Simulation-Based Validation for Autonomous Driving Systems

Changwen Li
SKLCS, ISCAS
Univ. of Chinese Academy of Sciences
Beijing, China

Joseph Sifakis
Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG
Grenoble, France

Qiang Wang
Academy of Military Sciences
Beijing, China

Rongjie Yan*
SKLCS, ISCAS
Univ. of Chinese Academy of Sciences
Beijing, China

Jian Zhang
SKLCS, ISCAS
Univ. of Chinese Academy of Sciences
Beijing, China

ABSTRACT

We investigate a rigorous simulation and testing-based validation method for autonomous driving systems that integrates an existing industrial simulator and a formally defined testing environment. The environment includes a scenario generator that drives the simulation process and a monitor that checks at runtime the observed behavior of the system against a set of system properties to be validated. The validation method consists in extracting from the simulator a semantic model of the simulated system including a metric graph, which is a mathematical model of the environment in which the vehicles of the system evolve. The monitor can verify properties formalized in a first-order linear temporal logic and provide diagnostics explaining their non-satisfaction. Instead of exploring the system behavior randomly as many simulators do, we propose a method to systematically generate sets of scenarios that cover potentially risky situations, especially for different types of junctions where specific traffic rules must be respected. We show that the systematic exploration of risky situations has uncovered many flaws in the real simulator that would have been very difficult to discover by a random exploration process.

CCS CONCEPTS

• Software and its engineering → Software verification and validation; **• Computing methodologies → Modeling and simulation**.

KEYWORDS

Autonomous driving systems, Simulation-based validation, Runtime verification, Formal specification, Temporal logic, LGSVL

ACM Reference Format:
Changwen Li, Joseph Sifakis, Qiang Wang, Rongjie Yan, and Jian Zhang. 2023. Simulation-Based Validation for Autonomous Driving Systems. In *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '23), July 17–21, 2023, Seattle, WA, United States.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3597926.3598100

*Corresponding author (yrj@ios.ac.cn)

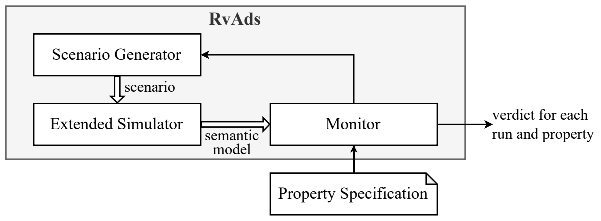


Figure 1: An overview of the proposed RvADS framework

1 INTRODUCTION

Autonomous driving systems (ADS) are real-time distributed systems involving components with partial knowledge of their environment, pursuing specific goals while the collective behavior must meet given global properties. They are probably the most difficult systems to design and validate, as they are built from heterogeneous components subject to temporal and spatial dynamism. They operate in unpredictable environments whose topological and geometric properties constrain the behavior of their agents. These characteristics challenge the application of rigorous model-based development and validation techniques that have been successfully applied to safety-critical systems such as aircraft systems [3]. In particular, formal methods are defeated by the complexity of these systems and can be applied only to their components [5, 24].

In the face of these problems, global validation through simulation and testing appears to be a viable approach to obtain evidence of the trustworthiness of ADS [23, 28].

Nevertheless, there is still a big gap between the state-of-the-art and the needs for rigorous validation of ADS. One challenging problem is how to connect off-the-shelf industrial simulators to validation tools systematically. Currently, industrial simulators are mostly built on top of game platforms in an ad hoc manner. They favor a certain realism of modeling with performance considerations, but lack semantic awareness required for rigorous validation. Semantic awareness implies the possibility to extract from the simulated system software, a semantic model with a well-defined notion of system state as the distribution of vehicles on a map with all their kinetic and time attributes. Such a model should define an abstract system behavior with a notion of execution step and should respect fundamental properties of time and space.

Another challenging problem is to explain how simulated miles relate to "real miles" [13]. A simple simulation, even for an extremely high number of simulated miles or hours, is not sufficient. We need well-founded criteria showing that simulation does indeed cover a large fraction of the relevant real-world situations. In addition, the validation must concern not only incidents, but also the detection of any type of potentially dangerous situation, such as traffic violations.

Fig. 1 presents an overview of the proposed RvADS framework, which integrates an off-the-shelf autonomous driving simulator, in particular the LGSVL Simulator [26], with a Scenario Generator and a Monitor for the rigorous validation of ADS.

The Scenario Generator drives the simulation process. It generates scenarios as the coordinated sequences of actions executed by the agents in the Simulator. Thus, scenarios play the role of test cases intended to drive the simulated system towards specific configurations, for example to explore particular high-risk configurations or to meet specific coverage criteria.

The Simulator runs the system model that consists of both the behavior models of traffic agents (e.g., mobile agents such as vehicles and pedestrians, and non-moving objects such as traffic signals) and the model of the static environment. The latter includes the map and all relevant information regarding the system state. The state of traffic agents consists of their kinematic attributes and their positions on the map. The state of the system model is defined by the positions of the agents and objects in their environment with their states. It determines the possible interactions between agents, objects and their physical environment. The resulting global behavior should satisfy safety and performance properties, in particular those implied by traffic rules.

To link the simulation and the validation processes, we have extended the Simulator in two main directions. The first is to modify the agents' controllers to follow predefined scenarios and explore specific situations. The second one is the development of an API to export an abstract semantic model of the simulated system on which the Monitor can check the properties to be validated.

The Monitor checks whether the runs of the simulated system satisfy given properties expressed as logical formulas. It evaluates the atomic propositions of the formulas based on the semantic model exported through the Simulator API, and applies the runtime verification technique to detect property violations from the observed behavior of the system.

Contribution. First, we provide a rigorous simulation-based validation framework for ADS. Our framework can automatically construct the semantic model from the simulated ADS and adopt a formal runtime verification technique to check against the semantic model the satisfiability of the system properties, e.g., traffic rules, specified in the first-order temporal logic. Second, we propose a concept of structural equivalence and the coverage criteria for scenarios containing junctions. The proposed concept can be used to guide the scenario generation systematically and to explore different types of potentially risky situations. Finally, we have integrated the high-fidelity LGSVL Simulator in our framework. The experimentation considering scenarios with two types of junctions has revealed several deficiencies in the Simulator.

Organization. Section 2 presents the way of connecting simulation to validation and the construction of semantic model of an ADS using the information extracted from the Simulator. Section 3 presents the formalization of system properties and the runtime verification method to validate such formulas against the semantic model. Section 4 presents the rigorous validation methodology using scenarios and the coverage criteria for scenarios containing junctions. Section 5 presents the implementation of RvADS and the experimental results. We conclude with a review of related work and a discussion of the future development.

2 CONNECTING SIMULATION TO VALIDATION

2.1 Map model formalization

The basis of the semantic model is a metric graph [5], defining the set of positions on which the simulated vehicles are located. We first present the HD map model used by the LGSVL simulator and then the transformation into a formal metric graph model.

The maps adopted in the LGSVL simulator involve lanes, junctions, self-reversing lanes, and other features. The annotations of lanes include waypoints and boundary lines. The annotations of intersections include lanes, traffic signals, traffic signs, and stop lines. For ease of discussion, we assume that all the roads and junctions are single-lane, and are equipped with the necessary signals to enforce the traffic rules. Nevertheless, the approach followed can be extended to multi-lane roads without significant changes. Furthermore, we ignore the width of lanes and simply represent them by their center lines. A lane is denoted by $\ell = (l_1, \tau, \lambda, l_2)$, where l_1 and l_2 record the locations of the endpoints of ℓ and the direction of the lane is from l_1 to l_2, τ represents the turn type of the lane (such as `left`, `right`, `straight`), and λ is the label type of the lane (i.e., either in a road or in a junction). For example, in the sub-map of Fig. 2(a), there is a road with lane ℓ_1 and a junction with two lanes ℓ_2 and ℓ_3, positioned with respect to their center-lines. The lane ℓ_1 in the road is denoted by $(l_2, \texttt{straight}, \texttt{road}, l_1)$, and the lane ℓ_3 in the junction is denoted by $(l_4, \texttt{left}, \texttt{junction}, l_2)$.

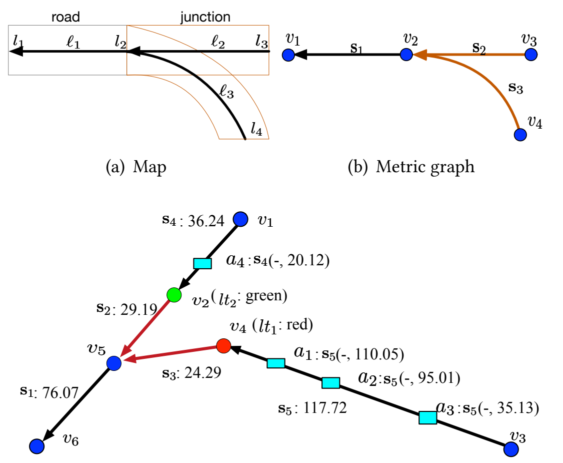
2.1.1 Metric graph We use metric graphs [5] for the formalization of the map model provided by the Simulator. These are directed graphs whose edges are labeled with segments that carry geometric information characterizing the road structure between their endpoints.

A metric graph is formally defined by a tuple $G = (V, \mathcal{S}, E)$, where V is a finite set of vertices, $\mathcal{S}$ is a set of segments with a partial concatenation operator $\cdot : \mathcal{S} \times \mathcal{S} \to \mathcal{S} \cup \{\perp\}$ with $\perp$ for undefined cases, and $E \subseteq V \times \mathcal{S}^+ \times V$ is a set of edges labeled by segments of non-zero length (denoted by $\mathcal{S}^+$). An edge $e = (v, \mathbf{s}, v')$ is denoted by $v \xrightarrow{\mathbf{s}} v'$ for brevity. For example, the metric graph representing the sub-map of Fig. 2(a) is shown in Fig. 2(b). There are three vertices in the graph and three edges labeled with segments $\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{s}_3$ corresponding to the three lanes of the map.

In addition to the vertices, we introduce the concept of *positions* on a segment $\mathbf{s}$ as follows. $\mathbf{s}(-, \eta)$ and $\mathbf{s}(\eta, -)$ are the positions of distance η from the beginning and the end of $\mathbf{s}$, respectively, and $\mathbf{s}[-, \eta]$ and $\mathbf{s}[\eta, -]$ are the segments defined respectively between the beginning of $\mathbf{s}$ and $\mathbf{s}(-, \eta)$, and between $\mathbf{s}(\eta, -)$ and its end.

For example in Fig. 2(c), the position of vehicle a_1 on the metric graph is $\mathbf{s}_5(7.67, -)$. The segment $\mathbf{s}_5[7.67, -]$ is the fragment of $\mathbf{s}_5$ from its end to the position of a_1.

2.1.2 Map to metric graph transformation The transformation from a map of the Simulator to a metric graph is straight-forward. Each location of a lane is regarded as a vertex, and an edge e with its labelling segment $\mathbf{s}$ is created if there is a lane ℓ between two locations. Finally, we associate the attributes of the lanes to the corresponding edges and segments, e.g., the turn type of a lane is associated to that of the segment. The traffic signals and stop signs are indicated at their positions on the metric graph. For example, the four locations in the map model of Fig. 2(a) correspond to the



(a) Map (b) Metric graph

(c) A state of the semantic model

Figure 2: An example map (a), its corresponding metric graph and (b) a state of the semantic model (c)

four vertices of the metric graph in Fig. 2(b). As there are three connections from l_2 to l_1, from l_3 to l_2, and from l_4 to l_2 respectively, we create three edges labeled with segments $\mathbf{s}_1$, $\mathbf{s}_2$ and $\mathbf{s}_3$ in the corresponding metric graph.

2.2 Semantic model of an ADS

In order to simplify the presentation, we regard the mobile traffic participants, such as vehicles and pedestrians as *agents*, and non-mobile traffic participants such as traffic lights and traffic signs as *objects* in the subsequent sections.

The semantic model provides a global view of the dynamics of the simulated system by combining in a coherent way two complementary aspects: on one hand the state of the agents and on the other hand the static environment where they move. We construct the semantic model from the state information of the simulated system exported by the Simulator API.

First, we extract, through the Simulator API, the state information of the agents and objects and build the state of the semantic model. The state of an ADS consists of a map with the states of its agents and objects. The state of an agent includes its position, orientation, velocity and itinerary, where an itinerary is a sequence of consecutive segments in the metric graph that cannot begin or finish in a junction. For example, an itinerary for the agent a_1 in Fig. 2(c) can be $\{\mathbf{s}_5[-, 110.05]\ \mathbf{s}_3\ \mathbf{s}_1[-, 7]\}$, and $\mathbf{s}_5[-, 110.05]$ is the first segment of the itinerary. The state of an object includes its position and for a traffic light additionally its color.

We represent the state of an agent by $s_a = (it, pos, sp, wt)$ where it is an itinerary with it^0 being the first segment in the itinerary, $pos = it^0(-, 0)$ is the position of agent a on the metric graph, sp is its speed, and wt is the waiting time elapsed since its speed became zero. Similarly, the state of a traffic light lt is denoted by $s_{lt} = (pos, color)$, where pos is its position in the metric graph, $color$ can be either red, yellow or green. For a stop sign, its state only records its position in the metric graph. Notice that the generation of the corresponding states of the semantic model requires the computation of the relative positions of agents and objects on the metric graph from their physical 2D Cartesian coordinates.

A run of an ADS is a sequence of states generated periodically by the Simulator. We assume that the states of agents and objects are updated periodically by the simulation process every time Δt. The global state (state for short) at time t for a set of n agents and objects is denoted by $S_t = (s_1, \ldots, s_n)$. The sequence of states in a time interval $[0, m * \Delta t]$ is then denoted by $(S_0, \ldots, S_m)$. Considering again the map model in Fig. 2, a state of a simulated system on the corresponding metric graph is shown in Fig. 2(c). There are one vehicle in segment $\mathbf{s}_4$, three vehicles in segment $\mathbf{s}_5$ and two traffic lights in segment $\mathbf{s}_2$ and $\mathbf{s}_3$ respectively. The length of $\mathbf{s}_5$ is 117.72.

The semantic model of a simulated system is then defined by the set of all the tuples $(G, \mathbb{S})$, where G is the metric graph of the involved map, and $\mathbb{S} = (S_0, \ldots, S_m)$ is a system run extracted from the Simulator in time interval $[0, m * \Delta t]$. Each element of a run describes a global state including the states of all agents and objects.

3 PROPERTY SPECIFICATION AND VALIDATION

In this section, we formalize a set of global system properties describing traffic rules as formulas of the linear temporal logic proposed in [5] and provide a runtime verification method to validate such formulas against the semantic model.

3.1 Property specification

The metric graph $G = (V, \mathcal{S}, E)$ representing a map can be decomposed into two sub-graphs corresponding respectively to its roads and its junctions.

The sub-graph $R = (V_r, \mathcal{S}_r, E_r)$ represents the roads with $E_r \subseteq E$ and $V_r \subseteq V$. Each road is a maximal directed path $r = v_0 \xrightarrow{\mathbf{s}_1} v_1 \cdots \xrightarrow{\mathbf{s}_n} v_n$, where all the vertices $v_1, \ldots, v_{n-1}$ have in-degree and out-degree equal to one. Vertices v_0 and v_n are called the *entrance* and *exit* of the road r, respectively.

The sub-graph $J = (V_J, \mathcal{S}_J, E_J)$ representing the set of junctions is obtained from G by removing from its roads all the vertices except their entrances and exits, with $E_J = E \setminus E_r$ the set of edges labelled with junctions, $V_J \subseteq V$ the vertices of edges in E_J, and $\mathcal{S}_J$ the set of the corresponding segments. We assume that in a junction each entrance v has a neighboring exit v' denoted by v `entex` v'.

Let $\mathcal{A}$ and O denote the set of agents and the set of objects, respectively. The properties of the system are expressed as formulas of a first-order linear temporal logic involving the following types of variables. Let a, o, r, j denote the variables of agents, objects, roads, and junctions, respectively. We denote by $x.pos$ the position of an agent $(x = a)$ or an object $(x = o)$ on the map, and by $y.en, y.ex$, an entrance and an exit of a road $(y = r)$ or junction $(y = j)$. Variables v and e represent a vertex and an edge of the map, respectively. Finally, we introduce three types of predicates on these variables defined as follows.

- v_1 *orientation* v_2, where *orientation* $\in$ {`right-of`, `opposite`} describes the relative position between two vertices v_1 and v_2 in a junction of a map. In particular, predicate v_1 `right-of` v_2 means that v_1 is to the right of v_2, and predicate v_1 `opposite` v_2 means that v_1 and v_2 are the origins of two segments oriented in opposite directions.
- $x@y$ indicates that the position of x is at y, where x is an agent or object variable and y is a sub-graph restricted to a set of vertices.

- $turn(a, d)$ indicates the direction taken by an agent a following its itinerary, where $d \in \{\texttt{left}, \texttt{right}, \texttt{straight}\}$ means respectively that a turns left, right or goes straight.

The semantics of the predicates for elements $(G, \mathbb{S})$ of the semantic model is defined in Table 1. We consider the first-order linear temporal logic generated from the above predicates using the temporal modalities $\mathbf{G}, \mathbf{F}, \mathbf{X}, \mathbf{U}$, i.e., always, eventually, next and until operator, respectively. For instance, the following formula specifies that for any run of the system with m agents and k objects, formula φ should always hold on a system run: $\forall a_1 \cdots \forall a_m . \forall o_1 \cdot \forall o_k . \mathbf{G}\ \varphi(a_1, \ldots, a_m, o_1, \ldots, o_k)$.

To ease the presentation, we also introduce the following predicate that states the agent a is at an entrance of junction $\jmath$ and heading in direction d: $take(a, \jmath.en, d) \overset{\text{def}}{=} [a@\jmath.en \wedge [a@\jmath.en\ \mathbf{U}\ [a@\jmath \wedge turn(a, d)]]]$

We provide traffic rules enforcing priorities of agents approaching junctions and their formal specifications in Table 2. The first three rules are for junctions with stop signs and the other three are for junctions with traffic lights.

3.2 Runtime verification

Given $(G, \mathbb{S})$ an element of the semantic model constructed from the exported states and a formula φ specifying the desired property, we apply runtime verification as shown in Algorithm 1 to check whether $(G, \mathbb{S})$ satisfies the given property.

```
1: Input: a formula $\varphi$ and an element $(G, \mathbb{S})$ of the semantic model
2: Output: verdict showing the verification result
3: $\varphi'$ = unfold($\varphi, G, \mathbb{S}$)
4: $\mathbb{S}^s$ = extract($\mathbb{S}$)
5: $\varphi''$ = simplify($\varphi', G, \mathbb{S}^s$)
6: verdict = check($\varphi'', G, \mathbb{S}$)
```

Algorithm 1: Runtime verification algorithm

The first step applies the `unfold` function (Line 3), which eliminates all the quantifiers of the input property φ and obtains a quantifier-free formula φ' in the following manner.

Let D_κ be the finite domain of a variable κ. Recall that each variable in the formula refers to an element of specific type which can be an agent, object or junction entrance. The domain D_κ of variable κ is a finite set of constants that can be extracted from $(G, \mathbb{S})$ according to the type of κ. For instance, if variable κ refers to elements of type `agent`, its domain D_κ will be simply all the agents of the simulated system.

Quantifier elimination for variable κ boils down to the application of the following two rules that instantiate κ with all the corresponding elements in its domain D_κ, where $\phi[\kappa/\mathbf{c}]$ represents substitution of κ by $\mathbf{c}$ in the formula ϕ.

$$\forall\kappa.\ \varphi \Leftrightarrow \bigwedge_{\mathbf{c} \in D_\kappa} \varphi[\kappa/\mathbf{c}];\ \exists\kappa.\ \varphi \Leftrightarrow \bigvee_{\mathbf{c} \in D_\kappa} \varphi[\kappa/\mathbf{c}]$$

The function `extract` (Line 4) constructs a static model $\mathbb{S}^s$, which contains the state attributes of non-mobile objects that do not change over the simulation, e.g., the positions of traffic lights. This static model can be used to further simplify the formula.

The function `simplify` (Line 5) evaluates the predicates involving non-mobile objects in the quantifier-free formula φ' according to $(G, \mathbb{S}^s)$, and outputs a simplified formula φ''.

Finally, the function `check` (Line 6) evaluates the simplified quantifier-free formula φ'' against $(G, \mathbb{S})$, and returns a verdict indicating the satisfaction or not of the given formula.

For formula evaluation, we adopt the method proposed in [4], that consists in extracting a deterministic finite state machine characterizing the behavior specified by formula φ''. Then the satisfiability check boils down to checking the run $\mathbb{S}$ of $(G, \mathbb{S})$ is accepted by the finite state machine.

4 RIGOROUS VALIDATION USING SCENARIOS

4.1 Structural equivalence for scenarios

Given a set of properties P, we define an equivalence relation $\sim_P$ on system runs. For two system runs $\mathbb{S}_1, \mathbb{S}_2$, we put $\mathbb{S}_1 \sim_P \mathbb{S}_2$ when $\mathbb{S}_1 \models p$ iff $\mathbb{S}_2 \models p$ for each property p of P. That is, the equivalence relation $\sim_P$ does not distinguish between system runs that satisfy exactly the same properties.

To define the structural equivalence for scenarios, we introduce some necessary notations. Recall that a system state can be defined as a tuple $S = (\{s_a\}_{a \in \mathcal{A}}, \{s_o\}_{o \in O})$ consisting of states of the agents and objects. For a given system state S, an abstract scenario is the set of the itineraries of the involved agents of $\mathcal{A}$, and denoted by $as(S) = \{it_a\}_{a \in \mathcal{A}}$. For example, one possible abstract scenario for the set of agents $\{a_1, a_4\}$ in Fig. 2(c) is $\{a_1 : \mathbf{s}_5[-, 110.05]\mathbf{s}_3\mathbf{s}_1[-, 7], a_4 : \mathbf{s}_4[-, 20.12]\mathbf{s}_2\mathbf{s}_1[-, 6]\}$.

Clearly, any state S can be considered as the disjoint union $S = as(S) \oplus cn(S)$ where $cn(S)$ is the context of S including the dynamic attributes of the states. This decomposition is motivated by the need to distinguish between the part of the agent's state that remains unchanged and the part that changes during system evolution.

Now, we define a structural equivalence $\approx_P$ on abstract scenarios parameterized by the set of properties P. For two abstract scenarios as_1, as_2, we say that $as_1 \approx_P as_2$ if $\mathbb{S}_1 \sim_P \mathbb{S}_2$ for any runs $\mathbb{S}_1, \mathbb{S}_2$ of the simulated system from the initial states S_1 and S_2 such that $as(S_1) = as_1, as(S_2) = as_2$ and $cn(S_1) = cn(S_2)$. That is, two equivalent abstract scenarios extended with identical contexts define two initial states from which the simulated system will generate equivalent runs with respect to the properties of P.

In our testing methodology, we assume that the scenario generator extends abstract scenarios by providing each agent its initial state and determining the whole initial execution context.

We focus on abstract scenarios of a system with a map and a fixed set of agents and objects. A key observation is that for most traffic rules, their application depends mainly on the relative positions of the agent itineraries regardless of the agent dynamics. For instance, if the itineraries of two vehicles cross a junction, the traffic rules in Table 2 are applicable independently of their speeds.

The considered abstract scenarios may involve multiple roads and junctions. As the behavior of a vehicle in one junction has little influence on its behavior in subsequent junctions of its itinerary, we focus on testing policies for different types of junctions.

4.2 Coverage criteria for junctions

We discuss below principles for generating abstract scenarios for junctions guided by a coverage criterion exactly as for structural testing of software systems [2]. We show how structural equivalence on abstract scenarios can be computed for the considered two

Table 1: Semantics of the predicates

$(G, \mathbb{S}) \models v_1$ `opposite` v_2	iff	$v_1 \xrightarrow{\mathbf{s}_1} v_3 \wedge v_2 \xrightarrow{\mathbf{s}_2} v_4 \wedge (v_1$ `entex` $v_4) \wedge (v_2$ `entex` $v_3) \wedge \mathbf{s}_1.\tau =$ `straight` $\wedge\ \mathbf{s}_2.\tau =$ `straight` for two vertices v_3 and v_4 in G
$(G, \mathbb{S}) \models v_1$ `right-of` v_2	iff	$v_1 \xrightarrow{\mathbf{s}_1} v_3 \wedge v_2 \xrightarrow{\mathbf{s}_2} v_3 \wedge ((\mathbf{s}_1.\tau =$ `right` $\wedge\ \mathbf{s}_2.\tau =$ `straight`$) \vee (\mathbf{s}_1.\tau =$ `straight` $\wedge\ \mathbf{s}_2.\tau =$ `left`$))$ for a vertex v_3 in G
$(G, \mathbb{S}) \models x@y$	iff	$v_1 \xrightarrow{\mathbf{s}} v_2 \wedge s_x.pos = \mathbf{s}(-, \eta)$ for two vertices v_1, v_2 and a segment $\mathbf{s}$ in y, and a $\eta \in [0, \lVert\mathbf{s}\rVert]$, where s_x is the state of agent or object x in S_0
$(G, \mathbb{S}) \models turn(a, d)$	iff	$s_a.it^0.\tau = d$ where s_a is the state of agent a in S_0

Table 2: Traffic rules and their formal specifications in linear temporal logic

Properties for a stop junction $\jmath$:
p_1: If a vehicle is in the junction, then no other vehicle can be in the junction: $\forall a.\forall a'.\ \mathbf{G}\ [[a@\jmath \wedge a \neq a'] \rightarrow \neg a'@\jmath]$
p_2: If a vehicle arrives at the same time as another vehicle, the vehicle on the right has the right-of-way: $\forall \jmath.en.\forall \jmath.en'.\forall a.\forall a'.\ \mathbf{G}\ [[a@\jmath.en \wedge a'@\jmath.en' \wedge a.wt = a.wt' \wedge \jmath.en$ `right-of` $\jmath.en'] \rightarrow [[\mathbf{X}\ a'@\jmath.en']\ \mathbf{U}\ a@\jmath]]$
p_3: The vehicle that arrives first at the entrance will pass before other vehicles: $\forall \jmath.en.\forall \jmath.en'.\forall a.\forall a'.\mathbf{G}\ [[a@\jmath.en \wedge a'@\jmath.en' \wedge a.wt < a'.wt] \rightarrow [[\mathbf{X}\ a@\jmath.en]\ \mathbf{U}\ a'@\jmath]]$
Properties for a traffic light junction $\jmath$:
p_4: Any vehicle facing a red light must stop until the traffic light turns green, unless the vehicle is turning right. $\forall \jmath.en.\forall a.\forall lt.\ \mathbf{G}\ [[a@\jmath.en \wedge lt@\jmath.en \wedge lt.cl = red \wedge \neg take(a, \jmath.en,$ `right`$)] \rightarrow [a@\jmath.en\ \mathbf{U}\ lt.cl = green]]$
p_5: If a vehicle facing a red light is turning right, then the vehicle should wait until there is no vehicle on the left. $\forall \jmath.en.\forall \jmath.en'.\forall a.\forall a'.\forall lt.\ \mathbf{G}\ [[a@\jmath.en \wedge a'@\jmath.en' \wedge (\jmath.en$ `right-of` $\jmath.en') \wedge lt@\jmath.en \wedge (lt.cl = red) \wedge take(a, \jmath.en,$ `right`$)] \rightarrow [[\mathbf{X}\ a@\jmath.en]\ \mathbf{U}\ a'@\jmath]]$
p_6: If two vehicles arrive at the entrances of a junction opposite each other, the vehicle turning left must give way to the other. $\forall \jmath.en.\forall \jmath.en'.\forall a.\forall a'.\mathbf{G}\ [[a@\jmath.en \wedge a'@\jmath.en' \wedge \jmath.en$ `opposite` $\jmath.en' \wedge take(a, \jmath.en,$ `left`$) \wedge \neg take(a', \jmath.en',$ `left`$)] \rightarrow [[\mathbf{X}\ a@\jmath.en]\ \mathbf{U}\ a'@\jmath]]$

types of junctions and the corresponding properties. We assume, without loss of generality, that the same type of traffic rules are applied to all junction entrances, i.e., regulation either by stop signs or by traffic lights.

Considering a junction with $k + 1$ entrances and corresponding exits, we can encode possible itineraries and the corresponding abstract scenarios for vehicles that are at its entrances in the following manner using a set of k symbolic directions $\{d_1, \ldots, d_k\}$. For a vehicle entering at entrance i, the symbolic direction d_j means that the vehicle is heading to exit $i + j\ (mod\ (k + 1))$. For instance, for a vehicle at entrance 0, its possible itineraries can be represented by the ordered set $\{0_{d_1}, 0_{d_2}, \ldots, 0_{d_k}\}$ heading to exits from 1 to k. Thus if we consider maximal abstract scenarios with one vehicle per entrance of the junction, we will have $k^{(k+1)}$ different abstract scenarios (for each entrance we have k different itineraries and we have $k + 1$ different entrances). The general form of a maximal abstract scenario is $i_{1 d_{j_1}} \cdots i_{(k+1) d_{j_k}}$, where $j_1, j_2, \ldots, j_k$ take values in the set $\{1, 2, \ldots, k\}$, and all the entrance indexes $i_1, \ldots, i_{(k+1)}$ are different.

Note that if the traffic rules do not particularize the inputs of the junction according to their positions, one can easily define structurally equivalent abstract scenarios by simply rotating the inputs and preserving the symbolic directions. This is generally the case when access to a junction is governed by the same traffic rules, such as an all-way stop or a traffic signal controlled junction. Thus, the abstract scenario $i_{1 d_{j_1}} \cdots i_{(k+1) d_{j_k}}$ under these assumptions is structurally equivalent to the one $(i_1 + 1)_{d_{j_1}} \cdots (i_{(k+1)} + 1)_{d_{j_k}}$ where the sum operation on indices is modulo $k + 1$. That is, this

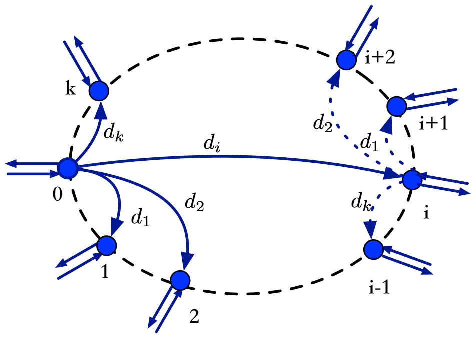


Figure 3: A junction with $k + 1$ entrances and exits

transformation gives structurally equivalent abstract scenarios by simple rotation of the positions of the agents without changing the relative positions of their trajectories.

We consider all possible itineraries (from an entrance to an exit of a junction) to ensure the full structural coverage of a junction with maximal abstract scenarios (one vehicle per entrance).

In Fig. 4 we show how possible itineraries and their abstract scenarios can be generated for a junction with 3 entrances/exits (Fig. 4.(a)). We have $k = 2$, thus we have $2^3 = 8$ abstract scenarios that are shown and grouped in four equivalence classes (as_1: Fig. 4(b), as_2: Fig. 4(c), as_3: Fig. 4(d)-(f), and as_4: Fig. 4(g)-(i)). The abstract scenario $0_{d_1} 1_{d_1} 2_{d_1}$ is alone in its equivalence class as well as the abstract scenario $0_{d_2} 2_{d_2} 1_{d_2}$. The abstract scenario $0_{d_1} 1_{d_1} 2_{d_2}$ is equivalent to two other abstract scenarios: $1_{d_1} 2_{d_1} 0_{d_2}$ and $2_{d_1} 0_{d_1} 1_{d_2}$ depicted in the second row of the figure. Finally, the third row presents three other structurally equivalent abstract scenarios.

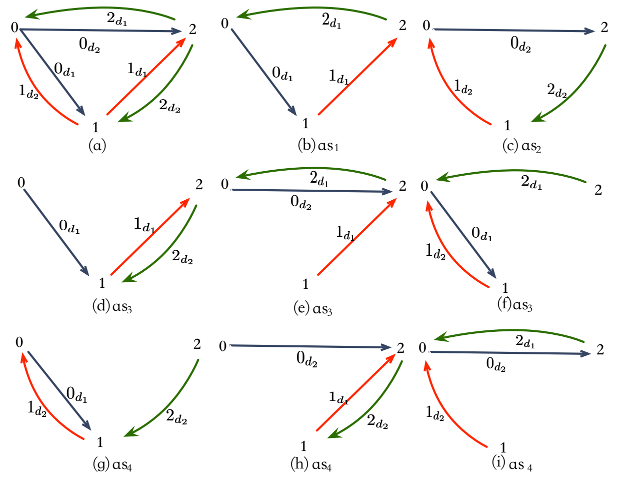


Figure 4: A 3-way junction and its equivalent classes

4.3 Generating concrete scenarios

By following an approach inspired from the metamorphic testing [8], we use equivalent scenarios to test the simulated system with respect to the properties under consideration. Equivalent scenarios are obtained by extending equivalent abstract scenarios with the same dynamic attributes.

Note that given two abstract scenarios for a junction, we can extend them into equivalent abstract scenarios by appending segments that modify corresponding itineraries in the same manner. That is, given two abstract scenarios $as_1 = (it_{11}, \ldots, it_{1n})$ and $as_2 = (it_{21}, \ldots, it_{2n})$, such that $as_1 \approx as_2$, we can extend them by appending road segments $\mathbf{s}_1, \ldots, \mathbf{s}_n$ leading to the entrances of the junction that are the starting points of the itineraries. We thus obtain abstract scenarios $as'_1 = (\mathbf{s}_1 it_{11}, \ldots, \mathbf{s}_n it_{1n})$ and $as'_2 = (\mathbf{s}_1 it_{21}, \ldots, \mathbf{s}_n it_{2n})$ such that $as'_1 \approx as'_2$. We use this simple extension mechanism to generate new equivalence classes of abstract scenarios by simply changing a parameter that is the distance of corresponding itineraries from the entrance of the junction.

For each abstract scenario, we consider one vehicle per direction. The vehicles are initialized at different positions and with different speeds in order to generate concrete scenarios. The positions of the vehicles cover different distances from the entrance of a junction. The speed of the vehicles ranges from zero to the speed limit of the road on which they are located. However, in order to generate realistic scenarios we should choose the initial speeds and distances from the entrance of a junction in such a manner that the vehicle can safely stop before the entrance of the junction if needed. According to the vehicle dynamics, for each distance d, there is a maximal braking speed $v(d)_{max}$ beyond which the vehicle cannot safely stop by braking over the distance d. Thus, we will consider scenarios where the following condition holds: if the distance from the entrance of a junction is d, then the initial speed of the vehicle will be less than or equal to $v(d)_{max}$. In that manner, we exclude the cases where a vehicle may violate a traffic rule for a junction because of excessive speed. Since $v(d)_{max}$ is an increasing function of d, the correspondence between the distance and the maximal braking speed can be estimated by iterative testing of the vehicle dynamics in the Simulator.

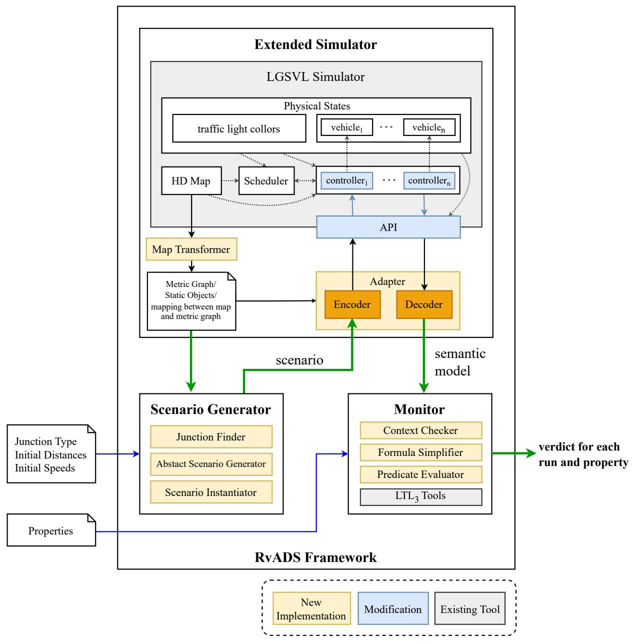


Figure 5: RvADS with the integrated LGSVL Simulator

5 IMPLEMENTATION AND EXPERIMENTATION

5.1 Implementation

We have implemented in the RvADS framework the proposed rigorous testing method to validate autonomous driving systems [1]. RvADS shown in Fig. 5 integrates a Scenario Generator and a Monitor with an ADS simulator. The Scenario Generator produces equivalent scenarios obtained from abstract scenarios that cover all the maximal possible configurations of vehicles from a junction entrance. The Monitor performs runtime verification of the simulated system against a set of properties specifying traffic rules applied at junctions.

RvADS uses the LGSVL Simulator to simulate an ADS. The LGSVL Simulator provides a controller for each vehicle, which is in charge of the speed control, as well as the direction to cross the junction. For speed control, the LGSVL controller evaluates the target speed of the vehicle based on the state of the static and dynamic environment. Additionally, it uses a scheduler taking into account the order of arrival of vehicles at each stop junction and deciding their order of entry in sequence.

In addition to the speed control, the controller randomly decides a direction to follow when a vehicle is approaching a junction. Furthermore, as explained, the original LGSVL Simulator API provides only the physical coordinates of the vehicles, without any explicit connection to the map information. In order to make it compatible with the proposed semantic model construction and validation, we have also extended the LGSVL Simulator. The major modifications are as follows:

- First, we have modified the implementation of the controllers to allow them to follow an itinerary on the HD Map when the vehicle approaches a junction. The itinerary on the HD Map is

[1]Available at https://github.com/LIIHWF/RvADS and [21]

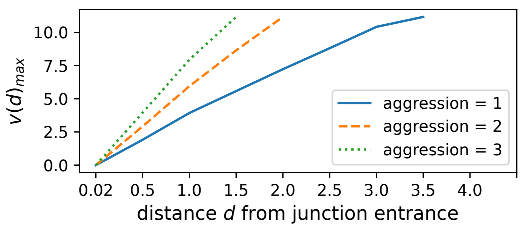


Figure 6: Maximal safe speeds as a function of the braking distance

defined as a sequence of segments using the extended API. In addition, we have established a connection between the controllers and the API such that they can access both the position and the itinerary of a vehicle on the HD map.

- Second, we have developed a Map Transformer from the HD map and static object state information in the Simulator into the corresponding metric graph and object state representation.
- Third, we have developed an Adapter between the Extended Simulator, the Scenario Generator, and the Monitor. Based on the mapping between the HD Map and the metric graph, the encoder in the Adapter transforms an itinerary on the metric graph into an itinerary on the HD Map; the decoder transforms state information of the simulated system into a state of the semantic model.

5.2 Experimental setup

The key question we would like to investigate in the experimentation is how effective RvADS is for the validation of ADS against complex traffic rules and for the systematic exploration of various types of scenarios.

We consider 4-way junctions (as shown in Fig. 7(a)) equipped with stop signs or traffic lights as the static environment of the simulated system. For the stop sign junction, we consider that the stop signs are located exactly at the entrances. A 4-way junction involves 81 abstract scenarios, which are partitioned into 24 structural equivalence classes. For each scenario, we set one vehicle per entrance of the junction.

The simulation period (Δt) of the Simulator is set to 10 ms, which is small enough to accurately approximate the continuous vehicle dynamics. For the simulation of each scenario, the modified LGSVL Simulator can enforce the execution of the given scenario so that each vehicle follows the corresponding itinerary. RvADS checks whether the properties specifying the traffic rules presented in Table 2 are satisfied by the simulated system at runtime. If the semantic model of a scenario is not within the context of the property, then RvADS produces an "NA" verdict. Otherwise if the generated system run satisfies a property, RvADS delivers a "pass" verdict; else it delivers a "fail" verdict.

In Fig. 6, we plot the maximal safe braking speed (in meters/second) as a function of the distance (in meters) from the entrance of a junction. We consider a city map with a speed limit of 11.176 m/s. The three curves correspond to the three speed adjustment rates (i.e., aggression $\in \{1, 2, 3\}$) used by the LGSVL Simulator. For a given distance, the lower the aggression, the lower the safe braking speed. We set aggression = 1 in this experimentation. In generating concrete scenarios, we consider the distance and speed so that the vehicles can stop by braking before the junction.

We remark that in estimating the correspondence between the maximal braking speed and the distance from a stop sign, we have found that a vehicle cannot brake at the stop sign when the distance is less or equal to 0.01 meters to the stop sign, even if the initial speed is zero. The reason is that the controller of the LGSVL Simulator will always apply a speeding-up policy to initialize the vehicle without checking the safe braking distance to the stop sign. That reveals the lack of controllability for small distances in the Simulator that would occur even when simulating a single vehicle.

5.3 Experimentation

RvADS proves to be very effective at testing ADS and allows systematic exploration of high-risk situations whose probability of occurrence is extremely low in random testing. In particular, we have been using RvADS to uncover several deficiencies of the LGSVL Simulator as summarized in Table 5.

5.3.1 Experimentation with a 4-way stop junction A 4-way junction contains 24 structural equivalence classes. In Table 3 we provide testing results for the traffic rules p_1, p_2, and p_3 for 8 abstract scenarios grouped into two structural equivalence classes. The concrete scenarios are then obtained by setting different distances from the entrance (i.e., 0.01 m, 0.3 m and 20 m) and different initial speeds (i.e., 0 m/s and 10 m/s) that respect the safe braking distance relation as shown in Fig. 6. For brevity, we denote a scenario by the index of its abstract scenario and the index of the initial distance and speed configuration. For example, $(1, A)$ includes the scenarios in the first structural equivalence class with 0.01 meters from the stop sign and zero initial speed for each vehicle.

We detail below the causes for each property violation. Experiments show that, for an all-way stop junction, vehicles do not decide autonomously based on knowledge of their surroundings. The Simulator uses a Scheduler to decide which vehicle should move first in case of a conflict. In addition, the Scheduler considers the junction as a different zone than the one defined by its entrances and exits. The use of such a centralized control mechanism goes against the assumption that vehicles are autonomous. Furthermore as discussed below, it is a source of inconsistencies as the Scheduler's perception of the junction differs from that of a vehicle based on its own sensor equipment.

1) Analysis for property p_1

Property p_1: If a vehicle is in the junction, then no other vehicle can be in the junction. Hence, violating p_1 means that more than one vehicle is in the junction at the same time.

This property is violated in all scenarios $(1, A)$, $(2, A)$, $(1, B)$ and $(2, B)$. For scenarios $(1, A)$ and $(2, A)$, the cause is deficiency $\mathbf{I}_1$ in Table 5, due to the lack of controllability for short distances. During initialization, the controller randomly assigns a rate for vehicle acceleration without considering the safe braking distance. When the distance to go is 0.01 meters, the vehicle cannot brake safely.

For scenarios $(1, B)$ and $(2, B)$, the cause is deficiency $\mathbf{I}_2$ in Table 5, due to hidden guidance for control. Vehicle control uses a boundary zone for junctions that is smaller than the area defined by its entrances and exits. For example, in Fig. 7(b), the shaded square area is considered as the junction area by the Scheduler. When the vehicle at entrance 0 leaves the shaded square area, the vehicle at entrance 1 is then scheduled to enter. In such a case, if we consider

Table 3: Experimental results for a 4-way stop junction

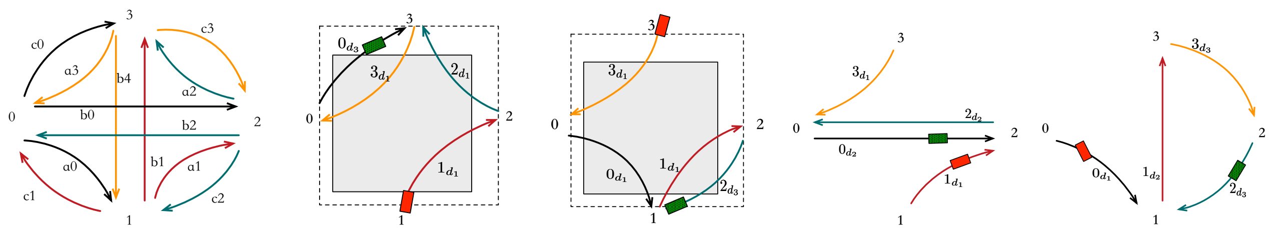
#	class of abstract scenarios	distance (0.01,0.01,0.01,0.01)			distance (0.3,0.3,0.3,0.3)			distance (20,20,20,20)						distance (0.3,0.3,20,20)		
		A: speed (0,0,0,0)			B: speed (0,0,0,0)			C: speed (0,0,0,0)			D: speed (10,10,10,10)			E: speed (0,0,0,0)		
		p_1	p_2	p_3	p_1	p_2	p_3	p_1	p_2	p_3	p_1	p_2	p_3	p_1	p_2	p_3
1	$0_{d_2}1_{d_1}2_{d_1}3_{d_1}$	Fail (0.13s)	Fail (0.61s)	NA (0.75s)	Fail (0.25s)	Fail (0.61s)	NA (0.78s)	Fail (0.30s)	Fail (0.74s)	NA (0.92s)	Fail (0.45s)	Fail (1.09s)	NA (0.98s)	Fail (0.27s)	Fail (0.66s)	Pass (2.37s)
	$0_{d_1}1_{d_2}2_{d_1}3_{d_1}$	Fail (0.13s)	Fail (0.61s)	NA (0.75s)	Fail (0.25s)	Fail (0.61s)	NA (0.78s)	Fail (0.32s)	Fail (0.74s)	NA (0.94s)	Fail (0.30s)	Fail (0.71s)	NA (0.90s)	Fail (0.27s)	Fail (0.63s)	Pass (2.23s)
	$0_{d_1}1_{d_1}2_{d_2}3_{d_1}$	Fail (0.14s)	Fail (0.62s)	NA (0.76s)	Fail (0.25s)	Fail (0.64s)	NA (0.79s)	Pass (0.33s)	Fail (0.74s)	NA (0.92s)	Pass (0.32s)	Fail (0.75s)	NA (0.93s)	Pass (0.31s)	Fail (0.65s)	Pass (2.29s)
	$0_{d_1}1_{d_1}2_{d_1}3_{d_2}$	Fail (0.13s)	Fail (0.62s)	NA (0.75s)	Fail (0.25s)	Fail (0.62s)	NA (0.78s)	Fail (0.32s)	Fail (1.02s)	NA (0.95s)	Fail (0.36s)	Fail (0.82s)	NA (0.99s)	Fail (0.27s)	Fail (0.62s)	Pass (2.33s)
2	$0_{d_3}1_{d_1}2_{d_1}3_{d_1}$	Fail (0.15s)	Fail (0.68s)	NA (0.94s)	Fail (0.27s)	Fail (0.72s)	NA (0.89s)	Fail (0.32s)	Fail (0.82s)	NA (1.03s)	Fail (0.32s)	Fail (0.81s)	NA (1.04s)	Fail (0.28s)	Fail (0.83s)	Pass (2.74s)
	$0_{d_1}1_{d_3}2_{d_1}3_{d_1}$	Fail (0.21s)	Fail (0.69s)	NA (0.86s)	Fail (0.28s)	Fail (0.71s)	NA (0.89s)	Fail (0.35s)	Fail (0.82s)	NA (1.02s)	Fail (0.40s)	Fail (0.90s)	NA (1.07s)	Fail (0.28s)	Fail (0.71s)	Pass (2.54s)
	$0_{d_1}1_{d_1}2_{d_3}3_{d_1}$	Fail (0.14s)	Fail (0.68s)	NA (0.85s)	Fail (0.27s)	Fail (0.71s)	NA (0.88s)	Pass (0.39s)	Fail (0.88s)	NA (1.09s)	Pass (0.36s)	Fail (0.80s)	NA (1.01s)	Fail (0.39s)	Fail (0.76s)	Pass (2.69s)
	$0_{d_1}1_{d_1}2_{d_1}3_{d_3}$	Fail (0.14s)	Fail (0.68s)	NA (0.85s)	Fail (0.26s)	Fail (0.72s)	NA (0.94s)	Fail (0.34s)	Fail (0.82s)	NA (1.03s)	Fail (0.35s)	Fail (0.87s)	NA (1.07s)	Fail (0.32s)	Fail (0.71s)	Pass (2.56s)

Table 4: Experimental results for a 4-way traffic light junction

#	class of abstract scenarios	distance (0.01,0.01,0.01,0.01)			distance (0.3,0.3,0.3,0.3)			distance (20,20,20,20)						distance (20,0.3,20,0.3)		
		F: speed (0,0,0,0)			G: speed (0,0,0,0)			H: speed (0,0,0,0)			I: speed (10,10,10,10)			J: speed (0,0,0,0)		
		p_4	p_5	p_6	p_4	p_5	p_6	p_4	p_5	p_6	p_4	p_5	p_6	p_4	p_5	p_6
3	$0_{d_1}1_{d_2}2_{d_3}3_{d_3}$	Fail (2.02s)	NA (1.88s)	Fail (4.90s)	Pass (2.19s)	NA (1.95s)	Fail (5.10s)	Pass (2.18s)	NA (1.95s)	Fail (5.02s)	Pass (2.08s)	NA (1.85s)	Fail (4.79s)	Pass (2.20s)	NA (1.96s)	Fail (5.10s)
	$0_{d_3}1_{d_1}2_{d_2}3_{d_3}$	Fail (2.02s)	NA (1.87s)	Fail (4.80s)	Pass (2.33s)	NA (1.93s)	Fail (5.08s)	Pass (2.17s)	NA (1.93s)	Fail (4.97s)	Pass (2.10s)	NA (1.85s)	Fail (4.91s)	Pass (2.11s)	NA (1.88s)	Fail (4.87s)
	$0_{d_3}1_{d_3}2_{d_1}3_{d_2}$	Fail (2.10s)	NA (1.80s)	Fail (4.71s)	Pass (2.15s)	NA (1.92s)	Fail (5.00s)	Pass (2.18s)	NA (1.94s)	Fail (5.11s)	Pass (2.19s)	NA (1.95s)	Fail (5.05s)	Pass (2.21s)	NA (1.95s)	Fail (5.09s)
	$0_{d_2}1_{d_3}2_{d_3}3_{d_1}$	Fail (2.04s)	NA (1.77s)	Fail (4.72s)	Pass (2.11s)	NA (1.87s)	Fail (4.90s)	Pass (2.28s)	NA (2.01s)	Fail (5.28s)	Pass (2.12s)	NA (1.88s)	Fail (4.91s)	Pass (2.21s)	NA (1.94s)	Fail (5.05s)
4	$0_{d_2}1_{d_1}2_{d_2}3_{d_1}$	NA (0.36s)	Pass (2.23s)	NA (0.77s)	NA (0.36s)	Pass (2.26s)	NA (0.78s)	NA (0.44s)	NA (0.99s)	NA (0.94s)	NA (0.42s)	NA (0.95s)	NA (0.90s)	NA (0.37s)	Fail (2.29s)	NA (0.79s)
	$0_{d_1}1_{d_2}2_{d_1}3_{d_2}$	NA (0.35s)	Pass (2.18s)	NA (0.75s)	NA (0.36s)	Pass (2.24s)	NA (0.77s)	NA (0.45s)	NA (1.03s)	NA (0.97s)	NA (0.43s)	NA (0.98s)	NA (0.92s)	NA (0.41s)	Fail (2.58s)	NA (0.89s)
	$0_{d_2}1_{d_1}2_{d_2}3_{d_1}$	NA (0.35s)	Pass (2.13s)	NA (0.73s)	NA (0.34s)	Pass (2.09s)	NA (0.71s)	NA (0.44s)	NA (0.98s)	NA (0.94s)	NA (0.42s)	NA (0.97s)	NA (0.92s)	NA (0.38s)	Fail (2.34s)	NA (0.81s)
	$0_{d_1}1_{d_2}2_{d_1}3_{d_2}$	NA (0.36s)	Pass (2.18s)	NA (0.73s)	NA (0.36s)	Pass (2.18s)	NA (0.75s)	NA (0.44s)	NA (0.98s)	NA (0.92s)	NA (0.44s)	NA (0.99s)	NA (0.94s)	NA (0.40s)	Fail (2.40s)	NA (0.84s)

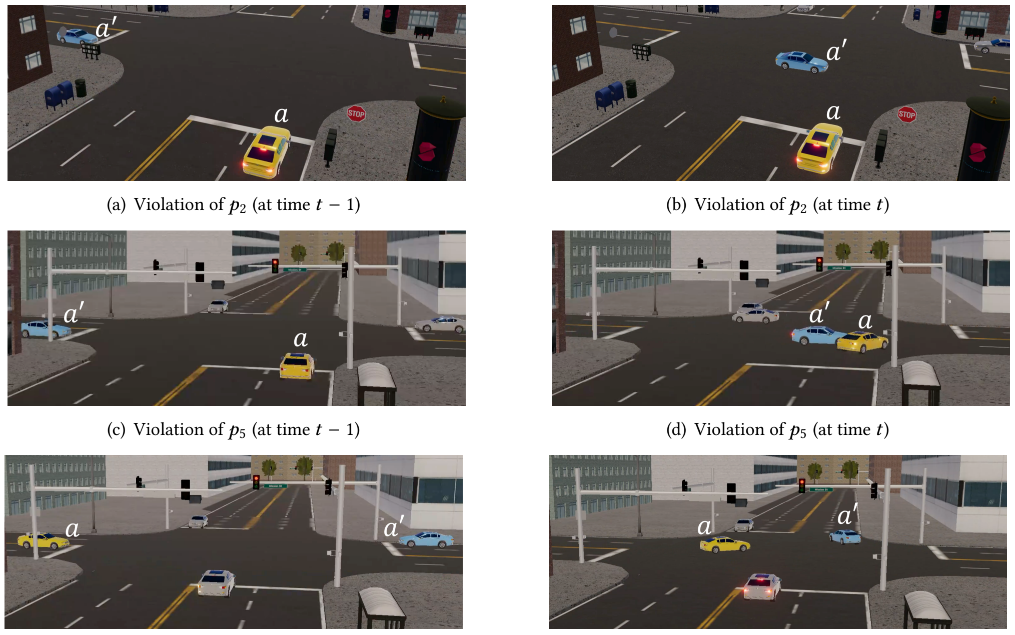
Table 5: Deficiencies discovered in the LGSVL Simulator

Deficiency ID	Explanation
$\mathbf{I}_1$	Lack of controllability for short distances as explained at the end of Section 5.2.
$\mathbf{I}_2$	Hidden guidance for control. Vehicle control uses a boundary zone for junctions that cannot be obtained by sensing the junction environment delineated by entrances/exits and traffic signs.
$\mathbf{I}_3$	No consideration of priorities between different itineraries with the same waiting time. The Scheduler sets priorities according to the creation order of vehicles.
$\mathbf{I}_4$	No consideration of the priorities when turning right at a red light. When a right turning vehicle faces a red light and stops at the entrance of a junction, it does not consider the priority of the vehicle at its left side facing a green light.
$\mathbf{I}_5$	Application of a partial order between the lanes of a junction followed by the vehicles. However, this order is incomplete and leaves unresolved conflicts.



(a) Possible itineraries (b) Violating property p_1 (c) Satisfying property p_1 (d) Violating property p_5 (e) Violating property p_6

Figure 7: Example scenarios of a 4-way junction



(a) Violation of p_2 (at time $t-1$)

(b) Violation of p_2 (at time t)

(c) Violation of p_5 (at time $t-1$)

(d) Violation of p_5 (at time t)

(e) Violation of p_6 (at time $t-1$)

(f) Violation of p_6 (at time t)

Figure 8: Snapshots for property violations

that the junction is delimited by its entrances and exits, the two vehicles are in the junction simultaneously.

We can also observe that even for the same class of scenarios, e.g., $(2, C)$, some scenario violates property p_1, while some do not. For example, there is one scenario in $(2, C)$ satisfying this property. The inconsistency is also due to the use of a boundary zone different from the area of the junction. Moreover, the boundary zone is not placed in the center of the junction, or symmetrical under rotation. Thus, the validity of a property for equivalent scenarios may depend on the configurations between vehicles and the boundary zone of the junction considered by the Scheduler.

For example, the first scenario shown in Fig. 7(b) of $(2, C)$ does not satisfy the property p_1, for the same reason as discussed for scenarios $(1, B)$ and $(2, B)$. However, when considering the case of Fig. 7(c), the third scenario in class $(2, C)$, the vehicle at entrance 2 enters the junction first. As the boundary zone is close to exit 0 and 1, the vehicle at entrance 3 will enter the junction after the one at 2 leaves the junction. The same is true for vehicles from entrances 3 and 0, and from entrances 0 and 1, respectively. Therefore, there is no more than one vehicle in the junction simultaneously, and the property p_1 is satisfied.

2) Analysis for property p_2

Property p_2: If vehicles arrive at the stop sign at the same time, the one on the right has the right-of-way.

Property p_2 is violated in all scenarios. For scenarios $(1, A)$ and $(2, A)$, the cause of violation is explained by deficiency $\mathbf{I}_1$ in Table 5, as the vehicles cannot brake safely and enter the junction.

For scenarios $(1, B)$, $(1, C)$, $(1, D)$, $(2, B)$, $(2, C)$, and $(2, D)$, the cause of violation is explained by deficiency $\mathbf{I}_3$ in Table 5. In these scenarios, the Scheduler ignores the priority rule. The four vehicles reach the stop sign at the same time. According to p_2, a circular priority relationship between the vehicles would result in a deadlock, as each of the four vehicles waits for the vehicle to its right to proceed. In the Simulator however, the Scheduler ignores the rule and schedules the vehicles according to the order of initialization.

For scenarios $(1, E)$ and $(2, E)$, the reason of violation is also explained by deficiency $\mathbf{I}_3$. Consider the first scenario in $(1, E)$ as an example. The vehicles at entrances 0 and 1 reach the stop sign of the junction at the same time. And according to the rule, the vehicle at entrance 1 has a higher priority than the one at 0. However, the Simulator schedules the vehicles according to the initialization order ignoring this rule.

Figs. 8(a) and 8(b) show two successive scenes illustrating a violation. In Fig. 8(a), vehicles a and a' arrive at the same time. Vehicle a' is to the right of a and supposed to have higher priority. However, in the next moment, as shown in Fig. 8(b), vehicle a enters before a'.

3) Analysis for property p_3

Property p_3: The vehicle that arrives at the entrance first will proceed before the other vehicles.

In the scenarios other than those with configuration E, all the vehicles arrive at the entrance simultaneously, so the above property is not applicable and the corresponding results are labelled by NA. For the scenarios in $(1, E)$ and $(2, E)$, property p_3 is satisfied. That is, the Simulator applies this first-in, first-out policy.

5.3.2 Experimentation with a 4-way traffic light junction The results of the validation of the properties of Table 2 for the four-way traffic light junction are shown in Table 4. We choose two structural equivalence classes different from those in Table 3. The initial distances to the entrance and speeds of the vehicles are also chosen from $\{0.01\,\mathrm{m}, 0.3\,\mathrm{m}, 20\,\mathrm{m}\}$ and $\{0\,\mathrm{m/s}, 10\,\mathrm{m/s}\}$, respectively. Experiments show that, for a traffic light junction, the vehicles follow for conflict resolution an implicit order encoded in the lanes of the junction. This is a source of inconsistencies as discussed below.

1) Analysis for property p_4

Property p_4: Every vehicle facing a red light should stop until the traffic light turns green, unless the vehicle is turning right.

In scenarios $(3, F)$, the property is violated. The cause is deficiency $\mathbf{I}_1$ in Table 5. The vehicle facing red light cannot brake safely when it is making a left turn.

In the other scenarios obtained from abstract scenarios of class 3, this property is satisfied, and all the vehicles wait before the traffic lights turn green. However, for scenarios obtained from class 4, this property is not applicable since the two vehicles facing the red light are turning right.

2) Analysis for property p_5

Property p_5: If a vehicle facing a red light is turning right, then the vehicle should wait until there is no vehicle on its left.

The abstract scenario 3 does not involve a vehicle turning right when facing a red light. Therefore, the property is not applicable for this class.

For scenarios in $(4, F)$ and $(4, G)$, the property is satisfied. In these scenarios, the vehicles facing green light will pass through the junction without decelerating. Then the vehicle facing the red light turns right.

For scenarios in $(4, H)$ and $(4, I)$, the vehicles turning right at entrances 1 and 3 are facing red lights and will decelerate first. When they arrive at the entrances, the other vehicles have already entered the junction. Therefore, the property is not applicable.

The scenarios in $(4, J)$ do not satisfy property p_5. As shown in Fig. 7(d), the vehicle facing a red light can enter the junction when there is a vehicle at the entrance on its left. The reason described by deficiency $\mathbf{I}_4$ of Table 5 is that when a vehicle facing green light arrives at the junction, the vehicle making a right turn may have already reached its randomly-assigned waiting time and enters the junction without considering the priority.

We provide a snapshot for such a case in the simulation. In Fig. 8(c), vehicle a is waiting to turn right facing a red light, and vehicle a' arrives. In Fig. 8(d), the traffic light for vehicle a is still red. However, it enters the junction and nearly causes a collision.

3) Analysis for property p_6

Property p_6: If two vehicles arrive at the entrances of a junction from opposite directions and the traffic lights are green, the vehicle turning left must give way to the other.

For class 3, property p_6 is not satisfied. For scenarios $(3, F)$, the reason is deficiency $\mathbf{I}_1$ of Table 5, that is lack of controllability for short distances. The vehicle turning left cannot stop safely and enters the junction before the other.

For the other scenarios from class 3, the reason is deficiency $\mathbf{I}_5$ of Table 5 due to the application of an incomplete priority order. For example, in Fig. 7(e) the vehicle at entrance 0 is turning right, and the vehicle at entrance 2 enters without considering that the one at entrance 0 has a higher priority. However, in the abstract scenario 4, no vehicle turns left. Thus this property is not applicable.

We provide snapshots showing the violation of the property in Figs. 8(e) and 8(f). In Fig. 8(e), vehicles a, a' arrive at the entrances simultaneously, where a is turning left and a' is turning right. According to the traffic rule, a' should proceed first. However, in Fig. 8(f), the two vehicles enter simultaneously.

5.3.3 Runtime overhead of RvADS In addition to the verdict reported by RvADS, we also present time consumption in validating the properties in Tables 3 and 4.

RvADS first checks the applicability of a scenario against a property by evaluating its implicant. If the implicant is not true, RvADS returns NA result without evaluating the whole formula. Therefore, the time costs for NA cases are low.

We can observe that for the same scenario, the time cost increases with the number of temporal operators and variables involved in a property. This is due to the increase in the size of the unfolded formulas.

Moreover, the number of simulation steps for scenarios with the traffic light junction is larger than the one with the stop junction. This fact leads to an increase in the size of the constructed semantic model. Therefore, the time cost of the scenarios with a signalized intersection is higher than that of a stop junction.

6 RELATED WORK

In the literature of simulation-based validation for ADS [19, 23, 34], there is a large body of work on the generation of safety-critical scenarios [7], either by using scenario modeling languages [12, 25], or from available databases [10, 31]. Scenic is a well-known probabilistic programming language for generating scenarios for ADS [12]. Paracosm [25] is another software system that also allows users to describe complex driving situations with specific characteristics and generate scenarios by different parameter configurations. Our approach is based on a semantic model extracted from the Simulator. We generate scenarios based on coverage criteria and test them against properties specified in a logic and verified by verification techniques. Many other works adopt search-based algorithms to generate scenarios challenging the autonomous driving systems. For example, Av-fuzzer [22] utilizes a genetic algorithm-based search to detect situations where an autonomous driving system can run into safety violations. MOSAT [31] uses genes to encode basic driving maneuvers and applies a multi-objective genetic algorithm to search for adversarial and diverse test scenarios.

Table 6: Comparison with existing validation tools for autonomous driving systems

tools	simulation environment	#ego vehicles	tested properties	specification language	test case generation
AsFault [14]	BeamNG	single	lane keeping performance	-	search for road networks
AV-Fuzzer [22]	Apollo+LGSVL	single	collision	-	search for NPC maneuvers
LawBreaker [27]	Apollo+LGSVL	single	traffic laws	STL	specification coverage
RvADS	LGSVL	multiple	traffic laws	parameterized first-order LTL	structural coverage

None of them considers violation of properties or traffic rules as the criteria in identifying safety-critical scenarios.

Efforts are also made to generate scenarios according to map topology. To facilitate the virtual testing of motion planners for automated vehicles, the work in [18] first extracts a large variety of road networks from OpenStreetMap. Then it uses the traffic simulator SUMO to generate traffic scenarios for these road networks. The criticality of the scenarios is reinforced by the use of nonlinear optimization. The works in [29, 30] classify junction lanes according to the collision avoidance maneuvers of the ego vehicle and build map topology-based scenarios with various algorithms. SOCA [6] adopts zone graphs as the abstraction of traffic situations at junctions for behavior analysis, where each zone graph represents an intention of the ego vehicle. Unlike these three works [6, 29, 30], we ignore the detailed topology of the junctions while we consider abstractions that suffice to define coverage criteria for a systematic exploration of traffic patterns.

In addition to generating scenarios, many works use linear temporal logic [5, 11, 16], or signal temporal logic [1, 27, 33], or metric temporal logic [9] to describe traffic rules or safety properties for the validation of ADS. We also adopt linear temporal logic to describe traffic rules. However formulas are parametric involving quantification over both map and vehicle attributes.

Testing techniques have also been widely applied for the validation of ADS [14, 17, 23]. The most related ones are combinatorial testing [20, 25], metamorphic testing [32, 35], fuzz testing [22] and search-based testing [15]. Among all, adopting combinatorial testing can guarantee the coverage of the generated scenarios with respect to the given parameters [20, 25]. By adopting metamorphic relations on the inputs, such as generating scenes with various weather conditions [32], or introducing noise [35], the inconsistency between the outputs can be detected, which eliminates the need to use a test oracle. In fuzz testing, existing scenarios can be mutated to search for potential bugs or safety violations [22]. Search-based testing leverages on optimization to generate concrete test cases [15]. In this work, we use equivalence on scenarios as a metamorphic testing relation. However, we use an Oracle applying runtime verification to detect discrepancies.

Table 6 compares the characteristics of the most recent tools for systematically testing autonomous systems, showing significant differences with RvADS. The criteria considered are the simulation environment, the number of ego vehicles, the type of properties tested, the specification language, and the test case generation method. It should be noted that RvADS, after making appropriate modifications to the Simulator, allows each vehicle to have its own autopilot. On the contrary, the other tools handle a single vehicle with autopilot. The other vehicles follow either a predefined trajectory or interact randomly with the ego vehicle. Thus, the scenarios in RvADS involve the autonomous movement of all the vehicles, allowing realistic critical situations to be created and tested. Furthermore, in RvADS we use a specification language that can deal with configurations between vehicles, and so to describe traffic rules for junctions. This is achieved using an appropriately formalized map on which we can reason and test vehicle configurations.

LawBreaker is the closest tool to RvADS. However, its logic cannot handle vehicle configurations on a map. In addition, LawBreaker's scenario description language is taken from AVUnit[2], which can introduce NPC vehicles that are unclear to what extent they can be controlled to achieve specific global configurations.

7 CONCLUSION

The paper proposes a rigorous simulation-based method for the validation of autonomous driving systems. A key idea is to link the LGSVL Simulator with the RvADS tool that combines a Scenario Generator and a Monitor to test sets of desired system properties specified in a linear temporal logic. The proposed methodology is quite general and can be applied to other Simulators after proper instrumentation.

Unlike simulation-based approaches that focus on quantitative criteria such as the number of hours or kilometers simulated, we rely on the semantic model of the system to achieve sufficient coverage of high-risk situations. We focus on testing traffic rules applicable to two types of junctions where the probability of accidents is increased. The main result is the definition of a property-preserving equivalence on scenarios. We show how we can use equivalent scenarios to discover flaws in the simulated system and ensure fair coverage of risky situations whose probability of occurring in random simulation is very low.

The obtained experimental results reveal several deficiencies in the simulated systems, some of which are related to the driving policies of the agents and some to the simulation runtime implementation. In addition, we found that the deceleration rates applied by the simulated vehicles are unreasonable. For example, the speed of a vehicle in the simulator can be reduced from 10 m/s to 0 m/s in less than 3.5 meters, which is impossible in reality.

The method is novel to our knowledge. It shows that for this reputedly difficult problem, there is a way to tame its complexity. Instead of exploring ordinary situations with low risk potential over long periods of time, we can decompose the validation problem into relatively independent contexts given the locality of the decision making. In a future work, we will show how to apply a compositionality principle to achieve global system validation.

ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their constructive comments. This work has been partly funded by the National Natural Science Foundation of China (Grant No. 62132020), and the CAS Project for Young Scientists in Basic Research (Grant No.YSBR-040).

[2]https://avunit.readthedocs.io/en/latest/

REFERENCES

[1] Nikos Arechiga. 2019. Specifying safety of autonomous vehicles in signal temporal logic. In *IV*. IEEE, 58–63. https://doi.org/10.1109/IVS.2019.8813875

[2] Victor R Basili and Richard W Selby. 1987. Comparing the effectiveness of software testing strategies. *IEEE transactions on software engineering* 12 (1987), 1278–1296. https://doi.org/10.1109/TSE.1987.232881

[3] Ananda Basu, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. 2011. Rigorous component-based system design using the BIP framework. *IEEE Software* 28 (2011), 41–48. https://doi.org/10.1109/MS.2011.27

[4] Andreas Bauer, Martin Leucker, and Christian Schallhart. 2011. Runtime verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology* 20, 4 (2011), 1–64. https://doi.org/10.1145/2000799.2000800

[5] Marius Bozga and Joseph Sifakis. 2021. Specification and validation of autonomous driving systems: A multilevel semantic framework. *CoRR* abs/2109.06478 (2021). https://doi.org/10.1007/978-3-031-22337-2_5

[6] Martin Butz, Christian Heinzemann, Martin Herrmann, Jens Oehlerking, Michael Rittel, Nadja Schalm, and Dirk Ziegenbein. 2020. SOCA: Domain analysis for highly automated driving systems. In *ITSC*. IEEE, 1–6. https://doi.org/10.1109/ITSC45102.2020.9294438

[7] Jinkang Cai, Weiwen Deng, Haoran Guang, Ying Wang, Jiangkun Li, and Juan Ding. 2022. A survey on data-driven scenario generation for automated vehicle testing. *Machines* 10, 11 (2022), 1101. https://doi.org/10.3390/machines10111101

[8] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, TH Tse, and Zhi Quan Zhou. 2018. Metamorphic testing: A review of challenges and opportunities. *ACM Computing Surveys (CSUR)* 51, 1 (2018), 1–27. https://doi.org/10.1145/3143561

[9] Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward J. Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. 2019. VerifAI: A toolkit for the formal design and analysis of artificial intelligence-based systems. In *CAV*. 432–442. https://doi.org/10.1007/978-3-030-25540-4_25

[10] E Esenturk, Siddartha Khastgir, A Wallace, and P Jennings. 2021. Analyzing real-world accidents for test scenario generation for automated vehicles. In *IV*. IEEE, 288–295. https://doi.org/10.1109/IV48863.2021.9576007

[11] Klemens Esterle, Luis Gressenbuch, and Alois Knoll. 2020. Formalizing traffic rules for machine interpretability. In *CAVS*. IEEE, 1–7. https://doi.org/10.1109/CAVS51000.2020.9334599

[12] Daniel J Fremont, Edward Kim, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L Sangiovanni-Vincentelli, and Sanjit A Seshia. 2022. Scenic: A language for scenario specification and data generation. *Machine Learning* (2022), 1–45. https://doi.org/10.1007/s10994-021-06120-5

[13] Daniel J. Fremont, Edward Kim, Yash Vardhan Pant, Sanjit A. Seshia, Atul Acharya, Xantha Bruso, Paul Wells, Steve Lemke, Qiang Lu, and Shalin Mehta. 2020. Formal scenario-based testing of autonomous vehicles: From simulation to the real world. In *ITSC*. 1–8. https://doi.org/10.1109/ITSC45102.2020.9294368

[14] Alessio Gambi, Marc Mueller, and Gordon Fraser. 2019. Automatically testing self-driving cars with search-based procedural content generation. In *ISSTA*. Association for Computing Machinery, 318–328. https://doi.org/10.1145/3293882.3330566

[15] Christoph Gladisch, Thomas Heinz, Christian Heinzemann, Jens Oehlerking, Anne von Vietinghoff, and Tim Pfitzer. 2020. Experience paper: Search-based testing in automated driving control applications. In *ASE*. IEEE Press, 26–37. https://doi.org/10.1109/ASE.2019.00013

[16] Luis Gressenbuch and Matthias Althoff. 2021. Predictive monitoring of traffic rules. In *ITSC*. IEEE, 915–922. https://doi.org/10.1109/ITSC48978.2021.9564432

[17] Wuling Huang, Kunfeng Wang, Yisheng Lv, and Fenghua Zhu. 2016. Autonomous vehicles testing methods review. In *ITSC*. 163–168. https://doi.org/10.1109/ITSC.2016.7795548

[18] Moritz Klischat, Edmond Irani Liu, Fabian Höltke, and Matthias Althoff. 2020. Scenario factory: Creating safety-critical traffic scenarios for automated vehicles. In *ITSC*. 1–7. https://doi.org/10.1109/ITSC45102.2020.9294629

[19] Philip Koopman and Michael Wagner. 2016. Challenges in autonomous vehicle testing and validation. *SAE International Journal of Transportation Safety* 4, 1 (2016), 15–24. http://www.jstor.org/stable/26167741

[20] Changwen Li, Chih-Hong Cheng, Tiantian Sun, Yuhang Chen, and Rongjie Yan. 2022. ComOpT: Combination and optimization for testing autonomous driving systems. In *ICRA*. IEEE, 7738–7744. https://doi.org/10.1109/ICRA46639.2022.9811794

[21] Changwen Li, Joseph Sifakis, Qiang Wang, Rongjie Yan, and Jian Zhang. 2023. *Reproduction Package for 'Simulation-Based Validation for Autonomous Driving Systems'.* https://doi.org/10.5281/zenodo.7825616

[22] G. Li, Y. Li, S. Jha, T. Tsai, and R. Iyer. 2020. AV-FUZZER: Finding safety violations in autonomous driving systems. In *ISSRE*. 25–36. https://doi.org/10.1109/ISSRE5003.2020.00012

[23] Guannan Lou, Yao Deng, Xi Zheng, Mengshi Zhang, and Tianyi Zhang. 2022. Testing of autonomous driving systems: where are we and where should we go?. In *ESEC/FSE*. 31–43. https://doi.org/10.1145/3540250.3549111

[24] Matt Luckcuck, Marie Farrell, Louise A Dennis, Clare Dixon, and Michael Fisher. 2019. Formal specification and verification of autonomous robotic systems: A survey. *ACM Computing Surveys (CSUR)* 52, 5 (2019), 1–41. https://doi.org/10.1145/3342355

[25] Rupak Majumdar, Aman Mathur, Marcus Pirron, Laura Stegner, and Damien Zufferey. 2021. Paracosm: A test framework for autonomous driving simulations. In *FASE*. 172–195. https://doi.org/10.1007/978-3-030-71500-7_9

[26] Guodong Rong, Byung Hyun Shin, Hadi Tabatabaee, Qiang Lu, Steve Lemke, Mārtiņš Možeiko, Eric Boise, Geehoon Uhm, Mark Gerow, Shalin Mehta, et al. 2020. LGSVL simulator: A high fidelity simulator for autonomous driving. In *ITSC*. IEEE, 1–6. https://doi.org/10.1109/ITSC45102.2020.9294422

[27] Yang Sun, Christopher M Poskitt, Jun Sun, Yuqi Chen, and Zijiang Yang. 2022. LawBreaker: An approach for specifying traffic laws and fuzzing autonomous vehicles. In *ASE*. 1–12. https://doi.org/10.1145/3551349.3556897

[28] Shuncheng Tang, Zhenya Zhang, Yi Zhang, Jixiang Zhou, Yan Guo, Shuang Liu, Shengjian Guo, Yan-Fu Li, Lei Ma, Yinxing Xue, et al. 2022. A survey on automated driving system testing: Landscapes and trends. *arXiv preprint arXiv:2206.05961* (2022). https://doi.org/10.48550/arXiv.2206.05961

[29] Yun Tang, Yuan Zhou, Yang Liu, Jun Sun, and Gang Wang. 2021. Collision avoidance testing for autonomous driving systems on complete maps. In *IV*. IEEE, 179–185. https://doi.org/10.1109/IV48863.2021.9575536

[30] Yun Tang, Yuan Zhou, Tianwei Zhang, Fenghua Wu, Yang Liu, and Gang Wang. 2021. Systematic testing of autonomous driving systems using map topology-based scenario classification. In *ASE*. IEEE, 1342–1346. https://doi.org/10.1109/ASE51524.2021.9678735

[31] Haoxiang Tian, Yan Jiang, Guoquan Wu, Jiren Yan, Jun Wei, Wei Chen, Shuo Li, and Dan Ye. 2022. MOSAT: Finding safety violations of autonomous driving systems using multi-objective genetic algorithm. In *ESEC/FSE*. ACM, 94–106. https://doi.org/10.1145/3540250.3549100

[32] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In *ASE*. IEEE, 132–142. https://doi.org/10.1145/3238147.3238187

[33] Ziyuan Zhong, Davis Rempe, Danfei Xu, Yuxiao Chen, Sushant Veer, Tong Che, Baishakhi Ray, and Marco Pavone. 2022. Guided conditional diffusion for controllable traffic simulation. *ArXiv* abs/2210.17366 (2022). https://doi.org/10.48550/arXiv.2210.17366

[34] Ziyuan Zhong, Yun Tang, Yuan Zhou, Vania de Oliveira Neves, Yang Liu, and Baishakhi Ray. 2021. A survey on scenario-based testing for automated driving systems in high-fidelity simulation. *arXiv preprint arXiv:2112.00964* (2021). https://doi.org/10.48550/arXiv.2112.00964

[35] Zhi Quan Zhou and Liqun Sun. 2019. Metamorphic testing of driverless cars. *Commun. ACM* 62, 3 (2019), 61–67. https://doi.org/10.1145/3241979

Received 2023-02-16; accepted 2023-05-03