems under a variety of conditions. In contrast, OpenDRIVE is used to define the static properties of the scenario, e.g., road networks, traffic signs, or road geometry, ensuring that the environment is accurately represented. Both standards interact closely with each other, as OpenSCENARIO uses the road network data to position and move all entities within the simulation. Utilizing both of these well-established standards ensures that our test scenarios are not only realistic and reproducible but also compatible with industry practices. The integration of these standards enhances the reliability and relevance of our experimental evaluation, providing a comprehensive framework for testing and validating the behavior of the integrated ALKS with our proposed methodology for continuous monitoring, fault detection, and explanation using ASP.

Although we generated a total of 977 scenarios per ALKS controller for evaluating our approach, in this paper, we want to highlight four representative test scenarios and the results we obtained from them. Figure 2 illustrates the maneuvers performed within each scenario when executed within Esmini. Before detailing the sample scenarios, it is important to note that *NPC1* represents an additional road user within the scenario whose trajectories and actions are predefined within the OpenSCENARIO files. *EGO* represents the automated vehicle equipped with the previously explained ALKS controller that should accurately react to maneuvers performed by *NPC1*. Below, we provide a brief description of each scenario along with the expected behavior of the automated vehicle.

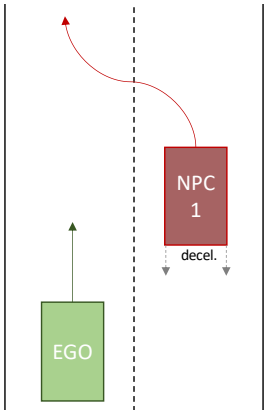
1. **ALKS R157 Test Scenario (Figure 2a)**
   Scenario Description: This scenario is already included within Esmini, however, we slightly adapted the starting positions and vehicle speeds as well as inserted speed limit signs. As depicted, the scenario consists of four separate maneuvers executed consecutively. The *EGO* vehicle is driving in the right lane and is overtaken by *NPC1*, which accelerates and performs a cut-in maneuver in front of the automated vehicle. After the successful cut-in, *NPC1* decelerates in front of the *EGO* vehicle. In the last two maneuvers of this scenario *NPC1* accelerates before coming to a complete stop.
   Expected EGO Behavior: After the first cut-in, a safe following distance should be re-established. During the deceleration phase in maneuver two, the *EGO* should react accordingly and also decelerate. For the stopping maneuver, *EGO* should detect the obstacle in front and come to a stop as well since the controllers are not equipped with lane change capabilities. Within the complete scenario, the *EGO* should avoid speed limit violations.
2. **Cut-In (Right) & Deceleration (Figure 2b)**
   Scenario Description: The *EGO* vehicle is traveling at a constant speed in its initial lane. Vehicle *NPC1* starts in the left adjacent lane with a longitudinal offset positioned ahead of the *EGO*. *NPC1* then performs a cut-in maneuver into the *EGO*'s lane and begins decelerating.
   Expected EGO Behavior: Initially, the *EGO* should continue driving without considering *NPC1* since it is in an adjacent lane. Once *NPC1* cuts in front of the *EGO*, the *EGO* should detect *NPC1* as a target object and maintain a safe following distance behind it.
3. **Cut-In (Left) & Deceleration (Figure 2c)**
   Scenario Description: The *EGO* vehicle is traveling at a constant speed in its initial lane. Vehicle *NPC1* starts in the right adjacent lane with a longitudinal offset positioned ahead of the *EGO*. *NPC1* then performs a cut-in maneuver into the *EGO*'s lane and begins decelerating.



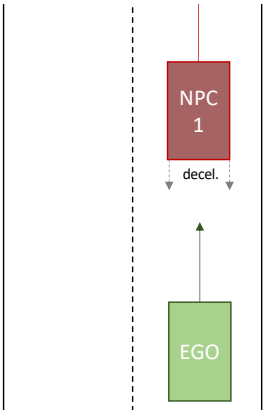
(a) ALKS R157 Test Scenario.



(b) Cut-In (Right) & Deceleration.



(c) Cut-In (Left) & Deceleration.



(d) Same Lane & Deceleration.

Figure 2 Representative scenarios for testing our approach for continuous monitoring, fault detection, and explanation.

Expected EGO Behavior: Initially, the *EGO* should not recognize *NPC1* as a target object since it is in an adjacent lane. Once *NPC1* cuts in front of the *EGO*, the *EGO* should detect *NPC1* as a target object and maintain a safe following distance to avoid a collision.

4. **Same Lane & Deceleration (Figure 2d)**

   Scenario Description: The *EGO* vehicle is traveling at a constant speed in its initial lane. Vehicle *NPC1* is driving at the same speed in the same lane, positioned ahead of the *EGO*. *NPC1* then begins to decelerate.

   Expected EGO Behavior: The *EGO* recognizes *NPC1* as a target object and follows it at a constant speed. Once *NPC1* starts decelerating, the *EGO* should decelerate accordingly to maintain a safe distance and avoid a collision.

In the subsequent chapters, we illustrate the results of the presented ALKS safety models and scenarios.

5.3 Results

In this section, we present the results of our experimental evaluation using Esmini as a simulation environment, the defined ALKS safety model as our systems under test, and the scenarios depicted in Section 5.2 as our use cases. We initially started our evaluation process by conducting 41 unit tests to ensure the correctness of our Continuous Validation Setup (CVS). This provided the foundation for performing an extensive experimental evaluation. To demonstrate the practical applicability of our methodology, based on 29 initial scenarios, we automatically generated 977 variations and integrated four different ALKS safety models, resulting in a total of 3,908 simulations executed. This process included the automatic post-processing of results and validation. All scenarios are based on the ASAM OpenSCENARIO and OpenDRIVE standards, ensuring realistic and reproducible test conditions. We evaluate the correct behavior of the system under test every 0.1 seconds. Considering all scenarios executed and investigated, Table 3 shows the number of violations that were identified for each safety model. It should be noted that each violation was counted, meaning that, for example, a continuous violation of one second consists of ten consecutive violations. With respect to *Overspeeding*, the number of violations is significantly lower compared to the other violations due to the fact that speed limits are only present in the *ALKS R157 Test Scenario*.

Table 3 Overview of all violations identified for each ALKS controller safety model.

Regulation Safety Model			
Overspeeding	No Stopping Maneuver	No Collision Avoidance	Insufficient Speed Adaption
78	10705	14082	222306
Reference Driver Safety Model			
Overspeeding	No Stopping Maneuver	No Collision Avoidance	Insufficient Speed Adaption
63	23374	10789	37211
RSS Safety Model			
Overspeeding	No Stopping Maneuver	No Collision Avoidance	Insufficient Speed Adaption
59	12904	5346	3804
FSM Safety Model			
Overspeeding	No Stopping Maneuver	No Collision Avoidance	Insufficient Speed Adaption
76	9889	16694	141892

From these scenarios, we highlight the results obtained for four representative use cases to illustrate the effectiveness of our approach. The scenarios involve various maneuvers and interactions between the *EGO* vehicle, equipped with an ALKS controller, and other road users (*NPC1*). The maneuvers and expected behaviors for each scenario are detailed in Section 5.2, providing insight into how the *EGO* vehicle should maintain safety and compliance with the regulations.

The following subsection will present a detailed description of the *ALKS R157 Test Scenario*, along with the observed faults and corresponding explanations. This comprehensive evaluation underscores the robustness of our proposed methodology for continuous monitoring, fault detection, and explanation using ASP in automated and autonomous driving systems. Within the Appendix, the results for the three other single maneuver scenarios are highlighted.

5.3.1 ALKS R157 Test Scenario (Figure 2a)

Within this section, we want to highlight the main results obtained for the scenario and included maneuvers depicted in Figure 2a. Figure 3, 4, 5, and 6 display the simulation output at the top as well as the output of our continuous validation setup at the bottom,

including the explanation of any misbehavior. As shown in the legend of the plots, the blue line represents the speed profile of the *EGO* vehicle, whereas the green line represents the vehicle speed of *NPC1*. Additionally, the dashed purple and magenta lines illustrate the relative distance to *NPC1* and the minimum following distance, respectively. Furthermore, it is important to note that the minimum following distance is dependent on the *EGO* vehicle speed. The current speed limit is depicted in red. At the bottom, we present the output of our CVS, where *FALSE* indicates that a fault was detected and *TRUE* shows that the ALKS controller complies with the regulations. Moreover, in case of a fault, the area below the curve is filled with different patterns to explain the cause of the fault. It is noteworthy to highlight the differences between *No Stopping*, *No Collision Avoidance*, and *Insufficient Speed Adaption*.

Insufficient Speed Adaption: This fault is identified if the *EGO* and *NPC1* are traveling with similar vehicle speeds, no abrupt decelerations occur, and the relative distance to the *NPC1* is less than the minimum following distance.

No Stopping: This occurs when the *EGO* is approaching a road user in the same lane who is traveling at low speeds or is performing a stopping maneuver, and the relative distance is greater than the minimum following distance. Hence, stopping in front of the road user would be possible without initiating a collision avoidance maneuver.

No Collision Avoidance: In contrast to *No Stopping*, this misbehavior is identified if the relative distance to a road user in the same lane is less than the minimum following distance, indicating that a collision would occur in case no avoidance maneuver is performed.

Additionally, since we receive no feedback from the integrated ALKS controllers on whether stopping or collision avoidance is initiated, we assume it to be false. Therefore, with more advanced controllers, some of the identified misbehaviors might be resolved when such additional information is available. In the following, we will elaborate in more detail on what faults occurred for the different safety models of the ALKS controller.

Figure 3 shows the results for the *ALKS R157 Test Scenario* and the ALKS controller with the integrated *Regulation* safety model. From the beginning of the simulation until around 2.5 seconds, we identify no faults since *NPC1* is still driving on the adjacent lane. However, after *NPC1* performs an overtaking maneuver followed by a cut-in, the *EGO* vehicle needs some time to re-establish a sufficient distance to the vehicle ahead. This delay results in a fault indicating *Insufficient Speed Adaptation* until the appropriate distance is regained. Furthermore, we see that between seconds 7 and 9, the controller fails to respect the present speed limit. This issue recurs during the time frame from 11 to 17 seconds. Our approach correctly detected both of these occurrences. Additionally, within seconds 11 and 20, we observe another instance of *Insufficient Speed Adaptation*, caused by the decelerating *NPC1*. At the end of the simulation, when *NPC1* comes to a stop, the *No Collision Avoidance* flag is raised because the relative distance to the road user is less than the minimum following distance, and no avoidance maneuver is initiated.

Within Figure 4, we showcase the results for the *Reference Driver* safety model. Similar to the previous results, the *EGO* vehicle's controller initially struggled to establish a safe following distance after *NPC1* performed the cut-in maneuver, which is also detected by our CVS. Additionally, unlike the *Regulation* model, the *Reference Driver* model successfully maintains the minimum following distance during the deceleration phase of *NPC1*. This indicates an improvement in adaptive behavior and responsiveness to changes in the relative speed and distance of *NPC1*. Despite these improvements, the *Reference Driver* model still

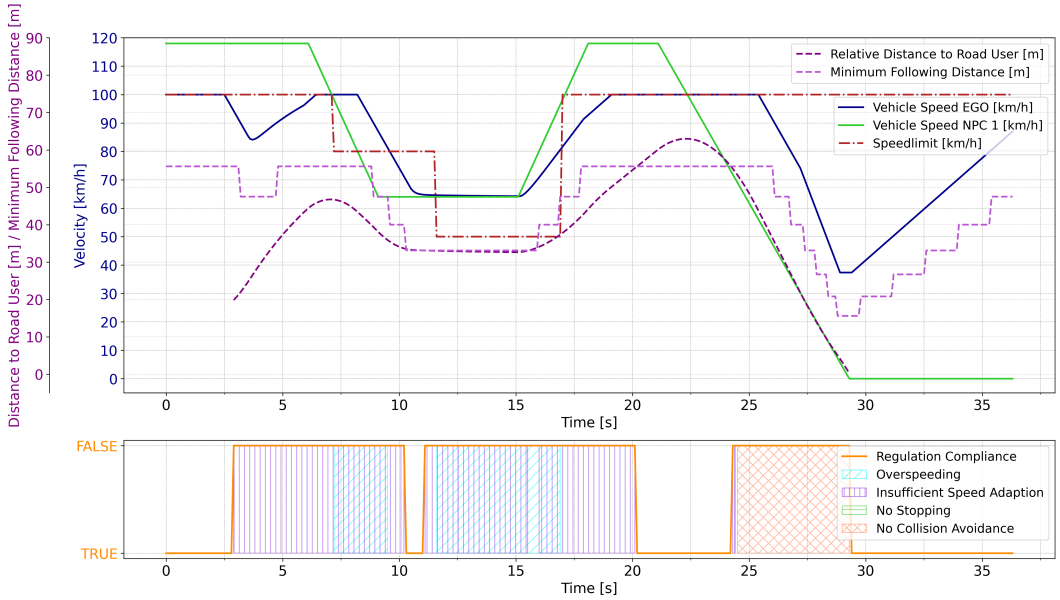


Figure 3 Validation results showcasing the output of our CVS approach for the ALKS R157 test scenario and the ALKS controller equipped with the *Regulation* safety model.

fails to comply with the speed limits present in the scenario. The *EGO* vehicle exceeds the speed limit at several points during the simulation, which is marked as overspeeding in the validation output.

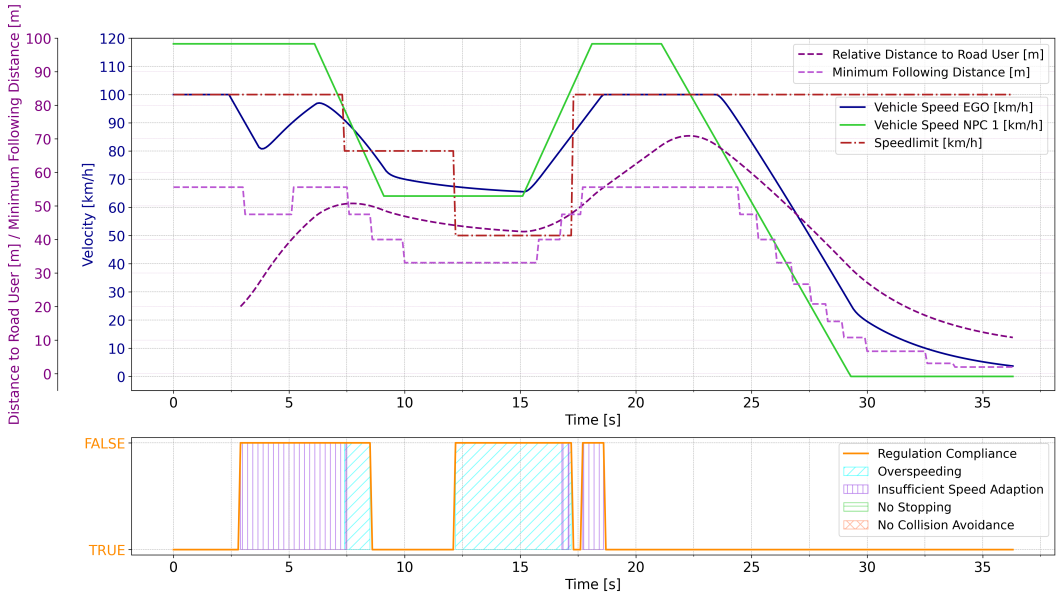


Figure 4 Validation results showcasing the output of our CVS approach for the ALKS R157 test scenario and the ALKS controller equipped with the *Reference Driver* safety model.

The outcome of our evaluation with the *RSS* safety model is depicted in Figure 5. As with the previous safety models, the ALKS controller fails to comply with the speed limits during certain time frames, i.e., between seconds 7 and 8 and from 12 to 17. Furthermore,

our CVS indicates a *Insufficient Speed Adaption* fault at the beginning of the simulation. Close to the end of the scenario, a *No Stopping* violation was identified. This stems from the fact that *NPC1* performs a stopping maneuver, and the *EGO* vehicle does not adapt to this reasonably. The *RSS* safety model demonstrates improvements over the previous safety models with respect to responsiveness in attempting to maintain a safe following distance, it still exhibits significant issues with speed limit compliance. However, as stated previously, those safety models are experimental implementations, and different outcomes are expected for more mature releases.

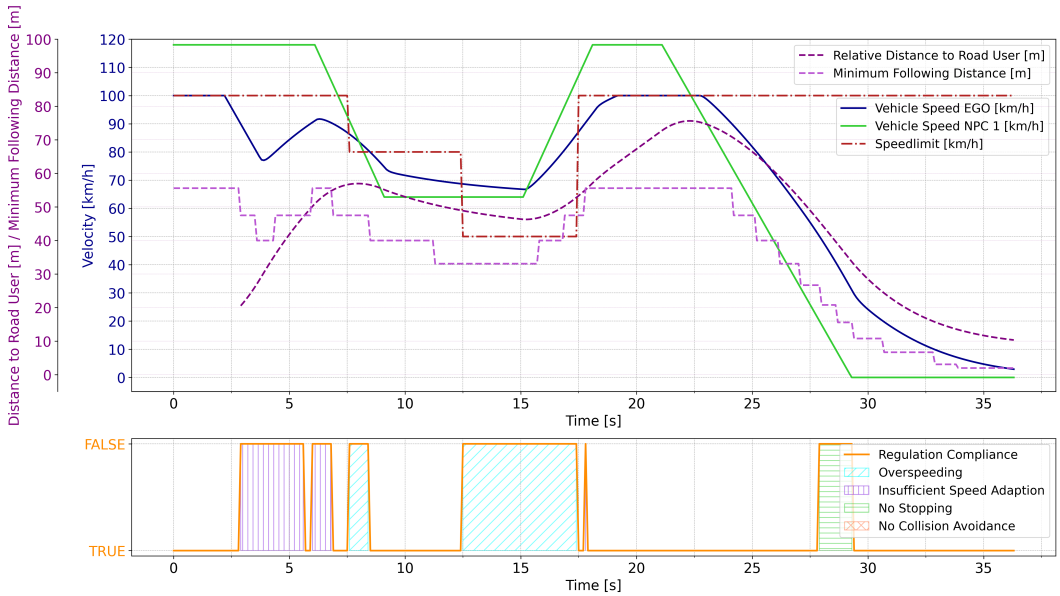


Figure 5 Validation results showcasing the output of our CVS approach for the ALKS R157 test scenario and the ALKS controller equipped with the *RSS* safety model.

The results depicted in Figure 6 illustrate the outcomes for the *FSM* safety model. There are noticeable similarities to the behavior of the *Regulation* safety model. We can observe that, in this simulation, the controller fails to adhere to the speed limits during specific periods. Additionally, it struggles to maintain a safe following distance to *NPC1* at several points, indicating that the vehicle does not consistently comply with the regulations. Furthermore, at the end of the simulation, the *No Collision Avoidance* fault is raised again, demonstrating that the *FSM* safety model reacts too late to the stopping maneuver performed by the *NPC1*.

The evaluation of our proposed methodology for continuous monitoring, fault detection, and explanation leveraging ASP was conducted through a series of unit tests and extensive experimental simulations using Esmini as a simulation environment. Initially, 41 unit tests were performed to ensure the correctness of our Continuous Validation Setup (CVS), all of which were successfully passed, providing a robust foundation for further testing.

Subsequently, 3,908 scenarios were executed, derived from 29 initial scenarios, incorporating four distinct ALKS safety models: *Regulation*, *Reference Driver*, *RSS*, and *FSM*. Our methodology was applied to each model to assess their compliance with UN R157 regulations and their overall performance under simulated driving conditions. Overall, our results demonstrate the successful identification of misbehaviors during various simulation periods, as depicted by the simulation data illustrated in the respective figures.

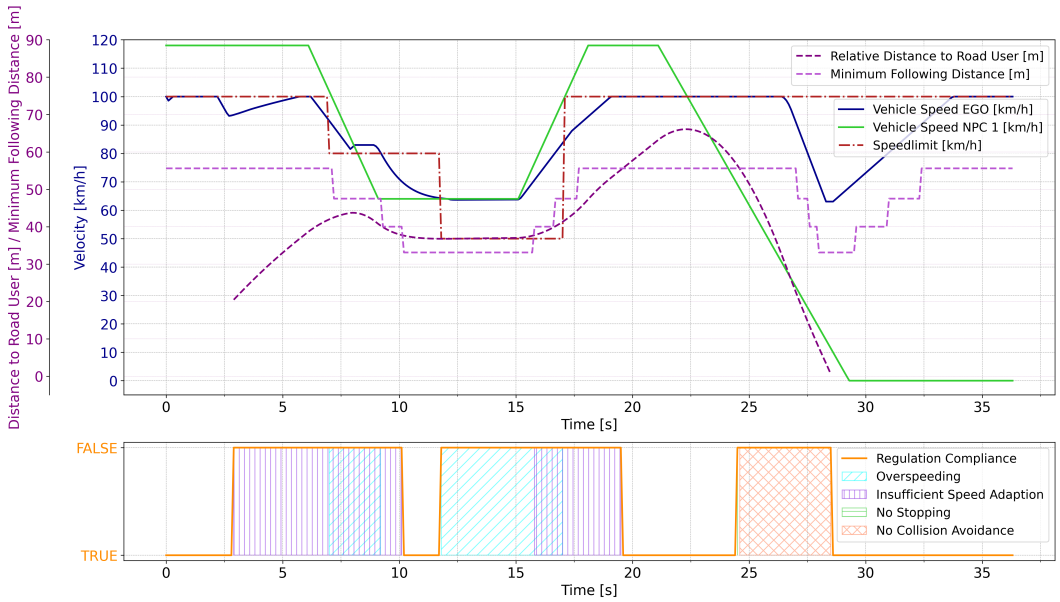


Figure 6 Validation results showcasing the output of our CVS approach for the ALKS R157 test scenario and the ALKS controller equipped with the *FSM* safety model.

6 Conclusion

In this paper, we proposed a novel methodology for continuous monitoring, fault detection, and explanation of automated and autonomous driving systems using Answer Set Programming (ASP). Our approach integrates ASP with data from simulation environments to identify discrepancies between expected and actual vehicle behavior, providing explanations for detected faults. Through a series of 41 unit tests, we validated the correctness of our Continuous Validation Setup (CVS). Furthermore, we conducted an extensive experimental evaluation within a simulation framework using Esmini as an environment simulator, encompassing 3,908 scenarios derived from 29 initial scenarios. These scenarios, in combination with four distinct ALKS safety models, were used to test our methodology's effectiveness in identifying faults during runtime. Our results demonstrated the effectiveness of our approach in identifying misbehaviors and compliance issues with UN Regulation No. 157 under various simulated driving conditions. The successful detection and explanation of faults, which can further serve as input to fault-adaptive control strategies and mitigation measures, highlights the potential of our methodology to enhance the reliability and safety of automated and autonomous driving systems. Future work will focus on extending our approach to more complex scenarios and integrating additional regulations and general driving knowledge. Additionally, challenges for real-world deployment, such as handling noisy signals and ensuring real-time performance within vehicles, will be addressed.

References

1 William F. Clocksin and Christopher S. Mellish. *Programming in Prolog*. Springer Berlin Heidelberg, 2003. doi:10.1007/978-3-642-55481-0.

2 Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. *Answer Set Programming: A Primer*, pages 40–110. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi:10.1007/978-3-642-03754-2_2.

3 United Nations Economic Commission for Europe. Un regulation no 157 – uniform provisions concerning the approval of vehicles with regards to automated lane keeping systems [2021/389], March 2021. URL: http://data.europa.eu/eli/reg/2021/389/oj.

4 Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming*, 19(1):27–82, 2019. doi:10.1017/S1471068418000054.

5 Leilani Hendrina Gilpin. *Anomaly Detection Through Explanations*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2020.

6 Ledio Jahaj, Lorenz Klampfl, and Franz Wotawa. Knowledge-based monitoring for checking law and regulation compliance. In Hamido Fujita, Richard Cimler, Andres Hernandez-Matamoros, and Moonis Ali, editors, *Advances and Trends in Artificial Intelligence. Theory and Applications*, pages 491–502, Singapore, 2024. Springer Nature Singapore. doi:10.1007/978-981-97-4677-4_40.

7 Nidhi Kalra and Susan 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*, 94:182–193, 2016. doi:10.1016/j.tra.2016.09.010.

8 Lorenz Klampfl, Florian Klück, and Franz Wotawa. Using genetic algorithms for automating automated lane-keeping system testing. *Journal of Software: Evolution and Process*, 36(3):e2520, 2022. doi:10.1002/smr.2520.

9 Lorenz Klampfl and Franz Wotawa. Identifying critical scenarios in autonomous driving during operation. In *Artificial Intelligence. ECAI 2023 International Workshops*, volume 1947 of *Communications in Computer and Information Science*, pages 156–172, Cham, 2024. Springer. Trustworthy AI for Safe & Secure Traffic Control in Connected & Autonomous Vehicles: TACTFUL 2023. doi:10.1007/978-3-031-50396-2_9.

10 Florian Klück, Martin Zimmermann, Franz Wotawa, and Mihai Nica. Genetic algorithm-based test parameter optimization for adas system testing. In *Proceedings of the 19th IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 418–425, 2019. doi:10.1109/QRS.2019.00058.

11 Philip Koopman and Michael Wagner. Challenges in autonomous vehicle testing and validation. *SAE Int. J. Trans. Safety*, 4:15–24, April 2016. doi:10.4271/2016-01-0128.

12 Konstantinos Mattas, Michail Makridis, George Botzoris, Akos Kriston, Fabrizio Minarini, Basil Papadopoulos, Fabrizio Re, Greger Rognelund, and Biagio Ciuffo. Fuzzy surrogate safety metrics for real-time assessment of rear-end collision risk. a study based on empirical observations. *Accident Analysis & Prevention*, 148:105794, 2020. doi:10.1016/j.aap.2020.105794.

13 Fabian Schuldt, Andreas Reschka, and Markus Maurer. *A Method for an Efficient, Systematic Test Case Generation for Advanced Driver Assistance Systems in Virtual Environments*, pages 147–175. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-61607-0_7.

14 Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. On a formal model of safe and scalable self-driving cars, 2018. arXiv:1708.06374.

15 Jakob Suchan, Mehul Bhatt, and Srikrishna Varadarajan. Out of sight but not out of mind: An answer set programming based online abduction framework for visual sensemaking in autonomous driving. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 1879–1885. International Joint Conferences on Artificial Intelligence Organization, July 2019. doi:10.24963/ijcai.2019/260.

16 Jakob Suchan, Mehul Bhatt, and Srikrishna Varadarajan. Commonsense visual sensemaking for autonomous driving – on generalised neurosymbolic online abduction integrating vision and semantics. *Artificial Intelligence*, 299:103522, 2021. doi:10.1016/j.artint.2021.103522.

17 Franz Wotawa. *Testing Autonomous and Highly Configurable Systems: Challenges and Feasible Solutions*, pages 519–532. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-31895-0_22.

18 Franz Wotawa. On the use of answer set programming for model-based diagnosis. In Hamido Fujita, Philippe Fournier-Viger, Moonis Ali, and Jun Sasaki, editors, *Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices*, pages 518–529, Cham, 2020. Springer International Publishing. doi:10.1007/978-3-030-55789-8_45.

19 Franz Wotawa and David Kaufmann. Model-based reasoning using answer set programming. *Applied Intelligence*, 52(15):16993–17011, 2022. doi:10.1007/s10489-022-03272-2.

20 Franz Wotawa, Bernhard Peischl, Florian Klück, and Mihai Nica. Quality assurance methodologies for automated driving. *Elektrotechnik & Informationstechnik*, 135(4–5), 2018. doi:10.1007/s00502-018-0630-7.

7 Appendix

In this section Table 4-6 showcase the results obtained for the scenarios depicted in Figure 2b, 2c, and 2d.

Table 4 Results obtained for the *Cut-In (Right)* & *Deceleration* scenario.

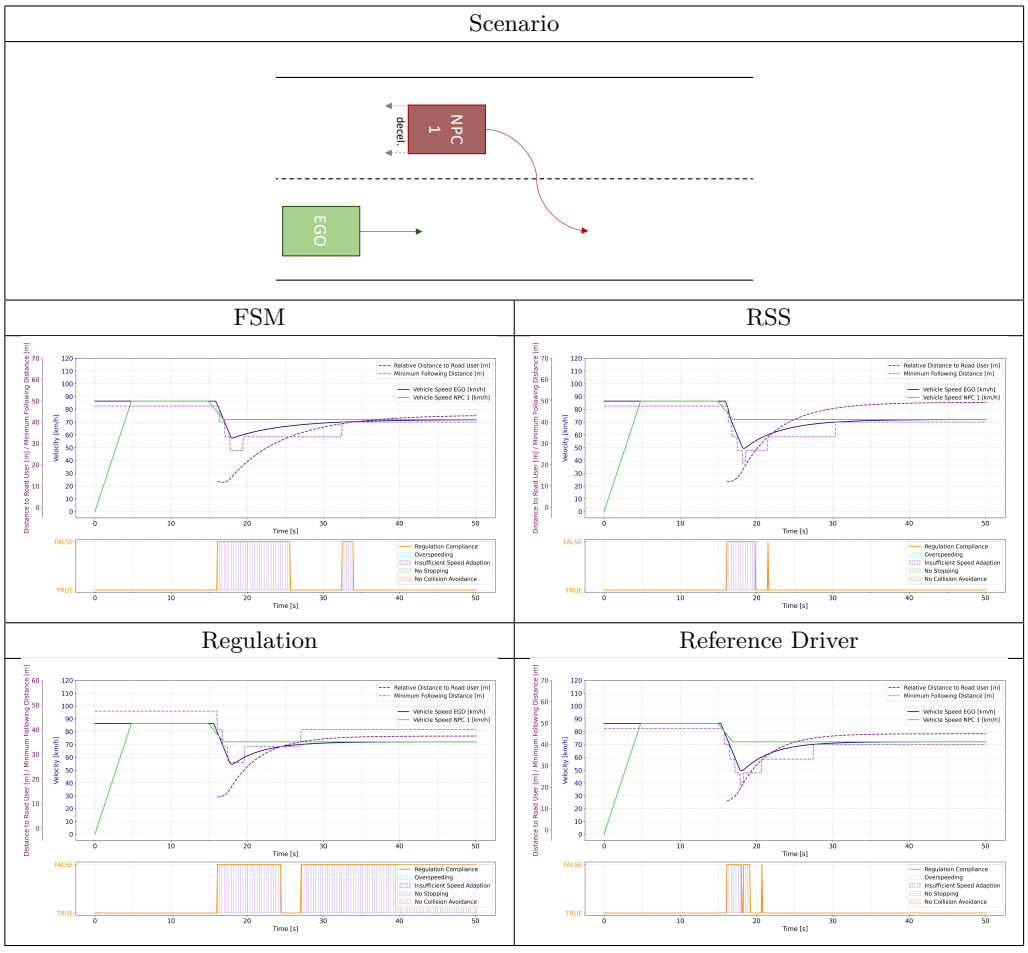


Table 5 Results obtained for the *Same Lane & Deceleration* scenario.

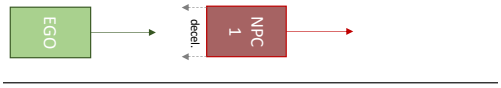
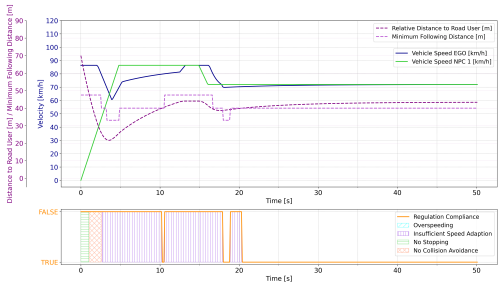
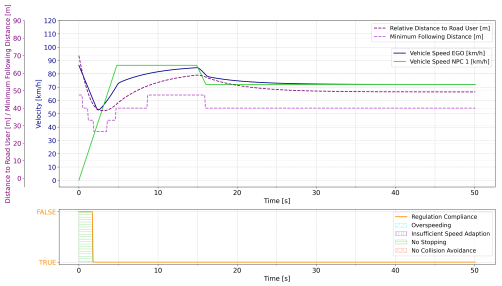
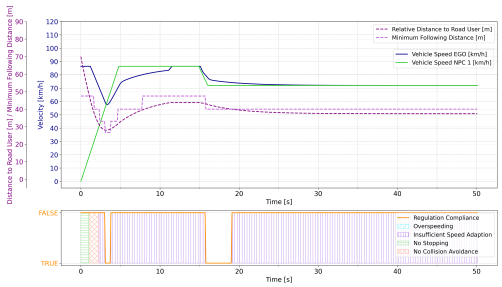
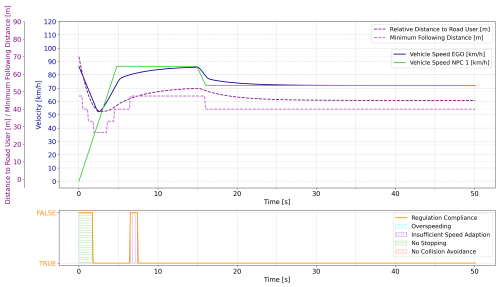
Scenario	
	
FSM	RSS
	
Regulation	Reference Driver
	

Table 6 Results obtained for the *Cut-In (Left)* & *Deceleration* scenario.