electronics

MDPI

Article

TraModeAVTest: Modeling Scenario and Violation Testing for Autonomous Driving Systems Based on Traffic Regulations

Chunyan Xia [1,2], Song Huang [1,*], Changyou Zheng [1,*], Zhen Yang [1], Tongtong Bai [1] and Lele Sun [1]

[1] College of Command and Control Engineering, Army Engineering University of PLA, Nanjing 210007, China; xiachunyan@mdjnu.edu.cn (C.X.); yangzhen@aeu.edu.cn (Z.Y.); baitongtong@aeu.edu.cn (T.B.); sunlele@aeu.edu.cn (L.S.)

[2] College of Computer and Information Technology, Mudanjiang Normal University, Mudanjiang 157011, China

* Correspondence: huangsong@aeu.edu.cn (S.H.); zhengchy@aeu.edu.cn (C.Z.)

Abstract: Current testing methods for autonomous driving systems primarily focus on simple traffic scenarios, generating test cases based on traffic accidents, while research on generating edge test cases for complex driving environments by traffic regulations is not adequately comprehensive. Therefore, we propose a method for scenario modeling and violation testing using an autonomous driving system based on traffic regulations named TraModeAVTest. Initially, TraModeAVTest constructs a Petri net model for complex scenarios based on the combination relationships of basic traffic regulation scenarios and verifies the consistency of the model's design with traffic regulation requirements using formal methods, to provide a representation of traffic regulation scenario models for the violation testing of autonomous driving systems. Subsequently, based on the coverage criteria of the Petri net model, it utilizes a search strategy to generate model paths that represent traffic regulations, and employs a parameter combination method to generate test cases that cover the model paths, to test the violation behaviors of autonomous driving systems. Finally, simulation experiment results on the Baidu Apollo demonstrate that the test cases representing traffic regulations generated by TraModeAVTest can effectively identify the behaviors of autonomous vehicles violating traffic regulations, and TraModeAVTest can effectively improve the efficiency of generating different types of violation scenarios.

Keywords: software testing; autonomous driving system; test case; Petri net model; traffic regulation

Citation: Xia, C.; Huang, S.; Zheng, C.; Yang, Z.; Bai, T.; Sun, L. TraModeAVTest: Modeling Scenario and Violation Testing for Autonomous Driving Systems Based on Traffic Regulations. *Electronics* **2024**, *13*, 1197. https://doi.org/10.3390/electronics13071197

Academic Editors: Sanghyuk Lee, Soo Kyun Kim, Scott Uk-Jin Lee, Seokhun Kim and Asad Abbas

Received: 3 January 2024
Revised: 30 January 2024
Accepted: 5 February 2024
Published: 25 March 2024

1. Introduction

With the rapid development of artificial intelligence, autonomous vehicles (AVs) have become a typical application of AI technology and hhave made significant progress in the past decade. As a type of non-deterministic system, the safety of autonomous driving systems (ADSs) has garnered increasing attention, and the prevention of traffic accidents is one of the crucial safety requirements for AVs [1]. In 2018, an AV operated by Uber was involved in a traffic accident on public roads, which resulted in the death of a pedestrian due to the failure to yield [2]. Therefore, as a safety-critical system, ADSs must undergo regulatory testing before deployment on actual roads to ensure their safety. Testing based on deep learning for ADSs is particularly challenging due to the lack of interpretability in the behavior of deep learning systems [3].

By employing effective scenario generation methods in combination with simulation testing techniques, the diverse and complex traffic scenarios can be provided for testing ADSs [4]. Simulation testing uses modeling methods and high-fidelity simulators, such as LGSVL [5] and CALAR [6], to map the real world into a virtual simulation environment, enabling the generation of intricate interactive environments and hazardous boundary scenarios [7]. In the field of autonomous driving, a scenario is understood as a comprehensive reflection of the driving environment and behavior within a specific time and space, and can

be categorized into functional, logical, and specific scenarios based on different abstraction levels [8]. Test cases are defined as specific scenarios containing "pass/fail" criteria to evaluate the performance of AVs, such as collisions between AVs and obstacles [9]. Due to the complexity of driving environments, including factors such as weather conditions, road layouts, and dynamic traffic information, these complex situations can have unpredictable effects on AVs. Therefore, researching testing techniques for autonomous driving in complex environmental scenarios is of significant importance for enhancing traffic safety and reducing the likelihood of traffic accidents involving ADSs.

Traffic regulations are utilized as external resources for testing autonomous driving, offering both scenario descriptions and driving rules concurrently [10]. As indicated by a recent study on autonomous driving testing [11], the development of test scenarios based on traffic regulations in a simulated environment is an urgent issue that requires attention. Therefore, we concentrate on the construction and reorganization of autonomous driving scenarios based on traffic regulations, as well as the generation of test cases that represent traffic regulations to identify violations within AVs.

In summary, this paper makes the following contributions:

1. We proposed a formal definition and combination relationships of autonomous driving scenarios and constructed a Petri net model for the complex traffic regulation scenarios based on the combination of basic traffic regulation scenarios. The consistency of the model design with traffic regulation requirements has been validated using formal methods;
2. We introduced coverage criteria suitable for Petri nets and utilized a search strategy to generate scenario model paths that represent traffic regulations. Subsequently, we have iteratively generated test cases for covering the model paths using a parameter combination approach;
3. TraModeAVTest was evaluated using simulation experiments on the Baidu Apollo platform. The experimental results demonstrate that the test cases generated by TraModeAVTest, covering traffic regulation scenario model paths, effectively represent traffic regulations and can efficiently test violations of traffic regulations within the ADS. Furthermore, the experimental comparisons with baseline methods indicate that TraModeAVTest effectively improves the efficiency of generating different types of violation scenarios.

2. Related Work

With the rapid development of autonomous driving technology, the testing of ADSs has become a research focus in both academia and industry. Currently, many companies are dedicated to this field, such as Tesla Autopilot [12], Baidu Apollo [13], and Autoware [14]. The majority of existing research has been focused on the collision avoidance functionality of autonomous driving systems, testing them by introducing various safety metrics such as accident rates [15], drivable area [16,17], collision time [18,19], and performance boundaries [20,21]. However, due to open and uncertain traffic environments, testing autonomous driving systems still remains challenging.

The currently prevalent methods for autonomous driving test scenario generation include data-driven scenario generation methods and model-driven scenario generation methods. In data-driven methods, scenarios are generated from existing datasets, such as collision reports [22,23] and real-world driving records [24–27], and deep learning techniques can also be used to enhance the diversity of these datasets [28–30]. Drawing on real traffic data, scenario reconstruction is driven by the extraction of necessary scenario data to replicate offline scenarios in virtual environments for testing with ADSs [31]. For example, Gambi et al. [32] introduced a method for generating crucial scenarios based on police reports and accident sketches. Since data-driven methods rely on traffic accident data to recreate pre-accident scenarios for testing ADSs, the testing scenarios are solely based on actual traffic accidents, posing challenges in generating rare boundary test cases and inadequately testing ADSs.

In model-driven methods, scenarios are generated based on the evaluation results of system models [33–35]. In particular, simulation-based test case generation methods, using high-fidelity simulators to generate test cases for detecting safety violations in ADSs, have attracted increasing attention due to their high testing efficiency, as well as good scalability and repeatability [36–38]. For example, Tuncali et al. [39] transformed the problem of finding fault-inducing scenarios into a function minimization problem, where a robustness function is defined in a given virtual environment to minimize the input parameters that can minimize the robustness function and find the initial settings for the specified scenarios. Furthermore, search-based methods generate test cases guided by the results of simulation experiments, and have been widely applied as an important approach to model-driven scenario generation [40–43]. Scenario generation based on search techniques involves modeling the spatial behavior of AVs, parameterizing scenario parameters within the spatial scope, and generating specific scenarios using a parameter value search [44]. For example, Haq et al. [45] proposed advanced techniques for testing ADSs, which expanded existing search algorithms and utilized surrogate models to effectively identify safety-related violations. However, existing approaches solely focus on safety-related guidance in generating critical scenarios, neglecting to fully consider the constraints imposed by traffic regulations. This oversight may result in generated test cases potentially disregarding compliance with traffic regulations.

Studies have shown that [46,47], in order to guarantee the safety and comfort of ADSs, AVs are required to adhere to a variety of regulatory standards. Presently, the testing of ADSs, based on scenarios, places a primary emphasis on safety [48], with adherence to traffic regulations being one of the essential prerequisites that ADSs must fulfill [49]. It is of great significance for the safety testing of ADSs to be able to generate test cases that represent traffic regulations and subsequently test whether the ADSs violates traffic regulations.

Additionally, the Petri net theory has also been widely employed in autonomous driving testing. Tang et al. [50] proposed a method for testing AV route coverage through map modeling, which models the map as a Petri net and then defines the topological and route characteristics of intersections based on the Petri net model, and the experiments demonstrated that the method generated test cases achieving higher route coverage. Building on this inspiration, in [51], the authors proposed a modeling and verification method for basic ADS testing scenarios. This method, starting from traffic regulations, integrates the interactive behaviors of complex driving environments and constructs a Petri net model of intersection scenarios. Experiments have shown that the constructed scenario model can represent traffic regulations. Based on this, this paper focuses on the definition and combination relationships of basic scenarios, in order to achieve the modeling and verification of complex scenarios based on traffic regulations. Subsequently, this facilitates the generation of test cases covering model paths to discover more violations of traffic regulations by ADS.

3. Background

3.1. Petri Net

The Petri net is commonly used as a mathematical representation of discrete parallel systems and is an important tool for modeling analysis, verification, simulation, and property analysis of complex systems [52].

The mathematical depiction of a Petri net involves representing it as a triplet, where S denotes places, T denotes transitions, and F denotes flow relations, all subject to the following conditions:

1. $S \cup T \neq \Phi$
2. $S \cap T = \Phi$
3. $F \subseteq (S \times T) \cup (T \times S)$
4. $dom(F) \cup cod(F) = S \cup T$, where $dom(F) = \{x \in S \cup T | \exists y \in S \cup T : (x, y) \in F\}$, $cod(F) = \{x \in S \cup T | \exists y \in S \cup T : (y, x) \in F\}$.

The basic elements of a Petri net include places, transitions, and arcs. Places represent the states that may occur in a simulated system, while transitions describe the actions that lead to the transition between two states. Arcs are used to connect places and transitions, representing the transition between states and actions. The black dots inside circular nodes represent tokens in places, and the transition of tokens within the system represents the flow of resources.

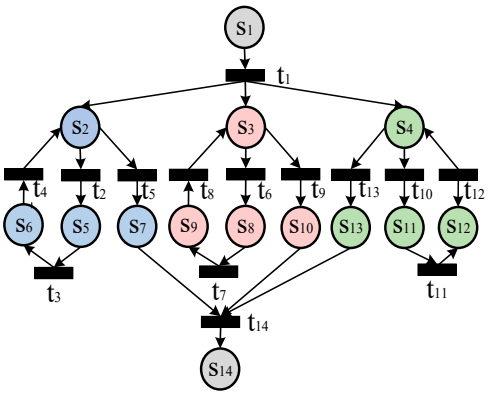
3.2. Model Path

In reference [51], the authors have presented a comprehensive overview of the modeling and verification methods for intersection scenarios based on traffic regulations. Specifically, commencing from traffic regulations and accounting for the complex driving environment, a Petri net is utilized to construct a model of an autonomous driving test scenario that conforms to traffic regulations. The rational states, transitions, and actions within the model pathway serve to represent the traffic regulations that an ADSs must adhere to. Formal methods are subsequently utilized to verify the consistency between the model pathway and the requirements of traffic regulations. Consequently, the scenario model constructed in accordance with traffic regulations can effectively represent the corresponding traffic regulations.

In the Petri net model, each test path is represented as a complete path from the initial state to the terminal state of the Petri net model. Newly generated test paths should contain at least one place or transition that has not appeared in other paths. Since each path of the scenario model corresponds to a traffic regulation, generating test cases that cover all paths of the model can effectively represent the validation of each traffic regulation, thereby enabling the testing of AVs' compliance with traffic regulations.

For instance, the Petri net model of the signal-free intersection scenario shown in Figure 1 represents the traffic regulations as follows:

1. The path P_1 represents the traffic regulation "vehicles on the left yield to vehicles on the right at the intersection", i.e., $P_1 : s_2 \rightarrow t_2 ® s_5 ® t_3 ® s_6 ® t_4 ® s_2 ® t_5 ® s_7$.
2. The path P_2 represents the traffic regulation "turning vehicles yield to vehicles going straight at the intersection", i.e., $P_2 : s_3 ® t_6 ® s_8 ® t_7 ® s_9 ® t_8 ® s_3 ® t_9 ® s_{10}$.
3. The path P_3 represents the traffic regulation "vehicles yield to pedestrians at the intersection", i.e., $P_3 : s_4 ® t_{10} ® s_{11} ® t_{11} ® s_{12}$ $® t_{12} ® s_4 ® t_{13} ® s_{13}$.



S1: Deceleration State
S2: Detection of vehicles on the right State
S3: Detection of vehicles proceeding straight State
S4: Detection of pedestrians State
S5: Presence of vehicles on the right State
S6: Giving way to vehicles on the right State
S7: Proceeding through the intersection State
S8: Presence of vehicles proceeding straight State
S9: Giving way to vehicles proceeding straight State
S10: Proceeding through the intersection State
S11: Presence of pedestrians State
S12: Giving way to pedestrians State
S13: Proceeding through the intersection State
S14: Exiting the intersection State

t1: Entering the intersection Transition
t2: Identifying vehicles on the right Transition
t3: Responding to vehicles on the right Transition
t4: Adaptive adjustment of vehicle speed Transition
t5: Perceiving no vehicles on the right Transition
t6: Identifying vehicles proceeding straight Transition
t7: Responding to vehicles proceeding straight Transition
t8: Adaptive adjustment of vehicle speed Transition
t9: Perceiving no vehicles proceeding straight Transition
t10: Identifying pedestrians Transition
t11: Responding to pedestrians Transition
t12: Adaptive adjustment of vehicle speed Transition
t13: Perceiving no pedestrians Transition
t14: Exiting the intersection Transition

Figure 1. Petri net model of the left turn scenario.

Therefore, from the perspective of generating test cases that cover structurally diverse paths, the capability to generate test cases covering model paths can effectively represent the corresponding traffic regulations. This provides a decision model for testing the compliance of ADSs in complex traffic environments.

3.3. Method Overview

We have proposed a method, named TraModeAVTest, for modeling scenarios and conducting violation tests for ADSs based on traffic regulations, to address the compliance verification issue of ADSs. Specifically, TraModeAVTest comprises two main components. Firstly, it involves the modeling and validation of traffic regulation scenarios based on Petri nets, which primarily focuses on the critical task of constructing a complex Petri net model for the interaction environment of an ADS based on traffic regulations, enabling the Petri net model paths to represent traffic regulations. Secondly, it encompasses the simulation testing of the AV violation behaviors, which primarily addresses the key issue of generating test cases covering model paths based on traffic regulation scenario models to identify AV behaviors that violate traffic regulations.

For the first challenge of scenario modeling and validation, we start the process by modeling the basic test scenarios of the ADS based on traffic regulations. Subsequently, we define the combination relationships of autonomous driving scenarios, thereby modeling the Petri net model of complex scenarios and verifying the consistency of the scenario model design and system traffic regulation requirements using formal methods, thereby achieving the validation of model paths representing traffic regulations.

For the second challenge of violation testing, we first utilize search strategies based on coverage standards suitable for Petri net models to generate model paths representing traffic regulations. Then, we use a parameter combination method to generate test cases covering model paths. Finally, through simulation experiments on the Baidu platform Apollo, we verify that the test cases generated by TraModeAVTest covering the scenario model paths effectively identify the behaviors of AVs violating traffic regulations, thus validating the testing scenarios based on traffic regulation modeling and the test cases generated based on models, effectively testing the compliance of the ADS. In fact, these violation behaviors are also important causes of traffic accidents involving AVs. The method framework of TraModeAVTest is illustrated in Figure 2.

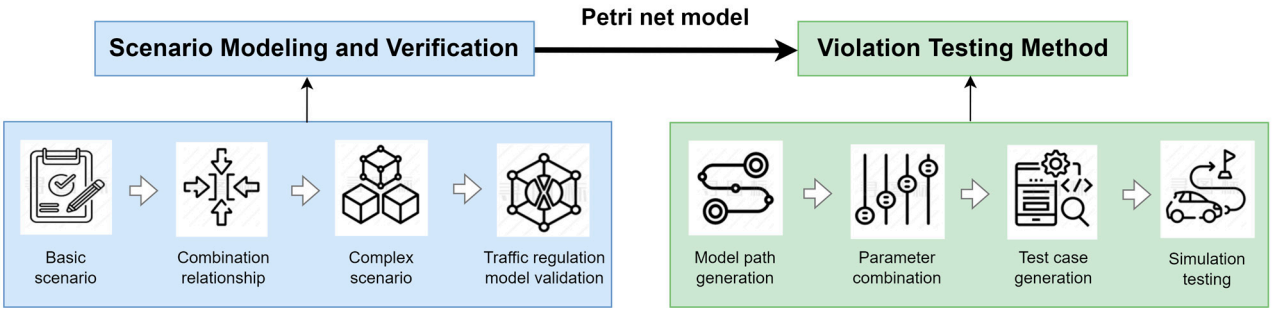


Figure 2. Method framework diagram.

4. Scenario Modeling and Validation

In our previous study [51], we presented a comprehensive overview of the modeling and validation methods for the intersection scenarios based on traffic regulations. Using similar methods, we can construct more basic scenarios based on traffic regulation, including cruising, lane changing, following, overtaking, and parking. In this section, we describe the combination relationships of basic traffic regulation scenarios, and then elaborate on the process of modeling and verifying complex traffic regulation scenarios.

4.1. Scenario Combination Relationship

According to the fundamental behaviors of vehicle driving, we categorized driving scenarios into basic scenarios such as cruising, lane changing, overtaking, intersection, following, and parking. These basic scenarios can be further classified into meta-scenarios. For example, the intersection scenario can be divided into two scenarios: crossroads and roundabouts. The crossroads scenario can be further subdivided into four meta-scenarios: left turn, straight, right turn, and U-turn. Similarly, the roundabout scenario can also

be further subdivided into entering the roundabout and exiting the roundabout. The cruising scenario includes static obstacles ahead, dynamic obstacles ahead, and normal cruising. The following scenario involves vehicles merging in, vehicles merging out, and emergency situations ahead. The lane changing scenario comprises vehicles changing left and vehicles changing right. The overtaking scenario encompasses following the target vehicle, merging into the adjacent lane, overtaking the target vehicle, and returning to the original lane. The parking scenario categorizes emergency vehicle parking and right-side vehicle parking. Furthermore, instantiating these meta-scenarios through parameters yields specific scenarios for testing the violation behaviors of ADS. Specifically, the relationship between basic scenarios and meta-scenarios is shown in Figure 3.

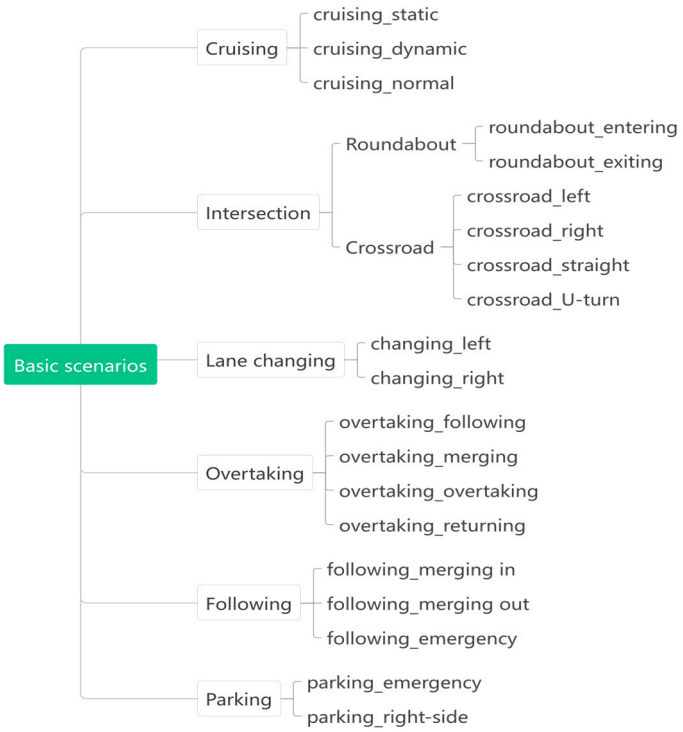


Figure 3. Relationship between basic scenarios and meta-scenarios.

The traffic regulations, upon which the construction of the scenario model is predicated, are articulated in natural language. In order to bridge the gap between natural language and formal language, we present a formal definition of the scenario. The formalized description of a scenario is denoted as a triplet $<\pi, \prec, s>$, where π represents a set of scenarios, s is a scenario within π (i.e., $s \in \pi$), and $\prec$ denotes a precedence relationship, which is a strict ordering relation on π. For example, $s_1 \prec s_2$ denotes that s_1 precedes s_2, signifying that s_1 takes precedence over s_2 in occurrence. As the occurrence of any scenario can be viewed as the occurrence of an event, the relationships between scenarios can be described using the relationships between events. The elements of the scenario still serve to describe the scenario. Therefore, based on the formal definition of the scenario, three common relationships between scenarios can be described, as shown in Table 1.

Table 1. Description of the scenarios relationship.

Definition	Meaning	Example
$s_1 \prec s_2$	The occurrence of s_1 strictly precedes the occurrence of s_2.	For s_1 = "Following" and s_2 = "Overtaking", then $s_1 \prec s_2$.
$s = s_1 \cup s_2$	If either s_1 or s_2 occurs, then s occurs.	For s = "Lane changing", s_1 = "changing_left", s_2 = "changing_right", then $s = s_1 \cup s_2$.
$s = s_1 \cap s_2$	If both s_1 and s_2 occur, then s occurs.	For s = "Roundabout", s_1 = "roundabout_entering", s_2 = "roundabout_exiting", then $s = s_1 \cap s_2$.

Based on the formal definition of the scenarios mentioned above and the description of the three common relationships between scenarios in Table 1, the relationships between the basic scenarios and meta-scenarios in Table 1 are described as follows:

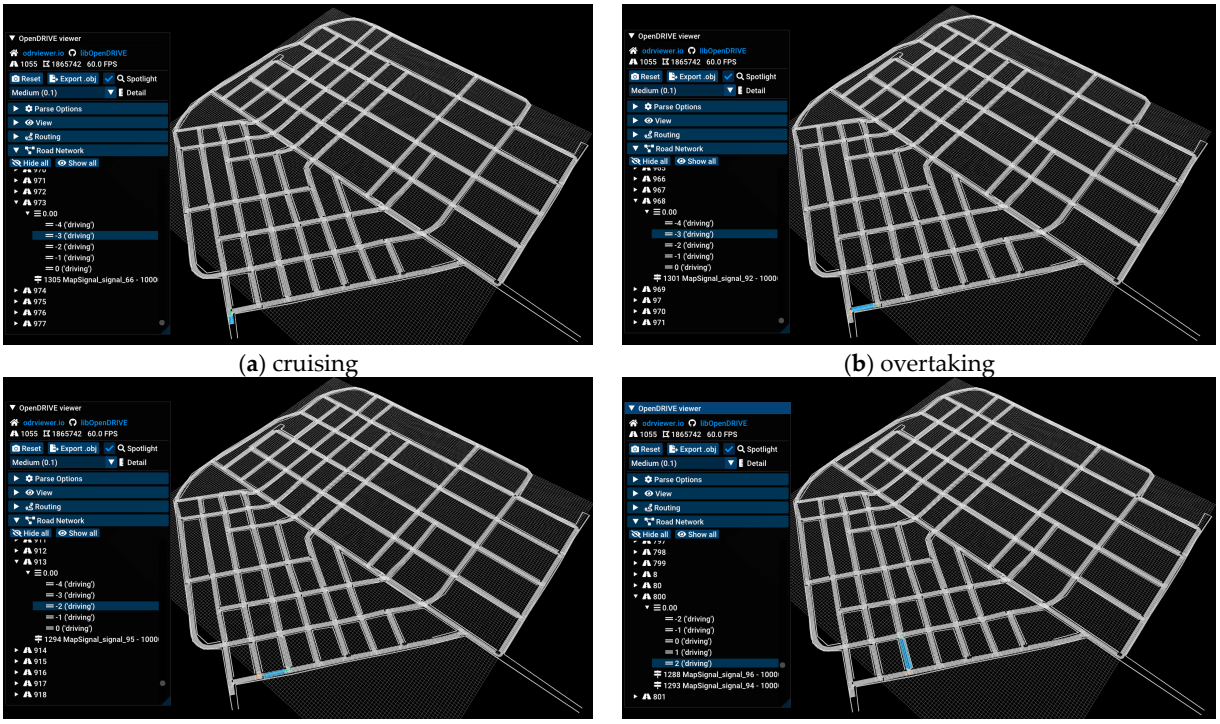
Relation 1: Cruising = cruising_static ∪ cruising_dynamic ∪ cruising_normal.
Relation 2: Lane changing = changing_left ∪ changing_right.
Relation 3: Following = following_merging in ∪ following_merging out ∪ following_emergency.
Relation 4: Intersection = Roundabout ∪ Crossroad.
Relation 4.1: Roundabout = roundabout_entering ∩ roundabout_exiting.
Relation 4.2: Crossroad = crossroad_left ∪ crossroad_straight ∪ crossroad_right ∪ crossroad_U-turn.
Relation 5: Overtaking = overtaking_following ∩ overtaking_merging ∩ overtaking_overtaking ∩ overtaking_returning.
Relation 6: Parking = parking_emergency ∪ parking_right-side.

In this instance, the intersection described by Relation 4 may represent either Crossroad or Roundabout. Intersection can only be triggered when one of these options is activated. Additionally, Roundabout outlined in Relation 4.1 comprises two meta-scenarios: roundabout_entering and roundabout_exiting. Activation of Roundabout is contingent upon the simultaneous triggering of both meta-scenarios. Crossroad detailed in Relation 4.2 encompasses four meta-scenarios: crossroad_left, crossroad_straight, crossroad_right and crossroad_U-turn. Activation of any of these meta-scenarios can trigger Crossroad. It is apparent that when constructing complex scenarios based on inter-scenario relationships, the "or" relationship dictates that only one meta-scenario can occur, while the "and" relationship stipulates that each meta-scenario must occur. Moreover, strict adherence to the "priority" relationship is imperative for all scenarios.

4.2. Complex Scenario Modeling

In order to model complex scenarios based on traffic regulations and verify the consistency of model design with system requirements, our research focuses on methods for integrating basic scenarios into complex scenarios. In practice, basic scenarios may involve only one pedestrian or one obstacle and are relatively straightforward. However, complex scenarios are comprised of multiple basic scenarios, taking into account the constraints of multiple traffic regulations and incorporating the interaction between AVs and obstacles or pedestrians, thereby amplifying the complexity of the scenario interaction environment.

To model basic scenarios based on traffic regulations, we select a section of the map to construct complex scenarios for the ADS, encompassing cruising, intersection, lane changing, following, overtaking, and parking. We employ the trajectory route of the AV depicted in Figure 4 as an example to elucidate the modeling and verification process of the complex scenarios.



(**a**) cruising (**b**) overtaking

(**c**) changing_left (**d**) following

Figure 4. Road trajectory diagram of the AV.

The scenario combination relationship of this road segment is as follows:

cruising $\prec$ crossroad_right $\prec$ overtaking $\prec$ crossroad_straight $\prec$ changing_left $\prec$ crossroad_left $\prec$ following $\prec$ parking_right-side

Here, overtaking = overtaking_following $\cap$ overtaking_merging $\cap$ overtaking_overtaking $\cap$ overtaking_returning.

Within the basic scenario model, a mechanism exists for receiving commands, which is equivalent to other scenario models calling the interface of this scenario model. It is evident that commands can be sent by invoking transitions, thereby combining basic scenario models into complex scenario models. Based on the road trajectory of the AV as shown in Figure 4 and the scenario combination relationships outlined in Section 4.1, a Petri net model for the complex scenarios is constructed for clarity. Figure 5 presents a simplified Petri net model for complex scenarios based on traffic regulation.

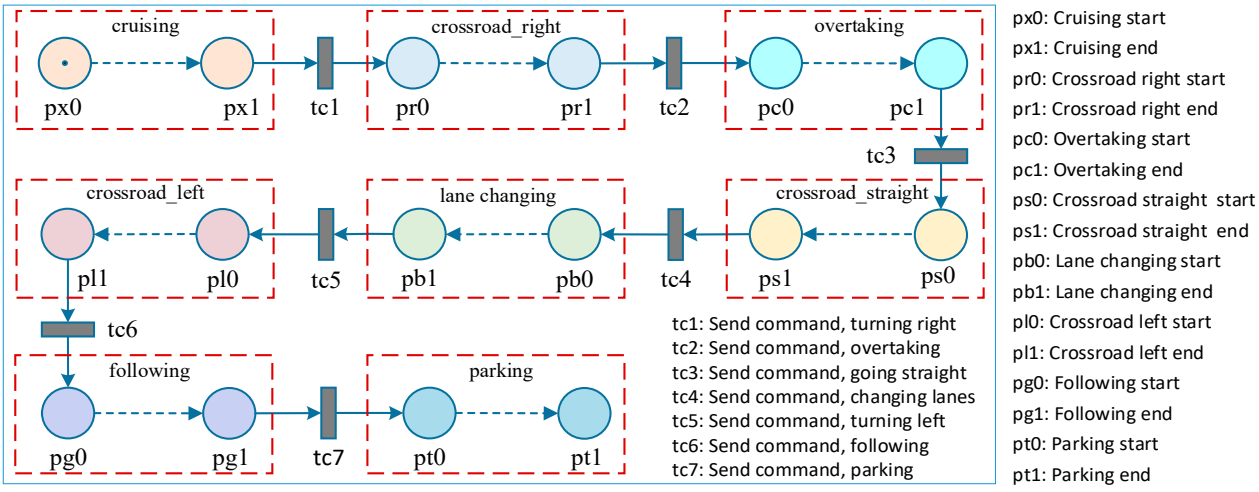


Figure 5. Simplified diagram of the Petri net model.

In our previous research, we presented a comprehensive overview of the modeling and validation techniques for crossroad scenarios utilizing Petri nets [51]. Similar methodologies can be applied to develop Petri net models for basic scenarios, including cruising, overtaking, lane changing, following, and parking, which will not be expounded upon here. In this section, our main contribution lies in the construction of basic scenarios based on traffic regulations, and the establishment of complex scenario models based on the combination of scenario relationships. For instance, as depicted in the simplified Petri net model in Figure 5, the combination of basic scenario Petri net models can be used to construct complex scenario models based on traffic regulations, as shown in Figure 6.

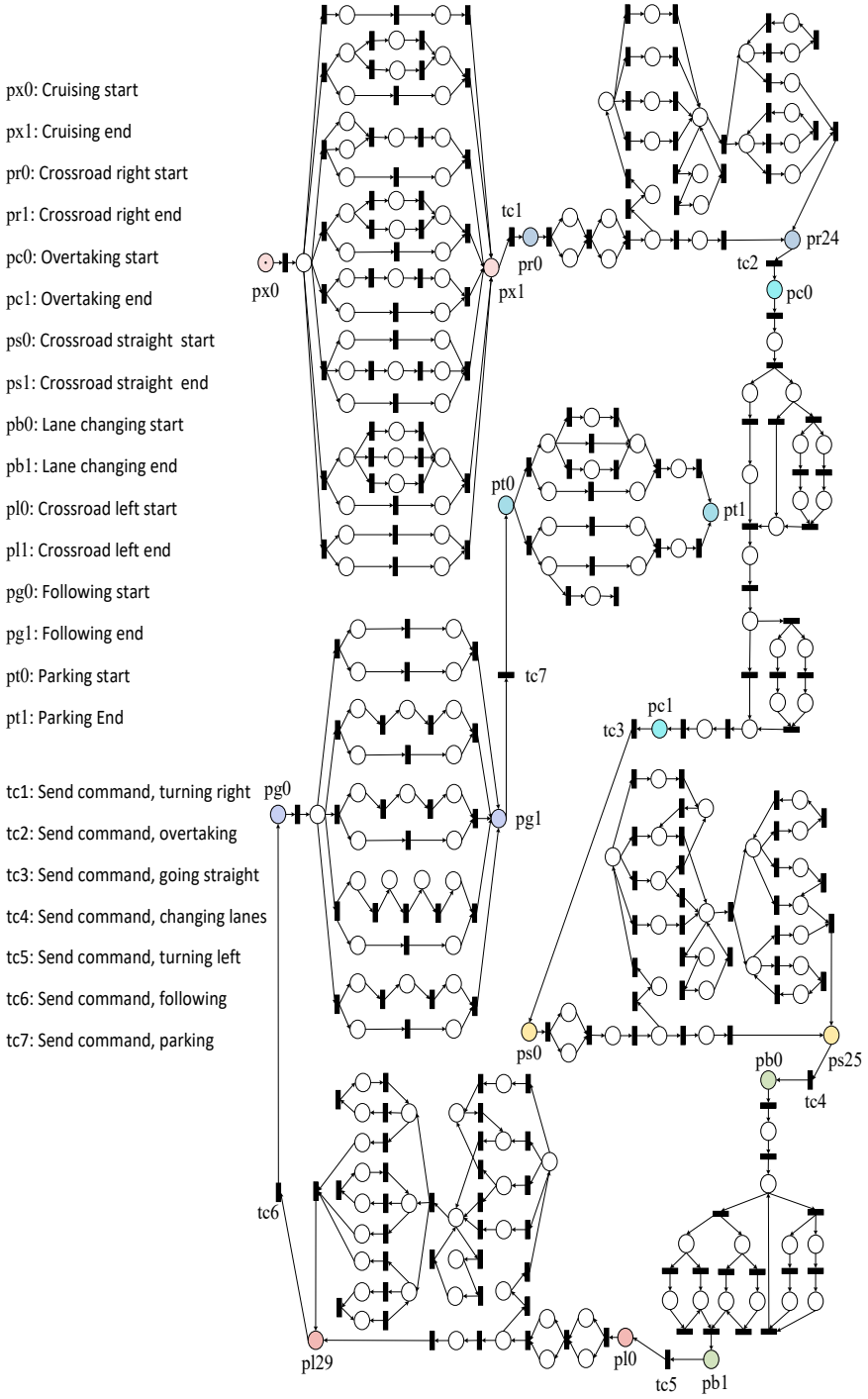


Figure 6. Petri net model for traffic regulation scenarios.

4.3. Scenario Model Validation

The Petri net model for complex scenarios based on traffic regulations undergoes model verification, involving correctness verification and consistency verification. In the correctness verification, formal verification of the scenario model is carried out using MyCCSL software v1.4.15, based on the transformation rules from Petri nets to CCSL constraints. Table 2 presents the experimental results for basic scenarios such as cruising, lane changing, overtaking, following, and parking, indicating that the constructed complex scenario models are deadlock-free and schedulable, thus demonstrating the correctness of the constructed complex scenario models.

Table 2. Results of correctness verification for traffic regulation scenario models.

Scenario	Consistency	Boundary						Correctness		
		30		40		50		Activity	Boundness	Reachability
		Result	Time(s)	Result	Time(s)	Result	Time(s)			
Cruising	satisfiability	√	293.92	√	921.57	√	1681.23	√	√	√
	deadlock	x	352.36	x	961.54	x	1548.41			
Lane changing	satisfiability	√	14.46	√	39.17	√	97.16			
	deadlock	x	22.89	x	80.44	x	153.34			
Overtaking	satisfiability	√	34.47	√	70.17	√	151.68			
	deadlock	x	41.95	x	123.83	x	255.47			
Following	satisfiability	√	94.82	√	219.52	√	529.25			
	deadlock	x	122.62	x	264.97	x	652.67			
Parking	satisfiability	√	19.70	√	44.72	√	167.63			
	deadlock	x	23.27	x	56.21	x	204.49			

In the consistency verification, the consistency between the model design of complex scenarios and the traffic regulation requirements of the system is examined using LTL formulas describing traffic regulation characteristics. Table 3 presents the descriptions of typical traffic regulation features, LTL formula representations, and experimental results within the basic scenarios. The results indicate that the model design has passed the consistency verification of the traffic regulation requirements. In particular, detailed verification results for the intersection scenario can be found in reference [51].

Table 3. Results of consistency verification for traffic regulation scenario models.

	Describe	LTL Formula	Result
Cruising	If encountering a stationary obstacle, then maneuver to avoid it.	$G((px21\_obstacle)\rightarrow(X((px23\_change.left)\vee(px24\_change.right))))$	√
	If encountering pedestrians crossing at a crosswalk, then yield to the pedestrians.	$G((px62\_mark)\rightarrow(X(px65\_stop)))$	√
	If encountering fire vehicles, then let the fire vehicles.	$G((px81\_truck)\rightarrow(X(px83\_slow)))$	√
Lane changing	If traveling in the current lane, then maintain a safe distance from the longitudinal target vehicle 2 and the lateral target vehicle 3.	$G((pb02\_driving)\rightarrow(F((pb03\_distance2.safe)\wedge(pb04\_distance3.safe))))$	√
	If the relative distance to the target vehicle is greater than the longitudinal and transverse safe lane change distance, then change the lane	$G(((pb09\_distance.vertical)\wedge(pb10\_distance.horizontal))\rightarrow(F(pb1\_change)))$	√

Table 3. *Cont.*

	Describe	LTL Formula	Result
Overtaking	If following another vehicle, then maintain a safe lane-changing distance.	$G((pc02\_travel.follow) \rightarrow (F(pc08\_change.safe)))$	√
	If there is a safe lane-changing distance, then overtake the target vehicle on the left.	$G((pc08\_change.safe) \rightarrow (X(pc09\_overtake.left)))$	√
Following	If the followed target vehicle 1 changes lanes, then adjust the speed to follow target vehicle 2.	$G((pg21\_near.target1) \rightarrow (F(pg24\_follow.target2)))$	√
	If approaching target vehicle 2 and target vehicle 1 changes lanes, then follow target vehicle 1.	$G((pg31\_near.target2) \rightarrow F(pg34\_follow.target1)))$	√
Parking	If there is a safe lane-changing distance and the speed limit requirements are met, then change to the right lane and decelerate.	$G(((pt12\_adjust.speed) \wedge (pt13\_change.distance)) \rightarrow (X(pt16\_go.right)))$	√
	If there is a safe lane-changing distance and the speed limit requirements are met, then change to the emergency lane and decelerate.	$G(((pt22\_adjust.speed) \wedge (pt23\_change.distance)) \rightarrow (X(pt25\_go.emergency)))$	√

The formal verification results of the model design and system requirements indicate that the paths of the Petri net scenario model are capable of representing traffic regulations. For instance, the intersection scenario model has successfully undergone validation for the "yield to pedestrians at intersections", signifying that the corresponding paths of the model can adequately depict the traffic regulation of "yielding to pedestrians at intersections". Therefore, TraModeAVTest utilizes the Petri net model of traffic regulation scenarios to generate test cases that cover model paths, which can characterize traffic regulation and be used to test the violations of traffic regulation for AVs, thereby improving the safety and reliability of ADS.

5. Violation Testing Method

In the previous section, we provided a method for constructing complex scenario models based on traffic regulations and formally verified that the paths of the Petri net model can characterize traffic regulations. Consequently, in this section, we introduce a method for generating test cases to cover model paths, utilizing the scenario model of traffic regulations, in order to assess the ADS for violations of traffic regulations. Initially, we employ a method of traversing the model to generate state invocation sequences without test cases, i.e., the test paths of the Petri net model. Subsequently, we employ a method of parameter combination to generate executable state invocation sequences for covering model paths, i.e., the test cases for covering model paths.

5.1. Test Path Generation

In this paper, the coverage criteria for Petri nets primarily include the place coverage criterion, transition coverage criterion, and place-transition coverage criterion.

1. Place coverage criterion: The test case set should ensure that all places in the Petri net model are accessed at least once.
2. Transition coverage criterion: The test case set should ensure that all transitions in the Petri net model are activated at least once.
3. Place-transition coverage criterion: The test case set should ensure that all places in the Petri net model are accessed at least once, and all transitions are activated at least once.

The Petri net model of the autonomous driving scenario based on traffic regulations exhibits a fundamental structure comprising sequential structure, selection structure, loop structure, synchronization structure, and concurrent structure. In the sequential structure, test paths are covered sequentially according to the ordered relationship of places and transitions. Within the selection structure, each test path is traversed, ensuring that each branch is explored at least once. Notably, the mutually exclusive branches of the selection structure cannot appear simultaneously in the same test path, as depicted in Figure 7a. Regarding the loop structure, to ensure comprehensive testing, scenarios where the loop body is executed 0 times and 1 time are considered. Consequently, the loop structure can be transformed into a selection structure, as illustrated in Figure 7b.

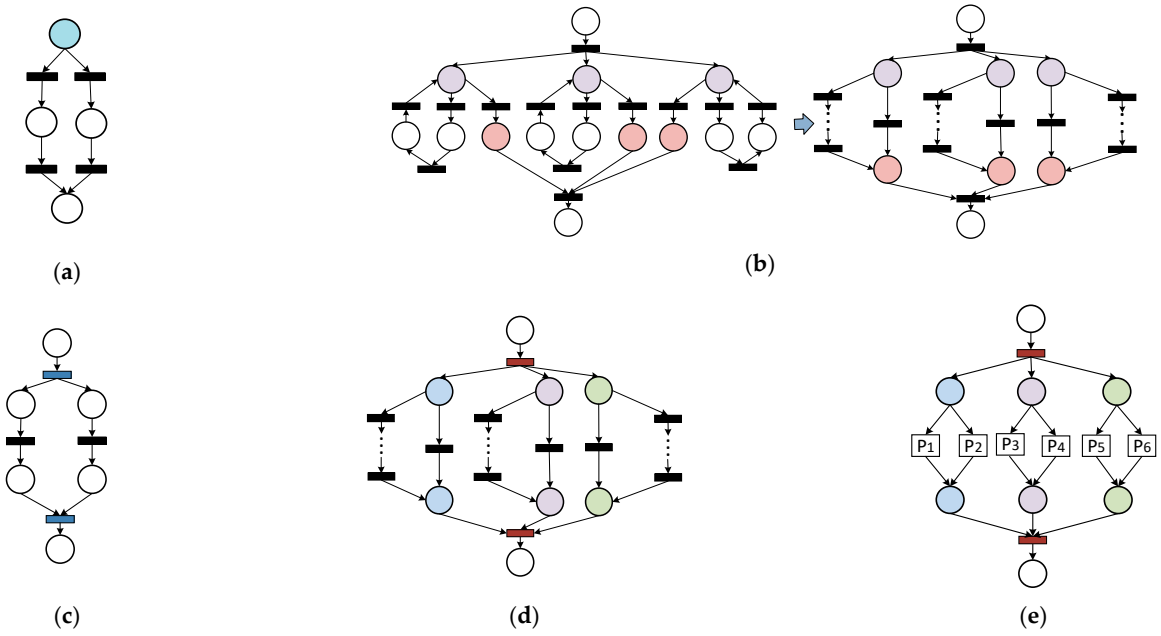


Figure 7. Petri net model for traffic regulation scenarios. (**a**) Selection structure. (**b**) Loop structure transform selection structure. (**c**) Concurrent-synchronization simple structure. (**d**) Concurrent-synchronization complex structure. (**e**) Simplified diagram of concurrent—synchronization complex structure.

For the concurrent and synchronization structures within Petri net models, the presence of mutually exclusive branches poses a challenge for testing. We will now analyze the two cases of whether mutually exclusive branches exist in concurrent-synchronous structures.

1. Non-existence of mutually exclusive branches: Within the concurrent-synchronization structure, there is not just a single path for the interacting transitions; rather, there are no mutually exclusive branches, as depicted in Figure 7c. In this simple structure devoid of mutually exclusive branches, there are numerous methods for selecting concurrent-synchronization paths. By employing a random generation approach to sequentially select paths for testing, the outcomes remain consistent, thus mitigating issues such as combinatorial explosion and repetitive testing.
2. Existence of mutually exclusive branches: Within the concurrent-synchronization structure, there are multiple paths for the interacting transitions, and mutually exclusive branches exist, as depicted in Figure 7d. In its simplified Figure 7e, the mutually exclusive branches are P_1 with P_2, P_3 with P_4, and P_5 with P_6. Analysis of the selection structure reveals that mutually exclusive paths cannot be executed simultaneously; thus, these mutually exclusive branches cannot be directly chosen as concurrent-synchronization paths. In the concurrent-synchronization structure, if the number of groups of interacting places is denoted as m, and the corresponding

number of mutually exclusive branches is $n_1, n_2, \ldots n_m$, then the combination number of the interacting places is $n_1 \times n_2 \times \ldots \times n_m$, with each combination representing a concurrent-synchronization path. In the concurrent-synchronization structure, the number of concurrent-synchronization paths increases when multiple branches are involved, leading to a combinatorial explosion. In response to this scenario, we adopt the pairwise combination testing coverage method.

In the concurrent-synchronization structure, when the number of groups of interacting places is m, and the number of mutually exclusive branches is $n_1, n_2, \ldots n_m$, the pairwise combination coverage criterion mandates that every possible execution path combination of any two branches must be encompassed by at least one test case. This criterion ensures both the completeness and sufficiency of testing, while also mitigating the issues related to combinatorial explosion. In Figure 7e, $P_1 \ldots P_6$ represent the six paths of the model. Employing the conventional combination method, the number of concurrent-synchronous paths generated is $2 \times 2 \times 2 = 8$, whereas using the pairwise combination method produces a distinct number, specifically four paths: $\{P_1, P_3, P_5\}$, $\{P_1, P_4, P_6\}$, $\{P_2, P_3, P_6\}$, and $\{P_2, P_4, P_5\}$. $\{P_1, P_3, P_5\}$ covers $\{P_1, P_3\}$, $\{P_1, P_5\}$, and $\{P_3, P_5\}$; $\{P_1, P_4, P_6\}$ covers $\{P_1, P_4\}$, $\{P_1, P_6\}$, and $\{P_4, P_6\}$; $\{P_2, P_3, P_6\}$ covers $\{P_2, P_3\}$, $\{P_2, P_6\}$, and $\{P_3, P_6\}$; $\{P_2, P_4, P_5\}$ covers $\{P_2, P_4\}$, $\{P_2, P_5\}$, and $\{P_4, P_5\}$. Thus, the pairwise combination method can cover all potential combinations of execution paths for any two branches, ensuring both the completeness and sufficiency of testing while averting a combinatorial explosion.

To ensure effective testing and reduce redundancy, the guiding principle for searching test paths in the model is that the set of all paths must cover all places and transitions in the model, branches should be executed at least once, and loop paths should be executed at most once. The complex structures of concurrency and synchronization are addressed by the pairwise combination criterion, and the test path set should encompass all paths from the initial state to the terminal state of the Petri net model, thus representing all traffic regulations in the scenario model.

5.2. Test Case Generation

TraModeAVTest defines abstract scenarios and specifies fixed features, such as the initial position of the ego vehicle (EGO), to search for potential collision test cases using the parameter combination method. Within the abstract scenario, each scenario has distinct configuration parameters that are parameterized through variables defined on feature thresholds. In the search space, test cases involve assignments of these variables, and specific scenarios can be derived by instantiating abstract scenarios with assigned variable values. For example, initial test cases are first generated, which are test cases without any non-player character vehicle (NPC), then complex scenario test cases are explored by adding NPCs to the test scenario. It is evident that the occurrence of a collision is influenced by the relative distance and velocity between the EGO and the NPC. Once the initial velocity of the NPCs is determined, each test case can be parameterized by the initial position and velocity of the EGO, as well as the relative distance between the EGO and the NPCs. Consequently, each test case can be described as a vector $t = (s, v, d)$, with s representing the initial position of the EGO, v representing the initial velocity of the EGO, and d representing the relative distance between the EGO and the NPC.

The Petri net scenario model based on traffic regulations, TraModeAVTest aims to generate test cases using the parameter combination method with the objective of collision occurrence. Initially, initial test cases that cover the model paths are generated, i.e., test cases without any NPCs, to ensure the effectiveness of the generated test cases. For the test path P^j of the scenario model, $\forall P^j \in PC$, where PC represents the set of test paths. The initial test case is denoted by $t^j_{i0} = \left(s^j_{i0}, v^j_{i0}, \varnothing\right)$, s^j_{i0} representing the initial position of the EGO, and v^j_{i0} representing the initial velocity of the EGO, $\varnothing$ indicating the absence of NPCs. $\forall t^j_{i0} \in T^j_i$, T^j_i is the initial test case set of the i-th meta-scenario in the test path P^j. $\forall T^j_i \in T^j$,

T^j represents the test case set of covered P^j, $\forall T^j \in TC$, while TC represents the test case set of covering all test paths.

Subsequently, collision test cases are identified by adding the NPCs, as illustrated in Figure 8. Given an initial test case of a meta-scenario covering the model path, $P^j = Cov\left(t_{i0}^j\right)$, the new test cases are generated by adding the NPCs, $t_{ik}^j = \left(s_{ik}^j, v_{ik}^j, d_{ik}^j\right)$. These new test cases encompass the position s_{ik}^j and the velocity v_{ik}^j of the EGO, as well as the relative distance d_{ik}^j between the EGO and the NPC. Based on the threshold values of the variables set by traffic regulation features, with the objective of collision occurrence, the parameter values of the variables are adjusted to obtain a test case set T_i^j of a meta-scenario covering the model paths, $\forall t_{ik}^j \in T_i^j$.

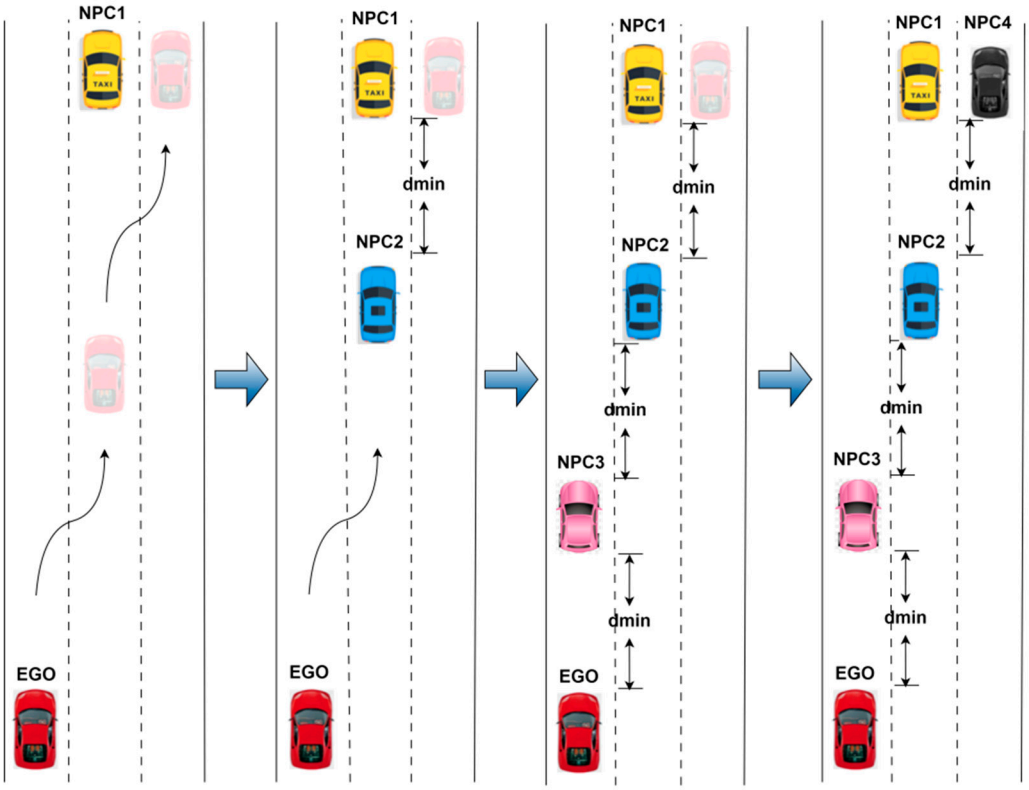


Figure 8. Workflow for test case generation.

Then, various parameters of the meta-scenario variables are chosen for combination, generating executable state invocation sequences of parameter combinations, i.e., a test case set of traffic regulation scenarios covering the model paths, $P^j = Cov(T^j), \forall T^j \in TC$. In order to ensure test case diversity and prevent the creation of excessive redundant test cases, the principle dictates the selection of distinct meta-scenario combinations from the basic scenario when the same basic scenario is encountered multiple times in traffic regulation scenarios. Consequently, this approach, based on the complexity of the actual scenarios arising from the combination of basic scenarios, can effectively meet the demand for complex test scenarios in the context of the ADS.

Algorithm 1 describes the method for generating test cases covering model paths. Initially, it produces the initial test case for each meta-scenario and verifies the validity of the generated test cases. If the generated initial test case covers the model path, it is added to the test case set for the meta-scenario (lines 2–6). Otherwise, the initial test case is regenerated until a valid test case is obtained (lines 7–13). Subsequently, it generates collision test cases for the meta-scenario and checks whether the newly generated test cases cover the model path. If they do, they are added to the test case set for the meta-scenario,

and the initial test case for that meta-scenario is removed (lines 14–18). Conversely, the test case set for the meta-scenario remains unaltered until a valid collision test case is generated (lines 19–20). Ultimately, the generated meta-scenario test cases are integrated into the test case set covering the model paths, thereby obtaining a test case set covering all model paths (lines 21–22).

Algorithm 1: Test Case Generation for Path Coverage.

Input: Set of test paths PC for the Petri net model.
Output: Test case set TC covering model paths.
Initialize: $TC = \emptyset$;
1 for $\forall P^j \in PC$ do
2 for $i = 0:n$ do
3 $t_{i0}^j = (s_{i0}^j, v_{i0}^j, \emptyset)$; /*Generate initial test cases for meta-scenarios without obstacles.*/
4 if $Cov(t_{i0}^j) \in P^j$ then
5 $T_i^j = T_i^j \cup \{t_{i0}^j\}$;
6 else
7 while $t_{i0}^j = (s_{i0}^j, v_{i0}^j, \emptyset)$ do
8 if $Cov(t_{i0}^j) \in P^j$ then
9 $T_i^j = T_i^j \cup \{t_{i0}^j\}$;
10 break;
11 else
12 continue;
13 while $t_{ik}^j = (s_{ik}^j, v_{ik}^j, a_{ik}^j)$ do /*Adjust variable parameter values and generate collision test cases.*/
14 if $Cov(t_{ik}^j) \in P^j$ then
15 $T_i^j = T_i^j \cup \{t_{ik}^j\}$;
16 $T_i^j = T_i^j \backslash \{t_{i0}^j\}$;
17 break;
18 else
19 continue;
20 $T^j = T^j \cup \{T_i^j\}$;
21 $TC = TC \cup \{T^j\}$; /*Generate a test case set covering the paths.*/

5.3. Simulation Experiment

In order to assess the performance of TraModeAVTest, we carried out simulation experiments on the Baidu platform Apollo 6.0, which is recognized as one of the most widely used ADSs in both academia and industry. The simulation environment employed was LGSVL 2021.3, and the experiments were executed on a desktop computer running Ubuntu 18.04, featuring an AMD Ryzen 9 3950x CPU, 64 GB of RAM, and an NVIDIA GeForce RTX 3090 GPU.

The primary research questions are as follows:

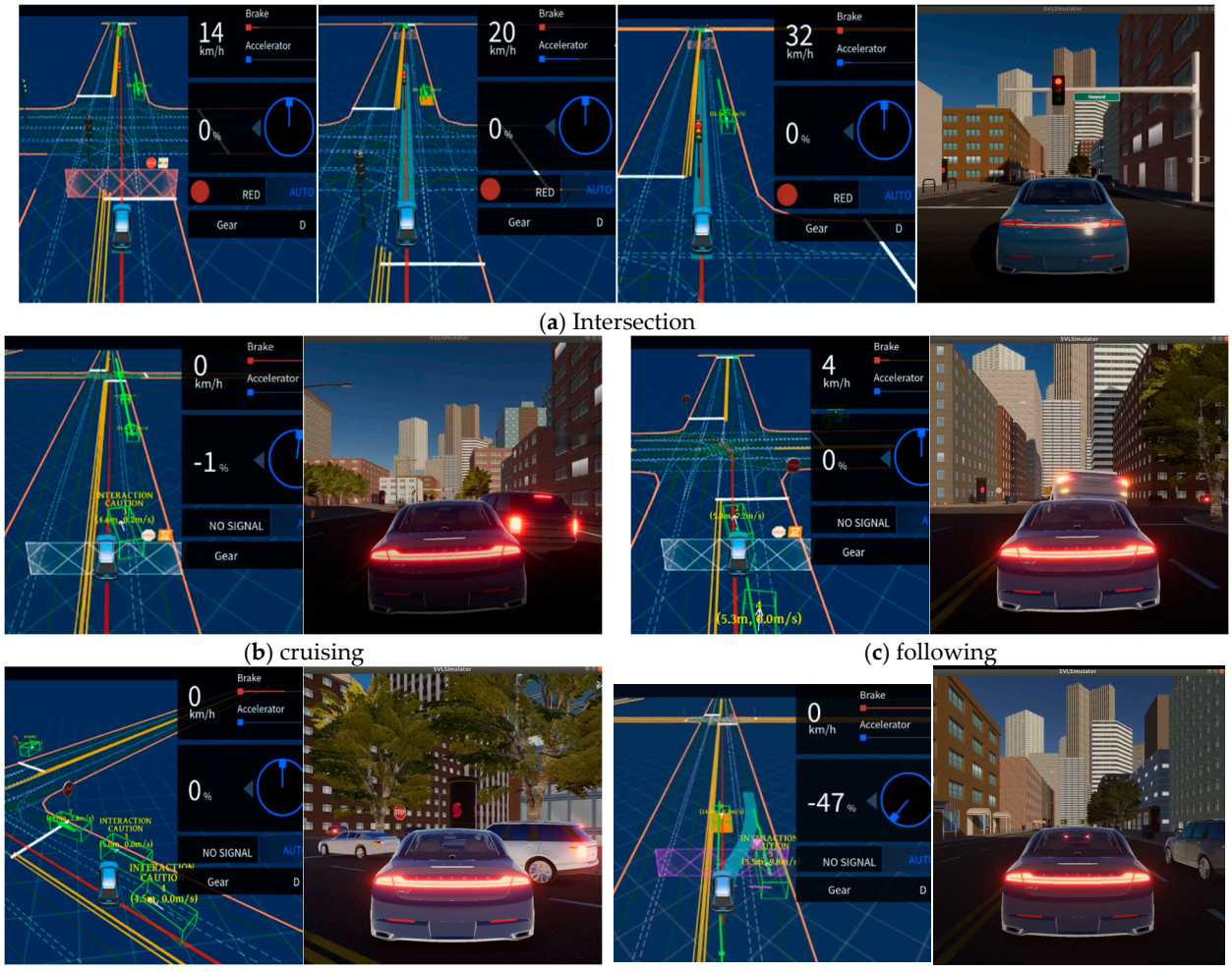
1. Effectiveness: Can the test cases generated by TraModeAVTest effectively detect violations of the ADS?
2. Efficiency: Can TraModeAVTest improve the efficiency of generating violation scenarios compared to baseline methods?

The primary traffic facilities in the simulation environment include urban roads with straight segments and intersections. Traffic control devices consist of traffic lights, road markings, and traffic signs. To ensure that the simulation experiments start from a valid and meaningful state, the initial state of the simulation environment is constrained such that a safe distance exists between the EGO and the NPCs when all vehicles are initiated. Furthermore, TraModeAVTest is not confined to a specific simulator. The selection of LGSVL is based on its compatibility with Apollo, but it can also be migrated to other simulators such as Calar, provided an API interface link to the ADS is available.

5.3.1. Effectiveness Analysis

TraModeAVTest models testing scenarios based on traffic regulations and subsequently generates test cases that represent regulations. Its primary objective is to test the violation of traffic regulations by AVs, with the aim of enhancing the safety of ADSs. The traffic regulation scenarios in this study encompass six fundamental scenarios: intersection, cruising, following, lane changing, overtaking, and parking. However, violations in parking scenarios are not within the scope of this analysis. Consequently, the following typical instances of five basic scenario types are discussed based on violations of traffic regulations by AVs, in order to explore potential causes of traffic accidents involving AVs and identify potential safety issues in ADSs, so as to illustrate the effectiveness of TraModeAVTest.

1. Intersection scenario: The EGO runs a red light and accelerates through the intersection, as shown in Figure 9a. When approaching the intersection stop line, the speed of the EGO is 14 km per hour, and the traffic signal is red. Despite the absence of a collision, the EGO chooses to proceed through the intersection instead of stopping. Upon entering the intersection, the speed of the EGO is 20 km per hour, posing a significant danger. As the EGO exits the intersection at a speed of 32 km per hour, it is already speeding. In this scenario, if the NPCs are passing through the intersection, the EGO running a red light and speeding could likely lead to a traffic accident.



(**a**) Intersection

(**b**) cruising

(**c**) following

(**d**) lane changing

(**e**) overtaking

Figure 9. Scenario diagram of simulation experiment.

In the intersection scenario, the traffic regulations are "prohibition of vehicle passage when the red light is on" and "the maximum speed for motor vehicles passing through

intersections should not exceed 30 km per hour". When the traffic signal at the intersection is red, the EGO not only fails to decelerate or stop but also accelerates, and subsequently exceeds the speed limit while passing through the intersection. Therefore, although no traffic accident occurred, the traffic violations of running a red light and speeding by the EGO are likely to cause accidents, indicating significant safety issues in the perception or prediction modules of the Apollo system.

2. Cruising scenario: The EGO engaged in cruising, collides with an NPC that cuts in from the right, as shown in Figure 9b. As the EGO cruises, an NPC from the adjacent right lane cuts in. The insufficient safe distance between the EGO and the NPC, coupled with the failure of the EGO to yield to the NPC in a timely manner, results in a collision between the two vehicles. In this scenario, if the EGO maintains a sufficient safe distance from the NPC and brakes promptly to yield, the collision could be avoided.

In the cruising scenario, the traffic regulation is "when motor vehicles are traveling on the road, they should yield to other vehicles and maintain a necessary safety distance". When the NPC in the right lane merges into the cruising lane, the EGO fails to yield in a timely manner, resulting in a collision between the two vehicles. Therefore, it is evident that the lack of maintaining a necessary safety distance by the EGO and its failure to decelerate and yield to the NPC are the most likely causes of the collision. The former indicates a safety issue in the positioning or planning module of the Apollo system, while the latter suggests a safety issue in the prediction or control module of the Apollo system.

3. Following scenario: The EGO following another vehicle, collides with a decelerating NPC in front, as shown in Figure 9c. As the NPC in front decelerates while approaching an intersection, the EGO following behind, maintains its current velocity. The inadequate safe distance between the EGO and the NPC leaves insufficient time for deceleration and braking, leading to the EGO rear-ending the NPC. In this scenario, maintaining a proper safe distance between the EGO and the NPC could have averted the rear-end collision.

In the following scenario, the traffic regulation is "when motor vehicles are following other vehicles, they should maintain a sufficient safety distance from the vehicle in front". In this instance, the EGO fails to uphold a proper safety distance while following the NPC, leading to a rear-end collision when the NPC decelerates. This underscores that the failure of both the EGO and NPC to maintain an adequate safety distance is the primary cause of the rear-end collision, thereby highlighting a safety issue in the positioning or planning module of the Apollo system.

4. Lane changing scenario: The EGO abruptly changes direction and plans to switch to the right lane as it approaches the intersection, as shown in Figure 9d. As the EGO is about to reach the intersection and the adjacent right lane NPC is traveling at a high speed, the sudden lane change by the EGO nearly causes a traffic accident. Furthermore, the abrupt lane change of the EGO obstructs the normal movement of the NPC behind it, almost resulting in a rear-end collision. In this situation, the EGO should maintain its original lane and proceed through the intersection as usual.

In the lane changing scenario, the traffic regulation is "motor vehicles should not change lanes when approaching an intersection". As the EGO approached the intersection, it abruptly changed lanes, narrowly avoiding a traffic accident. This implies that while no accident occurred, the sudden lane change violation of the EGO is likely to result in a traffic accident, indicating a safety issue within the planning or control module of the Apollo system.

5. Overtaking scenario: The EGO erroneously plans an overtaking route, as shown in Figure 9e. As the front NPC decelerates and the NPC in the adjacent lane behind accelerates rapidly, the EGO generates a trajectory to change lanes to the right for a forced overtaking, almost resulting in a traffic accident. In this situation, both the EGO

and NPC should maintain a sufficient safe distance, and the EGO should decelerate and continue in its original lane.

In the overtaking scenario, the traffic regulation is "overtaking motor vehicles should not impede the normal driving of motor vehicles in the relevant lane". When the NPC in the front lane decelerated while the NPC in the adjacent lane rapidly approached, the EGO generated a trajectory to forcefully overtake by changing lanes to the right, blatantly violating the traffic regulations. This indicates that although the EGO and NPC did not collide, the severe traffic violation of the EGO is likely to cause a traffic accident, underscoring a very serious safety issue within the planning module of the Apollo system.

From the above analysis, it can be concluded that TraModeAVTest can not only identify issues where AVs are involved in collisions, but also reveal a broader spectrum of traffic regulation violations by AVs. For example, TraModeAVTest has identified cases where AVs have incorrectly planned overtaking maneuvers, and such violations represent significant safety risks that can lead to traffic accidents. In conclusion, the test cases generated by TraModeAVTest have the ability to characterize traffic regulations and effectively identify violations of traffic regulations by AVs. These violations are also crucial factors leading to traffic accidents involving AVs, which shows the effectiveness and rationality of the application of TraModeAVTest to test the violations of AVs.

5.3.2. Efficiency Analysis

In order to assess the efficiency of TraModeAVTest in generating violation scenarios, we compared TraModeAVTest with two representative baseline methods, Random and AV-Fuzzer [53], in terms of both the number and time taken to discover autonomous vehicle violations. Random testing does not account for the influence of traffic regulations and simply generates test cases through parameter combinations. AV-Fuzzer, represents an advanced autonomous driving testing technique. It utilizes genetic algorithms to search for high-risk scenarios that contravene safety requirements and subsequently employs a local fuzzer to identify safety violations. This method is widely utilized in the assessment of autonomous driving testing techniques [54]. The experimental comparison results of TraModeAVTest, Random, and AV-Fuzzer are shown in Table 4.

Table 4. Experimental comparison results.

	Random	**AV-Fuzzer**	**TraModeAVTest**
Total number of test cases	437	392	384
Number of test cases for the violation scenarios	113	96	125
Number of types of violation scenarios	4	3	5
Average time for each violation scenario	2.5 h	3.3 h	2.0 h

As shown in Table 4, TraModeAVTest generated 384 test cases, Random generated 437 test cases, and AV-Fuzzer generated 392 test cases, indicating that TraModeAVTest produced the fewest test cases. Compared to the baseline method, TraModeAVTest reduced the total number of test cases by 12.13% and 2.04%, i.e., 384 vs. 437 and 384 vs. 392. However, TraModeAVTest generated 125 test cases for violation scenarios, including 102 violation scenarios for the EGO and 23 violation scenarios for the NPC. Random generated 113 test cases for violation scenarios, including 89 violation scenarios for the EGO and 24 violation scenarios for the NPC. AV-Fuzzer generated 96 test cases for violation scenarios, including 75 violation scenarios for the EGO and 21 violation scenarios for the NPC. The number of violation scenarios generated by the three methods for EGO and NPC is illustrated in Figure 9.

From Figure 10, it is evident that TraModeAVTest generated the highest number of test cases for EGO violation scenarios. Compared to the baseline method, TraModeAVTest increased the number of valid test cases by 14.61% and 36%, i.e., 102 vs. 89 and 102 vs. 75. This indicates that, in comparison to the baseline method, TraModeAVTest is capable

of identifying more behaviors of traffic regulation violations by AVs while reducing the overall number of generated test cases. In other words, although the baseline method generated a larger quantity of test cases, it identified fewer behaviors of traffic regulation violations, suggesting that TraModeAVTest generates a greater number of effective test cases while the baseline method generates more redundant test cases.

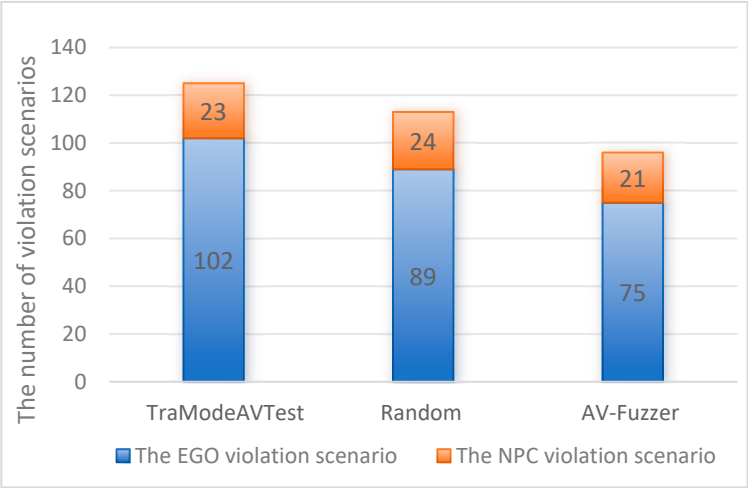


Figure 10. Comparison diagram of violation scenario.

The experimental results in Table 4 also indicate that the test cases generated by TraModeAVTest for EGO violations cover all five basic scenario types, excluding parking scenarios. In contrast, the EGO violation test cases generated by Random encompass four basic scenario types, excluding the scenario of incorrectly planned overtaking routes. Meanwhile, the EGO violation test cases generated by AV-Fuzzer include three basic scenario types, omitting violations such as running red lights at intersections and changing lanes near intersections. Furthermore, the violation scenario types for all three methods are observed within the initial 10 h of the experiment. The average time to generate different types of violation scenarios during the search process is computed as the ratio of the total simulation time to the number of violation scenario types. Specifically, the average time for Random to generate different types of violation scenarios is 2.5 h, for AV-Fuzzer it is 3.3 h, and for TraModeAVTest it is 2 h. Consequently, it is apparent that, in comparison to the best baseline methods, TraModeAVTest achieves an average time reduction of 0.5 h in generating different types of violation scenarios within the constrained time allocation for test case generation, signifying an efficiency improvement of 20%.

Furthermore, it is noteworthy that all three methods can identify issues related to collisions involving AVs, yet TraModeAVTest excels in identifying a greater number of violations of traffic regulations by AVs. For instance, Random failed to identify instances of AVs incorrectly planning overtaking routes, and AV-Fuzzer did not identify instances of AVs running red lights at intersections. These violations of regulations by AVs represent significant safety hazards that can lead to traffic accidents.

6. Conclusions

A method has been proposed for modeling scenarios using traffic regulations and generating test cases to address the compliance and safety testing issues of ADSs, with the aim of identifying the safety violation behavior of ADSs. For compliance testing, a method has been developed to construct Petri net models of complex autonomous driving scenarios based on traffic regulations and the combined relationships of scenarios. Formal methods were employed to verify the consistency between the scenario model design and the system's regulatory attributes, with the goal of providing scenario model support to identify behaviors of ADSs that violate traffic regulations. For safety testing, coverage criteria suitable for Petri net models were introduced, and a search strategy was used to

generate test paths for scenario models. Additionally, a parameter combination method was employed to generate test cases for covering model paths, specifically representing test cases for traffic regulations, with the aim of providing specific test scenarios for testing the behaviors of ADSs that violate safety regulations. The results of simulation experiments on the Baidu Apollo platform demonstrate that the test cases generated by TraModeAVTest can effectively uncover safety violations in ADSs, including three types of safety violation scenarios and two types of regulatory violation scenarios. The most probable cause of these safety violations is identified as the AVs' non-compliance with traffic regulations, which is also the fundamental cause of traffic accidents involving AVs. This validates the effectiveness of TraModeAVTest in utilizing traffic regulations to construct scenario models and test safety violations in AVs. In addition, TraModeAVTest can effectively improve the efficiency of generating different types of safety violation scenarios. By comparing with the experimental results of the baseline method, TraModeAVTest improves the efficiency of generating different types of safety violation scenarios by 20%.

Future work will involve researching additional methods, such as evolutionary methods, for modeling complex autonomous driving scenarios based on traffic regulations and testing the violation behavior of AVs. Furthermore, there will be a focus on studying optimization methods for test cases to improve the testing efficiency of ADSs.

Author Contributions: Conceptualization, C.X. and S.H.; methodology, C.X. and C.Z.; algorithm, C.Z. and C.X.; software, T.B. and L.S.; validation, C.X. and Z.Y.; investigation, C.X. and Z.Y.; writing—original draft preparation, C.X.; writing—review and editing, C.X. and S.H.; project administration, T.B. and L.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Heilongjiang Provincial Department of Education project, grant number 1453ZD018; Discipline Construction Project of Mudanjiang Normal University, grant number MSYSYL2022008; Basic Research Fund Project of Provincial Universities in Heilongjiang Province, grant number 1355JG011 and 1451TD003; Key Commissioned Project for Higher Education Teaching Reform in Heilongjiang Province, grant number SJGZ20200175; Natural Science Fund Project of Heilongjiang Province, grant number LH2020F038.

Data Availability Statement: No new datasets were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Tang, Y.; Zhou, Y.; Liu, Y.; Sun, J.; Wang, G. Collision avoidance testing for autonomous driving systems on complete maps. In Proceedings of the 2021 IEEE Intelligent Vehicles Symposium (IV), Nagoya, Japan, 11–17 July 2021; pp. 179–185. [CrossRef]
2. An, D.; Liu, J.; Chen, X.; Sun, H.Y. Formal modeling and dynamic verification for human cyber physical systems under uncertain environment. *J. Softw.* **2021**, *32*, 1999–2015. [CrossRef]
3. Wu, H.; Wang, H.; Su, X.; Li, M.; Xu, F.; Zhong, S. Security testing of visual perception module in autonomous driving system. *J. Comput. Res. Dev.* **2022**, *59*, 1133–1147. [CrossRef]
4. Zhao, Xiang-mo's Team Supported by the National Key Research and Development Program of China (2021YFB2501200). Research progress in testing and evaluation technologies for autonomous driving. *J. Traffic Transp. Eng.* **2023**, *23*, 10–77. [CrossRef]
5. Rong, G.; Shin, B.; Tabatabaee, H.; Lu, Q.; Steve, L.; Možeiko, M.; Boise, E.; Uhm, G.; Gerow, M.; Mehta, S.; et al. LGSVL simulator: A high fidelity simulator for autonomous driving. In Proceedings of the 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), Rhodes, Greece, 20–23 September 2020; pp. 1–6. [CrossRef]
6. Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An open urban driving simulator. In Proceedings of the 1st Annual Conference on Robot Learning (CoRL), Mountain View, CA, USA, 10 November 2017; pp. 1–16. [CrossRef]
7. Deng, W.; Li, J.; Ren, B.; Wang, W.; Ding, J. A survey on automatic simulation generation methods for autonomous driving. *China J. Highw. Transp.* **2022**, *35*, 316–333. [CrossRef]
8. Yu, R.; Tian, Y.; Sun, J. Highly automated vehicle virtual testing: A review of recent developments and research frontiers. *China J. Highw. Transp.* **2020**, *33*, 125–138. [CrossRef]
9. Zhou, Y.; Sun, Y.; Tang, Y.; Chen, Q.; Sun, J. Specification-based Autonomous Driving System Testing. *IEEE Trans. Softw. Eng.* **2023**, 1–19. [CrossRef]
10. Deng, Y.; Yao, J.; Tu, Z.; Zheng, X.; Zhang, M.; Zhang, T. Target: Traffic rule-based test generation for autonomous driving systems. *arXiv* **2023**, arXiv:2305.06018. [CrossRef]

11. Lou, G.; Deng, Y.; Zheng, X.; Zhang, M.; Zhang, T. Testing of autonomous driving systems: Where are we and where should we go? In Proceedings of the 2022 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Singapore, 14–18 November 2022; pp. 31–43. [CrossRef]
12. Tesla. "Autonomous Driving Unit". Available online: https://www.tesla.com/autopilot (accessed on 10 April 2022).
13. Baidu. "Apollo Open Platform". Available online: https://apollo.auto/ (accessed on 10 December 2021).
14. Autoware.AI. "Autoware.AI". Available online: https://www.autoware.ai/ (accessed on 17 April 2022).
15. Zhao, D.; Lam, H.; Peng, H.; Bao, S.; LeBlanc, D.; Nobukawa, K.; Pan, C. Accelerated evaluation of automated vehicles safety in lane-change scenarios based on importance sampling techniques. *IEEE Trans. Intell. Transp. Syst.* **2017**, *18*, 595–607. [CrossRef] [PubMed]
16. Althoff, M.; Lutz, S. Automatic generation of safety-critical test scenarios for collision avoidance of road vehicles. In Proceedings of the 2018 IEEE Intelligent Vehicles Symposium (IV), Changshu, China, 26–30 June 2018; pp. 1326–1333. [CrossRef]
17. Klischat, M.; Althoff, M. Generating critical test scenarios for automated vehicles with evolutionary algorithms. In Proceedings of the 2019 IEEE Intelligent Vehicles Symposium (IV), Paris, France, 9–12 June 2019; pp. 2352–2358. [CrossRef]
18. Beglerovic, H.; Stolz, M.; Horn, M. Testing of autonomous vehicles using surrogate models and stochastic optimization. In Proceedings of the 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), Yokohama, Japan, 16–19 October 2017; pp. 1–6. [CrossRef]
19. Chen, R.; Sherony, R.; Gabler, H. Comparison of time to collision and enhanced time to collision at brake application during normal driving. In Proceedings of the SAE 2016 World Congress and Exhibition, Technical Paper 2016–01–1448, Detroit, MI, USA, 5 April 2016. [CrossRef]
20. Mullins, G.; Stankiewicz, P.; Gupta, S. Automated generation of diverse and challenging scenarios for test and evaluation of autonomous vehicles. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 July 2017; pp. 1443–1450. [CrossRef]
21. Mullins, G.; Dress, A.; Stankiewicz, P.; Appler, J.; Gupta, S. Accelerated testing and evaluation of autonomous vehicles via imitation learning. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 21–25 May 2018; pp. 1–7. [CrossRef]
22. Nitsche, P.; Thomas, P.; Stuetz, R.; Welsh, R. Pre-crash scenarios at road junctions: A clustering method for car crash data. *Accid. Anal. Prev.* **2017**, *107*, 137–151. [CrossRef] [PubMed]
23. Gambi, A.; Huynh, T.; Fraser, F. Generating effective test cases for self-driving cars from police reports. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Tallinn, Estonia, 12 August 2019; pp. 257–267. [CrossRef]
24. Hauer, F.; Schmidt, T.; Holzmuller, B.; Pretschner, A. Did we test all scenarios for automated and autonomous driving systems? In Proceedings of the 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 27–30 October 2019; pp. 2950–2955. [CrossRef]
25. Ding, W.; Xu, M.; Zhao, D. Cmts: A conditional multiple trajectory synthesizer for generating safety-critical driving scenarios. In Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 31 May–31 August 2020; pp. 4314–4321. [CrossRef]
26. Chen, J.; Shu, X.; Lan, F.; Wang, J. Construction of automatic driving test scenarios with typical dangerous accident characteristics. *J. South China Univ. Technol.* **2021**, *49*, 1–8. [CrossRef]
27. Roesener, C.; Fahrenkrog, F.; Uhlig, A.; Eckstein, L. A scenario-based assessment approach for automated driving by using time series classification of human-driving behaviour. In Proceedings of the 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil, 1–4 November 2016; pp. 1360–1365. [CrossRef]
28. Pei, K.; Cao, Y.; Yang, J.; Jana, S. DeepXplore: Automated whitebox testing of deep learning systems. *Commun. ACM* **2019**, *62*, 137–145. [CrossRef]
29. Tian, Y.; Pei, Y.; Jana, S.; Ray, B. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In Proceedings of the 40th International Conference on Software Engineering (ICSE), Gothenburg, Sweden, 27 May 2018; pp. 303–314. [CrossRef]
30. Zhang, M.; Zhang, Y.; Zhang, L.; Liu, C.; Khurshid, S. DeepRoad: GAN-based metamorphic testing and input validation framework for autonomousdriving systems. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE), Montpellier, France, 3 September 2018; pp. 132–142. [CrossRef]
31. Caesar, H.; Bankiti, V.; Lang, A.; Vora, S.; Liong, V.; Xu, Q.; Krishnan, A.; Pan, Y.; Baldan, G.; Beijbom, O. nuScenes: A Multimodal Dataset for Autonomous Driving. In Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, 13–19 June 2020; pp. 11618–11628. [CrossRef]
32. Gambi, A.; Nguyen, V.; Ahmed, J.; Fraser, G. Generating Critical Driving Scenarios from Accident Sketches. In Proceedings of the 2022 IEEE International Conference on Artificial Intelligence Testing (AITest), Newark, CA, USA, 15–18 August 2022; pp. 95–102. [CrossRef]
33. Abdessalem, R.; Nejati, S.; Briand, L.; Stiffer, T. Testing vision-based control systems using learnable evolutionary algorithms. In Proceedings of the 40th IEEE/ACM International Conference on Software Engineering (ICSE), Gothenburg, Sweden, 27 May–3 June 2018; pp. 1016–1026. [CrossRef]

34. Tang, Y.; Zhou, Y.; Zhang, T.; Wu, F.; Liu, Y.; Wang, G. Systematic Testing of Autonomous Driving Systems Using Map Topology-Based Scenario Classification. In Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 15–19 November 2021; pp. 1342–1346. [CrossRef]
35. Lee, S.; Cha, S.; Lee, D.; Oh, H. Effective white-box testing of deep neural networks with adaptive neuron-selection strategy. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, 18 July 2020; pp. 165–176. [CrossRef]
36. Schultz, A.; Grefenstette, J.; De Jong, K. Adaptive testing of controllers for autonomous vehicles. In Proceedings of the 1992 Symposium on Autonomous Underwater Vehicle Technology, Washington, DC, USA, 2–3 June 1992; pp. 158–164. [CrossRef]
37. Rocklage, E.; Kraft, H.; Karatas, A.; Seewig, J. Automated Scenario Generation for Regression Testing of Autonomous Vehicles. In Proceedings of the 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), Yokohama, Japan, 16–19 October 2017; pp. 476–483. [CrossRef]
38. Fremont, D.; Kim, E.; Pant, Y.; Seshia, S.; Acharya, A. Formal Scenario-Based Testing of Autonomous Vehicles: From Simulation to the Real World. In Proceedings of the 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), Rhodes, Greece, 20–23 September 2020; pp. 1–8. [CrossRef]
39. Tuncali, C.; Pavlic, T.; Fainekos, G. Utilizing S-TaLiRo As an Automatic Test Generation Framework for Autonomous Vehicles. In Proceedings of the 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil, 1–4 November 2016; pp. 1470–1475. [CrossRef]
40. Klischat, M.; Althoff, M. A Multi-Step Approach to Accelerate the Computation of Reachable Sets for Road Vehicles. In Proceedings of the 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), Rhodes, Greece, 20–23 September 2020; pp. 1–7. [CrossRef]
41. Zhong, Z.; Kaiser, G.; Ray, B. Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles. *IEEE Trans. Softw. Eng.* **2022**, *49*, 1860–1875. [CrossRef]
42. Abdessalem, B.; Panichella, A.; Nejati, S.; Briand, L.; Stiffer, T. Testing autonomous cars for feature interaction failures using many-objective search. In Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), Montpellier, France, 3–7 September 2018; pp. 143–154. [CrossRef]
43. Tian, H.; Jiang, Y.; Wu, G.; Yan, J.; Wei, J.; Chen, W.; Li, S. MOSAT: Finding safety violations of autonomous driving systems using multi-objective genetic algorithm. In Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Singapore, 14–18 November 2022; pp. 94–106. [CrossRef]
44. Gambi, A.; Mueller, M.; Fraser, G. Automatically testing self-driving cars with search-based procedural content generation. In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, Beijing, China, 15–19 July 2019; pp. 318–328. [CrossRef]
45. Haq, F.U.; Shin, D.; Briand, L. Efficient online testing for DNN-enabled systems using surrogate-assisted and many-objective optimization. In Proceedings of the 44th International Conference on Software Engineering, Pittsburgh, PA, USA, 25–27 May 2022; pp. 811–822. [CrossRef]
46. Zhu, M.; Wang, Y.; Pu, Z.; Hu, J.; Wang, X.; Ke, R. Safe, efficient, and comfortable velocity control based on reinforcement learning for autonomous driving. *Transp. Res. Part C Emerg. Technol.* **2020**, *117*, 102662. [CrossRef]
47. Tuncali, C.; Fainekos, G.; Prokhorov, D.; Ito, H.; Kapinski, J. Requirements-driven test generation for autonomous vehicles with machine learning components. *IEEE Trans. Intell. Veh.* **2020**, *5*, 265–280. [CrossRef]
48. Zhong, Z.; Tang, Y.; Zhou, Y.; Neves, V.; Liu, Y.; Ray, B. A Survey on Scenario-Based Testing for Automated Driving Systems in High-Fidelity Simulation. *arXiv* **2021**, arXiv:2112.00964. [CrossRef]
49. Luo, Y.; Zhang, X.; Arcaini, P.; Jin, Z.; Zhao, H.; Ishikawa, F.; Wu, R.; Xie, T. Targeting Requirements Violations of Autonomous Driving Systems by Dynamic Evolutionary Search. In Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 15–19 November 2021; pp. 279–291. [CrossRef]
50. Tang, Y.; Zhou, Y.; Wu, F.; Liu, Y.; Sun, J.; Huang, W.; Wang, G. Route coverage testing for autonomous vehicles via map modeling. In Proceedings of the 2021 IEEE International Conference on Robotics Automation(ICRA), Xi'an, China, 30 May–5 June 2021; pp. 11450–11456. [CrossRef]
51. Xia, C.; Huang, S.; Zheng, C.Y.; Zhang, Q.; Wang, Y.; Wei, Y. Modeling and verification method of intersection test scenario for automatic driving. *J. Softw.* **2023**, *34*, 3002–3021. [CrossRef]
52. Murata, T. Petri Nets: Properties, Analysis and Applications. *Proc. IEEE* **1989**, *77*, 541–580. [CrossRef]
53. Li, G.; Li, Y.; Jha, S.; Tsai, T.; Sullivan, M.; Kalbarczyk, Z.; Iyer, R. AV-FUZZER: Finding Safety Violations in Autonomous Driving Systems. In Proceedings of the 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), Coimbra, Portugal, 12–15 October 2020; pp. 25–36. [CrossRef]
54. Feng, S.; Feng, Y.; Sun, H.; Bao, S.; Zhang, Y.; Liu, H. Testing Scenario Library Generation for Connected and Automated Vehicles, Part II: Case Studies. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 5635–5647. [CrossRef]