A Reinforcement Learning Framework with Description Language for Critical Driving Scenario Generation

Shuting Kang, Heng Guo, Yunzhi Xue
Institute of Software Chinese Academy of Sciences
shuting2018@iscas.ac.cn, guoheng21@mails.usca.edu.cn, yunzhi@iscas.ac.cn

Abstract—Critical scenario generation requires the ability of finding critical parameter combinations from the infinite parameter space in the logic scenario. Existing solutions aim to explore the correlation of parameters in the initial scenario without considering the connection between the parameters in the action sequence. How to model action sequences and consider the effects of different action parameter in the scenario remains a key challenge to solve the problem. In this paper, we propose a framework to generate critical scenarios for speeding up evaluating specific tasks. Specifically, we first propose a description language, BTScenario, to model the scenario, which contains the map, actors, interactions between actors, and oracles. We then use reinforcement learning to search for combinations of critical parameters. By adopting the action mask, the effects of non-fixed length and sequences in parameter space can be prevented. We demonstrate that the proposed framework is more efficient than random test and combination test methods in various scenarios.

Keywords—Critical scenario generation, reinforcement learning, scenario description language

I. INTRODUCTION

Despite the great success in Autonomous driving systems (ADS), safety engineering is more challenging for complex and dynamic environments, also known as scenarios. The number of possible test scenarios is virtually infinite, but the critical test scenarios are only a small fraction. It becomes imperative how critical scenarios are generated. With critical scenarios, we refer to situations that cause potential safety risks [1]. Critical scenario generation (CSG) aims to find concrete critical scenarios within a given logical scenario, where the logical scenario is on a state-space level with parameter ranges, while a concrete scenario is a concretization of a logical scenario with concrete variable values [2]. Therefore, how to find critical variable values from infinite parameter space becomes a core problem of CSG.

To advocate research in this direction, native methods (random testing (RT) and combination testing (CT)) sample directly from the parameter range of actors’ actions. This kind of approach is inefficient when critical scenarios are rare. To resolve this problem, recent works [3]–[5] focus on the correlation of actor’s parameters by reinforcement learning (RL). However, they compute the probability distribution of all parameters in the scenario based on RL from the beginning, which is suitable for one action of the environment vehicle. Actually, a scenario can often be modeled as a sequence of atomic action sets, and the parameter relationship between them should also be considered. Therefore, we propose a method for CSG from the perspective of the spatial and temporal distribution in a scenario. Considering the question in Figure 1, we model a complete overtaking scenario with two atomic actions (follow lane and change lane) in five steps. Actually, the sampled value of the first step (follow lane) affect the occurrence of the second step (change lane). The generation of the critical scenario needs to consider the parameter sampling order of the actor. Nevertheless, previous works sample all action parameters simultaneously in the overtaking scenario at the start.

In this paper, we propose a novel RL framework for CSG. The core mechanism behind our framework is to model the action sequence of the scenario, so as to consider the influence relationship between the action parameters. To represent various logical scenarios flexibly, we propose a new scenario description language, named BTScenario, where the representation of action sequences is inspired by composite nodes of the Behavior Tree (“sequence” and “parallel”) [6] using two temporal operators (“serial” and “parallel”). Compared with other description languages, such as scenario Scenic 1.0 / 2.0 [7], Paracosm [9], GeoScenario [10], and OpenScenario 1.0 [11], BTScenario focuses on describing the scenario from the perspective of action interaction. Reinforcement learning is then proposed to sample parameter ranges in logical scenarios from reward feedback at different steps. Besides, we use the action mask to fix the length of action space and the order of action information sharing, which means that the choice of next-step parameters does not affect the action state of the previous step.

The main contributions of this work are as follows:

- We propose a RL framework, which exploits the scenario description language to represent the action sequences in logical scenarios and uses RL combined with the simulator to generate concrete critical scenarios.
- We design a scenario description language, BTScenario, to model logical scenarios. BTScenario supports an abstract and explicit specification of the interactions between actors in chronological order.

1https://www.asam.net/standards/detail/openscenario
2https://www.asam.net/index.php?id=dropFile&f=t&f=3460&token=14e7c7fab9c9b75118bb4939c725738fa0521fe9
We demonstrate the efficiency of our algorithm and provide different test scenarios compared with RT and CT. The good performance can be well explained by focusing on the relationship of parameters between action sequences.

II. RELATED WORK

A. Scenario representation

Scenario representation is the first step in critical scenario generation. Compared with the graphic model (Bayesian probability graph [3], causal graph and behavioral graph [5], temporal logic [11], and Markov decision model [12]–[14]), description languages are more flexible and convenient for users. Let us briefly summarise Scenic 1.0 [7] / 2.0 [8], Paracosm [9], GeoScenario [10], and OpenScenario 1.0 / 2.0 about scenario specification in the sequel.

Scenic 1.0 [7] focuses on the spatial layouts of map objects and the actors but does not allow specifying the temporal actions of actors. Scenic 1.0 was extended to Scenic 2.0 to remedy this deficiency. Scenic 2.0 specifies the actions of actors locally in the sense that actions are specified separately for each actor. We give a simplified example to show this local viewpoint as follows.

ego = Car with behavior EgoBehavior()
pedestrian = Pedestrian with behavior CrossingBehavior()

Paracosm [9] further abstracts the Python API interfaces of the simulator for users to support the temporal action of actors in a way similar to Scenic 2.0. In particular, it assumes a local viewpoint as Scenic 2.0.

Unlike Scenic and Paracosm, GeoScenario [10] and OpenScenario can support cross-platform and cross-programming language. They focus on describing the scenario from a global viewpoint. To put it another way, they specify the scenario through the interaction sequence of actors.

GeoScenario, based on XML, specifies the scenarios by describing the trajectories of actors by a sequence of nodes specified on the map, the triggers for starting the actors, as well as the expected speeds when reaching the nodes in the trajectories.

OpenScenario 1.0 [1] defines the storyboard to describe the temporal actions of actors and events to trigger the start or end condition of the scenario. Nevertheless, the XML-based scenario description language exhibits complex formats and poor readability. In view of this, OpenScenario 2.0 [2] has recently been proposed as an extension of OpenScenario 1.0.

OpenScenario 2.0 defines the position modifiers used to specify the interactions between actors and introduces the serial, parallel, and repeat operators to compose actors’ actions. For instance, we use a lane change scenario to show interactions between actors.

B. Critical Scenarios Generating

Given a logical scenario, we focus on the problem that a set of concrete critical scenarios are generated with a parameter space exploration method. A native scenario exploration approach is assigning each parameter’s value in the logical scenario space. The CT [15] aims to generate a minimum set of test cases that satisfy N-wise coverage [16]. The CT can be used to find unknown combinations of parameters that may fail.
the ADS. Both continuous and discrete variables are prevalent in the parameter space of the logical scenario, but N-wise coverage can only be defined in discrete space. A common method to handle continuous parameters is discretization. However, these native approaches (RT and CT) can be inefficient because of a low probability of critical scenarios.

For the above challenge, the guided search methods have the potential to be more efficient, since the searching direction at each iteration is adjusted, so as to converge the exploration to critical regions. Considering the correlation between parameters, most of the recent works sample critical parameters from a joint probability distribution of actor parameters by RL [3], [4] in a logical scenario. Another solution [5] introduces human prior knowledge by the causal graph and then represents the interaction by the behavioral graph. However, all of these methods mainly consider the correlation of all actor parameters from the beginning. In this paper, to consider the correlation of parameters in the action sequence, we first represent a logical scenario as a collection of action sequences modeled by BTScenario, then exploit RL to sample parameters for CSG.

III. METHOD

Given a logic scenario representation, the CSG aims to find many concrete critical scenarios, which are collided scenarios in this paper. The key is to find the critical parameter combination in actions’ parameters space, i.e., \( A = A_1, A_2, \ldots, A_T \), where \( A \) is the set of actions’ parameters of a scenario, \( T \) is the number of steps in a scenario, and \( A_t \) contains the parameters of \( N \) actions’ parameters \( a_{t1}, a_{t2}, \ldots, a_{tN} \) in the step \( t \).

Figure 2 gives a detailed illustration of our model. We first introduce a new scenario description language, BTScenario, including the “map”, “init”, “execute”, and “oracle” modules. Then we perform the scenario generation model that consists of two parts: the RL algorithm aims to explore the critical parameters, and the simulator aims to simulate the physical environment and returns the reward. By stacking the above two processes multiple times, our model learns the policy and target networks and results in the optimal parameters combination.

A. BTScenario

To model the interaction of actions at different steps clearly, we design BTScenario, inspired by the Behavior Tree which has been widely used in robotics to describe robot actions [17]. Although Various researchers have formulated scenarios in different ways [18]–[23], there do exist some common components, map, actor, interactions between actors, and oracle. BTScenario also provides abundant map elements to show the driving area (III-A1), the specification of the initial states (III-A2), interaction (III-A3), and the test oracles (III-A4). In this subsection, we illustrate these main concepts of the BTScenario through examples.

1) The Map: To intuitively specify the map area for the scenario, we design more abundant map objects with properties. Referring to the Apollo map structure [24], BTScenario supports eleven map objects (e.g., junction, road, lane, etc.), where several roads can be attached to a junction, and a road can have several lanes.

The following example shows how map objects are specified in BTScenario: A junction named “crossroad” is declared as “+” and its first road consists of four lanes. Note that a junction can also be of “T”, “X”, “Y”, or “unknown” type, where the “unknown” type denotes the junctions not of a fixed shape, i.e., not of type “+”, “T”, “X”, or “Y”.

```
Junction crossroad with
type == "+",
crossroad.roads [1].lane_total_number == 4;
```

2) Initial States Of Actors: In BTScenario, we consider various types of actors: Aut_Car representing the car under test, Car representing the other cars, Pedestrian representing pedestrians. We distinguish the tested car from other cars that are considered its surroundings. For instance, the following code declares “ego” as the car under test and “car” as the environment car.

```
Aut_Car ego;
Car car;
```

Moreover, we can specify some additional constraints when declaring an actor. For instance, the following code specifies a car with a red color.
Car car with color == "red";

An important attribute of actors is their positions. BTSce-

nario supports the declaration of both absolute positions and

relative positions. For instance, the following code states that

“car1” is at a randomly chosen position of “lane1”, “car2”
is at coordinate point (1,2), and “car3” is 100 meters ahead
of “car1”, with a clockwise 180 degree angle relative to the
direction of “car1” (i.e., in the opposite direction as “car1”).

Car car1 with absolute_position == lane1;
Car car2 with absolute_position == 1@2;
Car car3 with relative_to == car1, angle == 180,
front_distance == 100;

3) Temporal Actions of Actors: Before introducing the
temporal actions of actors, we first show the specification of
the atomic actions. For cars, the atomic actions include
followLane and changeLane, etc. The followLane describes the
action of following the current lane, and the changeLane de-
dscribes the action of changing a lane to the other. BTScenario
supports the two types of parameter expressions of actions. For
example, the code below specifies that the driving distance of
“car1” can be defined as the default value, 100 meters, or the
interval modified.

car1 . followLane (distance = 100);
car1 . followLane (distance = [0:1000]);

Moreover, actions can have pre-conditions and post-
conditions that specify the enabling and terminating conditions
of the action. For instance, the following code specifies that
if the distance between “car1” and “car2” is less than 100
meters, then “car1” follows the current lane until the distance
is more than 300 meters.

\[( \text{distance(car1,car2)} < 100) \land \text{car1 . followLane} (\text{targetSpeed=60}) \land \text{distance(car1,car2)} > 300\];

Inspired by the Behavior Tree, we take the concepts of
its composite nodes (e.g., sequence and parallel) to define
temporal operators serial and parallel. BTScenario specifies the
temporal actions of actors as compositions of the atomic
actions by two temporal operators, where the serial and
parallel means the sequential and parallel composition of
actions respectively.

For instance, the following code specifies a complete sce-
nario that “car1” and “car2” first follow the current lane in
parallel, then “car1” changes to the left lane and “car2” keeps
straight.

serial(){
parallel(){
car1 . followLane (targetSpeed=[10,20],
distance=[30,50]);
car2 . followLane (targetSpeed=[10,20],
distance=[30,50]);
}
parallel{
car1 . changeLane (direction="left",
targetSpeed = [25,30]);
car2 . followLane (targetSpeed=20, distance= 30);
}
}

4) The Oracle: To specify the test standards for critical
scenarios, there are two kinds of operators in oracle module
of BTScenario: “periodic” and “record”. The “periodic” type
indicates that the oracle needs to be verified unless it is
violated or the simulation is completed. The “record” type
indicates that the program must record information about the
actors or environment in the scenario without impacting the
simulation process. For example, the code below specifies that
the simulator monitors whether “car1” collides and records the
change of “car2” speed value.

periodic: car1 . isCollided () ;
record: car2 . speed ;

Through the above language design, BTScenario can model
the action parameters space of the actors at different steps in
the scenario. Moreover, using BTScenario, we can describe a
variety of complex scenarios.

B. Scenario Generation Model

We divide a scenario S into T steps according to the
temporal relationship of actions. At each discrete step t, with
a given collision state \( s \in S \), the actors select the concrete
actions’ parameters \( A_t \) to generate the concrete scenario. The
CSG process is similar to the Markov decision process [25] in
that the future state depends not only on the current state but
also on the action parameters taken by the actors in the current
state. Thus we use RL to explore critical scenarios parameter
combinations between different steps.

In RL, we regard the environment vehicle as the agent and
the full-stack autonomous driving algorithm to be evaluated
as an environment. The state of the environment has two
parts. The first part contains a road map and information
on the route, which is the reference trajectory for the task
algorithm. The second part consists of the actions’ parameters
of the environment vehicle, which influence the ego’s decision-
making ability.

Next, we introduce the reward design, a trick on parameters
of actions, and the optimization process.

1) Reward Design: The reward function consists of two
parts,

\[ R = -\min_{i=1,...,m} r_d(\rho_{ego}, \rho_{car_i}) + r_c \] (1)

where \( \rho_{ego} \) and \( \rho_{car_i} \) represent the positions of the ego and
the environment cars, respectively. The first part is the risk
metric \( r_d \), which we choose the minimum distance between
the ego and environment cars. The \( r_d \) is

\[ r_d = ||\rho_{ego} - \rho_{car_i}||_2 \] (2)

When a collision occurs, the distance between cars becomes
zero. Therefore, we provide an extra bonus \( r_c \):

\[ r_c = \begin{cases} C & \text{collision} = \text{True} \\ 0 & \text{collision} = \text{False} \end{cases} \] (3)

2) Action Mask: In the implementation, since the action
parameter range may differ in each step of the scenario, and
theoretically, the parameter values selection after the t step
cannot affect the parameter values selection in the $t$ step, we need to process these two problems by two steps. First, to solve the non-fixed-length action spaces, we calculate the max length for the action space $A$ modeled by the BTScenario. We pad the action space as the input of the actor neural network.

Second, we do not want the model to share any information regarding the action parameters in the following steps when giving a prediction using all the previous action parameters. Inspired by the Transformer model [26] in the field of natural language processing, we mask action parameters (setting them to inf) in the future steps.

3) Optimization Process: TD3 [27] is one of RL algorithms. Since TD3 can be used to handle continuous action parameter spaces and is insensitive to hyper-parameter settings, we follow the TD3 to solve our optimization problem. In RL, the objective is to find the optimal policy $\pi_\phi$, with parameters $\phi$, which maximizes the expected return $J(\phi)$. The optimal policy $\pi_\phi$ in our model is the optimal parameters of action $A$ that caused the collision. For the continuous action space, $\pi_\phi$ can be updated by the gradient of the expected return $\nabla_\phi J(\phi)$. In the TD3 method, the policy, known as the actor, can be updated by the deterministic policy gradient [28]. The gradient for updating actor networks parameters $\phi$ is:

$$\nabla_\phi J(\phi) = E_{s \sim \rho}(\nabla_A Q_\theta(s,a)|a=\pi_\phi(s)) \nabla_\phi \pi_\phi(s)$$  \hspace{1cm} (4)$$

where $Q_\theta(s,a)$ is the action-value function, and the parameter values of actions $a \in A$ are generated from an actor network. The critic network is updated by temporal difference learning to maintain a fixed objective $y = r + \gamma \min_a Q_\theta(s',a)$ over multiple updates. The gradient for updating critic networks parameters $\theta$ is:

$$\theta_i = \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s,a))^2$$  \hspace{1cm} (5)$$

The entire algorithm is shown in Alg.1.

### IV. Experiment

#### A. Experiment Settings

1) Implementation Details: The map data of program is a virtual town on the plain, which is Apollo format. In Implementation, we connect Carla as the back-end simulator and Unreal Engine4 as the engine. We apply Lark to compile the language, and take PyTrees to construct and execute the behavior tree. We use the feed-forward PID speed control algorithm [29] and Stanley direction control algorithm [30] for ego. Other details about the hyper-parameter in the Alg. 1 are listed in Table IV-A1.

2) Evaluation Metric: We follow the metrics of Ding to evaluate the performance, the collision rate after a static policy of the model and the number of iterations required for the model to reach stability.

#### B. Verification Experiment

We give an overtaking scenario to show the generation of risky scenarios in Figure 1. We first specify the logic scenario by BTScenario as follows:

```python
scenario overtaking()
    {map()
        Road road with "two_lane_two_way";
    }
    init{
        Car blue_car with color="blue";
        Aut_Car ego with relative_to==blue_car, angle ==0, front_distance == 20;
    }
    execute{
        serial(){
            blue_car, followLane (targetSpeed=[25:45], scale = [20:50]) [distance (ego, blue_car) <[5:10]]; 
            blue_car, changeLane (direction="left", scale = [5:15], targetSpeed=[30:40]);
            blue_car, followLane (targetSpeed=[30:40], scale = [50:80]) [distance (blue_car, ego) >[5:10] ;
            blue_car, changeLane (direction="right", scale = [6:9], targetSpeed=[10:20]);
            blue_car, followLane (scale=[10:20], targetSpeed=[25:30]);
        }
    }
    oracle{
        periodic: ego.isCollided() || blue_car.isCollided()
    }
}
```

When the scenario initializes, the ego is 60 meters in front of the environment car (“blue_car”). Although we can filter out many similar map areas in the map file, we choose a road that meets the conditions as the test area in order to reduce the influence of other factors such as the length of the road. In the “oracle” module, we consider the critical scenario referring to the ego or blue car crashing.

#### C. Comparison Experiment

Figure 3 shows the comparison of our method with other methods.

- Random Sampling: We sample all actions’ values from the specifications of actions by BTScenario. This method

4https://docs.unrealengine.com/en-US/index.html
5https://lark-parser.readthedocs.io/en/latest/grammar.html
6https://py-trees.readthedocs.io/en/release-2.1.x/index.html

### TABLE I

| Hyper-parameter | Description | Value |
|-----------------|-------------|-------|
| $E$             | max epoch number |       |
| $\sigma$        | variance of the noise | 1     |
| $N$             | batch size |       |
| $C$             | collision reward |       |
| $\tau$          | delayed update parameters |       |
| $\gamma$        | threshold in $Q$ |       |
| $c$             | range of action noise |       |
| $C'$            | collision reward |       |
| $B$             | size of replay buffer |       |
| $d$             | delayed steps for updating network |       |
may find critical scenarios but does not consider the influence of the combination of the action parameters.

- **Pairwise Testing**: The core is to cover the pairwise discrete combinations of multiple factor values with the fewest test cases. To reduce the searching expense, we discrete each action range, taking values every two numbers, by the PICT tool.

- **Ablation of Masking**: We evaluate the contribution of the action mask for our method in this experiment.

Our model consistently outperforms all the approaches on the metrics and achieves XXX boost on the collision rate compared with random search methods (RT and CT). Since the critical scenario is at the tail of the long-tailed distribution, random search methods are inefficient. By considering the temporal relationship of parameters, our model gains remarkable improvement.

To verify the contribution of the action mask, we show the ablation results in Table 2. We observe that the collision rate and iteration number decreases by XXX and XXX compared with the entire model, which indicates that the action mask is beneficial for focusing on the current and previous information for critical scenario generation.

### D. Experiment On Other Scenarios

We test two other scenarios to verify the effectiveness of our model. The generated critical scenarios with our framework are shown in Figure 3.

- **Vehicle Running Red Light**: The ego vehicle turns right through an intersection, while the opposite blue vehicle turns left before the red car goes straight by running a red light.
- **Vehicle Right Turn**: The ego goes straight through an intersection, while the blue vehicle accelerates and then turns right.

### V. Conclusion

In this paper, we propose a RL framework for CSG, which focuses on the correlation between parameters of the action sequence in a logical scenario. We model the action sequence by our description language, BTScenario, with temporal operators. We adopt RL with the action mask to sample action sequence in a logical scenario. We model the action which focuses on the correlation between parameters of the metrics and achieves XXX boost on the collision rate compared with random search methods (RT and CT). Since the critical scenario is at the tail of the long-tailed distribution, random search methods are inefficient. By considering the temporal relationship of parameters, our model gains remarkable improvement.

To verify the contribution of the action mask, we show the ablation results in Table 2. We observe that the collision rate and iteration number decreases by XXX and XXX compared with the entire model, which indicates that the action mask is beneficial for focusing on the current and previous information for critical scenario generation.

**REFERENCES**

[1] X. Zhang, J. Tao, K. Tan, M. Törmänen, J. M. G. Sánchez, M. R. Ramli, X. Tao, M. Gyllenhammar, F. Wotawa, N. Mohan et al., “Finding critical scenarios for automated driving systems: A systematic literature review,” arXiv preprint arXiv:2110.08864, 2021.

[2] T. Menzel, G. Bagschik, and M. Maurer, “Scenarios for development, test and validation of automated vehicles,” in 2018 IEEE Intelligent Vehicles Symposium (IV). IEEE, 2018, pp. 1821–1827.

[3] W. Ding, B. Chen, M. Xu, and D. Zhao, “Learning to collide: An adaptive safety-critical scenarios generating method,” in 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2020, pp. 2243–2250.

[4] W. Ding, B. Chen, B. Li, K. J. Eun, and D. Zhao, “Multimodal safety-critical scenarios generation for decision-making algorithms evaluation,” IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 1551–1558, 2021.

[5] W. Ding, H. Lin, B. Li, and D. Zhao, “Causalaf: Causal autoregressive flow for goal-directed safety-critical scenes generation,” arXiv preprint arXiv:2110.13939, 2021.

[6] F. Martín, M. Morelli, H. Espinoza, F. J. Lera, and V. Matellán, “Optimized execution of pdpl plans using behavior trees,” arXiv preprint arXiv:2101.01964, 2021.

[7] D. J. Fremont, T. Dreesi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, “Scenic: a language for scenario specifications and scene generation,” in Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2019, pp. 63–78.

[8] D. J. Fremont, E. Kim, T. Dreesi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, “Scenic: A language for scenario specification and data generation,” arXiv preprint arXiv:2010.06530, 2020.

[9] R. Majumdar, A. Mathur, M. Piron, L. Stegner, and D. Zufferey, “Paracosim: A test framework for autonomous driving simulations,” in Fundamental Approaches to Software Engineering, E. Herrera and M. Stoelinga, Eds. Springer International Publishing, 2021, pp. 172–195.

[10] R. Queiroz, T. Berger, and K. Czarnecki, “Geoscenario: An open dsl for autonomous driving scenario representation,” in 2019 IEEE Intelligent Vehicles Symposium (IV), 2019.

[11] C. Gladisch, T. Heinz, C. Heinzemann, J. Oehlerking, A. von Visteinhoff, and T. Pfitzer, “Experience paper: Search-based testing in automated driving control applications,” in 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2019, pp. 26–37.

[12] A. Wachi, “Failure-scenario maker for rule-based agent using multi-agent adversarial reinforcement learning and its application to autonomous driving,” arXiv preprint arXiv:1903.10654, 2019.

[13] M. Koren and M. J. Kochenderfer, “Efficient autonomy validation in simulation with adaptive stress testing,” in 2019 IEEE Intelligent Transportation Systems Conference (ITSC). IEEE, 2019, pp. 4178–4183.

[14] M. Koren, S. Alsaiif, R. Lee, and M. J. Kochenderfer, “Adaptive stress testing for autonomous vehicles,” in 2018 IEEE Intelligent Vehicles Symposium (IV). IEEE, 2018, pp. 1–7.

[15] D. R. Kuhn, R. Bryce, F. Duan, L. S. Ghandehari, Y. Lei, and R. N. Kacker, “Combinatorial testing: Theory and practice,” Advances in Computers, vol. 99, pp. 1–66, 2015.

[16] E.-H. Choi, T. Kitamura, C. Artho, and Y. Oiwa, “Design of prioritized n-wise testing,” in IFIP International Conference on Testing Software and Systems. Springer, 2014, pp. 186–191.

[17] M. Iovino, E. Scukins, J. Styrud, P. ¨Ogren, and C. Smith, “A survey of behavior trees in robotics and ai,” 2020.

[18] K. Go and J. M. Carroll, “The blind men and the elephant: Views of scenario-based system design,” Interactions, vol. 11, no. 6, pp. 44–53, 2004.

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

[20] S. Geyer, M. Baitzer, B. Franz, S. Hakuli, M. Kauer, M. Kienle, S. Meier, T. Wöllkerber, K. Bengler, R. Bruder et al., “Concept and development of a unified ontology for generating test and use-case catalogues for assisted and automated vehicle guidance,” IET Intelligent Transport Systems, vol. 8, no. 3, pp. 183–189, 2013.
[21] M. Jarke, X. T. Bui, and J. M. Carroll, “Scenario management: An interdisciplinary approach,” Requirements Engineering, vol. 3, no. 3-4, pp. 155–173, 1998.

[22] J. Bock, R. Krajewski, L. Eckstein, J. Klimke, J. Sauerbier, and A. Zlocki, “Data basis for scenario-based validation of had on highways,” in 27th Aachen colloquium automobile and engine technology, 2018.

[23] M. Scholtes, L. Westhofen, L. R. Turner, K. Lotto, M. Schuldes, H. Weber, N. Wagener, C. Neurohr, M. Bollmann, F. Körtke, J. Hiller, M. Hoss, J. Bock, and L. Eckstein, “6-layer model for a structured description and categorization of urban traffic and environment,” IEEE Access, vol. 9, pp. 59 131–59 147, 2021.

[24] J. Xu, Q. Luo, K. Xu, X. Xiao, S. Yu, J. Hu, J. Miao, and J. Wang, “An automated learning-based procedure for large-scale vehicle dynamics modeling on baidu apollo platform,” in 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019, pp. 5049–5056.

[25] A. Abakuks, “Reviewed work: Markov processes: characterization and convergence. by sn ethier, tg kurtz,” Biometrics, vol. 43, no. 2, pp. 113–122, 1987.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, “Attention is all you need,” in Advances in Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017.

[27] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in International conference on machine learning. PMLR, 2018, pp. 1587–1596.

[28] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, “Deterministic policy gradient algorithms,” in International conference on machine learning. PMLR, 2014, pp. 387–395.

[29] F. Gene, J. D. Powell, A. Emami-Naeini, R. Braun, and J. Flatley, “Feedback control of dynamic systems (7th global edition),” 2015.

[30] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, “Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing,” in 2007 American Control Conference. IEEE, 2007, pp. 2296–2301.