Using Formal Conformance Testing to Generate Scenarios for Autonomous Vehicles

Jean-Baptiste Horel
Univ. Grenoble Alpes
Inria
38000 Grenoble, France
jean-baptiste.horel@inria.fr

Christian Laugier
Univ. Grenoble Alpes
Inria
38000 Grenoble, France
christian.laugier@inria.fr

Lina Marsso
Dept. of Computer Science
University of Toronto
Toronto, Canada
lina.marsso@utoronto.ca

Radu Mateescu
Univ. Grenoble Alpes
Inria, CNRS, Grenoble INP, LIG*
38000 Grenoble, France
radu.mateescu@inria.fr

Lucie Muller
Univ. Grenoble Alpes
Inria, CNRS, Grenoble INP, LIG*
38000 Grenoble, France
lucie.muller@inria.fr

Anshul Paigwar
Univ. Grenoble Alpes
Inria
38000 Grenoble, France
anshul.paigwar@inria.fr

Alessandro Renzaglia
Univ. Grenoble Alpes
Inria
38000 Grenoble, France
alessandro.renzaglia@inria.fr

Wendelin Serwe
Univ. Grenoble Alpes
Inria, CNRS, Grenoble INP, LIG*
38000 Grenoble, France
wendelin.serwe@inria.fr

***Abstract*—Simulation, a common practice to evaluate autonomous vehicles, requires to specify realistic scenarios, in particular critical ones, occurring rarely and potentially dangerous to reproduce on the road. Such scenarios may be either generated randomly, or specified manually. Randomly generating scenarios is easy, but their relevance might be difficult to assess. Manually specified scenarios can focus on a given feature, but their design might be difficult and time-consuming, especially to achieve satisfactory coverage. In this work, we propose an automatic approach to generate a large number of relevant critical scenarios for autonomous driving simulators. The approach is based on the generation of behavioral conformance tests from a formal model (specifying the ground truth configuration with the range of vehicle behaviors) and a test purpose (specifying the critical feature to focus on). The obtained abstract test cases cover, by construction, all possible executions exercising a given feature, and can be automatically translated into the inputs of autonomous driving simulators. We illustrate our approach by generating thousands of behavior trees for the CARLA simulator for several realistic configurations.**

***Index Terms*—Behavior trees, CARLA simulator, Formal methods, Input-output conformance, Scenario generation, Test purpose**

I. Introduction

AV (Autonomous Vehicles) are complex and safety critical systems. Although their main objective is to propose a safer alternative to human-driven cars, in the last years autonomous cars were involved in several accidents, some of them even fatal (e.g., the ones of a Uber car in 2018 and a Tesla in 2019 [3][1]). To prevent these accidents, all the components of an autonomous vehicle—which include both the sensors (e.g. LiDAR, cameras, GPS, etc.) and the algorithms that are behind the perception, decision making and control modules—need to be thoroughly tested to ensure they handle critical situations even better than human drivers. To avoid taking unnecessary risks, and because such critical situations are unlikely to happen—at least frequently enough—in real road traffic situations, a common and essential [8] practice is to reproduce these critical situations in a simulator, such as CARLA [7]. However, these scenarios to drive a simulator need to be specified. Two techniques are used currently: constrained random generation and manual specification [21]. Randomly generated scenarios can be easily produced, but their relevance might be difficult to assess, since they can present a high level of redundancy, which is hard to detect and strongly limits their coverage [21]. On the other hand, manually specifying a large number of scenarios is extremely time consuming and a satisfactory coverage of all possible situations is hardly achievable.

In this work we propose to apply formal methods to automatically generate scenarios, which are guaranteed to be relevant for testing AV's behaviour in a particular situation (e.g., collision, near miss, etc). More precisely, we will use a conformance testing tool to generate scenarios from a formal model and a test purpose characterizing the situation. The formal model is specialized to a given configuration, which includes a scene map and several actors with their initial positions and constraints on their trajectories. Precise trajectories of the actors will be automatically induced by the generated scenarios. In general, each test purpose will yield several scenarios, with guarantees to cover all relevant variations of the behavior related to the test purpose. These scenarios are then automatically transformed to be used as input for a driving simulator. To ensure the generation of relevant and critical scenarios, test purposes (e.g., reaching a collision) and test configurations can be defined

A part of the work has been performed in the project ArchitectECA2030 that has been accepted for funding within the Electronic Components and Systems for European Leadership Joint Undertaking in collaboration with the European Union's H2020 Framework Programme (H2020/2014-2020) and National Authorities, under grant agreement No. 877539. A part of this work has been supported by the PRISSMA project, co-financed by the French Grand Défi on Trustworthy AI for Industry. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

* Institute of Engineering Univ. Grenoble Alpes

[1] See also https://www.tesladeaths.com/ for a list of fatal accidents.

based on critical situations emerging from road accident data [6]. We illustrate our approach with CARLA by providing a method to translate the scenarios into behavior trees. Our approach is evaluated on ten configurations, involving three scene maps (T-crossing, highway, and X-crossing) and various actors, for which we generated several scenarios featuring collisions of the AV with other actors, near-misses of such collisions, and arrivals at the destination.

The rest of the paper is organized as follows. Section II compares our approach to existing ones for scenario generation in driving simulators. Section III gives a bird's eye view of our approach. Section IV illustrates the application of our approach on several configurations. Finally, Section V gives some concluding remarks and future work directions.

II. Related Work

Although AV scenario generation approaches have been intensively studied [21], current methods never test the complete ODD (Operational Design Domain) of the AV, but rather restrict scenarios to specific configurations, such as highway overtaking or an urban intersection. Hence, the testing methods can neither verify the full safety of the AVs nor be used at industrial scale, as they do not guarantee a complete coverage of the ODD.

A common way to create interesting and critical AV scenarios is to start from an existing abstract and parameterized scenario, from which several AV scenarios can be generated by choosing the values of the parameters. As not all of them are critical and interesting to run, several studies adopt optimization methods to find critical sets of parameters. The three following approaches use stochastic optimization methods to search the parameters values leading to the corner cases of an abstract scenario. The Matlab toolbox S-TaLiRo implements several optimization methods used in [22] to generate scenarios causing the motion controller of an AV to fail in an overtaking abstract scenario and generating a collision. Bayesian optimization is used by [9] to generate AV scenarios from a parameterized abstract scenario. In particular, to find the values of these parameters that cause collision between a car and a pedestrian, the optimization method also exploits the feedback from the simulated execution of the previously generated scenario. An AEB (Automatic Emergency Braking) system is tested in [13] using constrained randomization techniques to generate AV scenarios from an abstract scenario and then converge to critical AV scenarios by reducing the constraints. Our approach generates the actors behaviors (like car trajectories), thus we do not use an abstract scenario with predefined and parameterized behaviors as input. Additionally, our approach can generate several different scenarios covering better the many possible outcomes of an abstract scenario.

[1] and [14] relate the criticality of a scenario to the dimension of the solution space, defined as the control space that does not lead to a collision. The solution space is computed using reachability analysis. [14] uses an evolutionary algorithm to find the best parameters (speeds, initial positions, etc.) of an existing scenario to minimize the solution space, while [1] uses optimization techniques by modeling the generation problem as a quadratic problem with constraints. In contrast, our approach does not start with an existing parameterized scenario to then increase its criticality. We can generate this scenario based on an abstract scenario, and one could reuse our generated scenarios as an input for these approaches.

[16] proposes a different method to test an AEB system by defining an ontology based on the ODD of the AEB system, and then by using a combinatorial test generation approach to create AV scenarios from the ontology. A machine learning-based approach is also used to increase the criticality of the scenarios by exploiting the result of the already simulated scenarios. In contrast, our approach is used on several abstract scenarios while the ontology is associated to one abstract scenario of AEB testing. In addition, the complexity of the scenarios depends on the number of parameters and their value range in the ontology, and so this approach depends more on heavily parameterized scenarios than our approach.

Machine learning is also used to create realistic and critical scenarios. In [6], a neural network generates safety-critical AV scenarios from an abstract scenario of an urban intersection. The obtained scenarios are modeled as series of probability distributions, from which a final one is sampled and run on CARLA. [15] uses a generative adversarial network, trained on a dataset of vehicle trajectories on a highway, to generate realistic lane change trajectories. While learning-based generators are limited to what they learned, our approach does not have standard machine learning drawbacks, does not require long training, does not over fit the training data, and hence can generalize to new domains, i.e., we can include new configurations at inference time. Most importantly, our approach comes with guarantees that the generated scenarios are diverse (i.e, different trajectories), and all relevant to the AV's behavior in a given event (specified in the test purpose).

III. Methodology

Figure 1 gives an overview of the proposed flow. Its first input is a configuration defining the scene with its objects and their behavior, from which a formal model and a corresponding CARLA configuration are derived. The second input is a test purpose, describing the intent all test cases should focus on. From these inputs, we automatically compute a comprehensive test suite, and translate each generated test case into a behavior tree to drive the CARLA simulator. The subsequent sections present these steps, which are all fully automatic, besides the two steps represented by dotted lines, i.e., deriving a formal model and a CARLA configuration. To ease the latter two steps, we provide libraries factoring common parts.

A. Formal model of a scene

We focus on scenes with an ego vehicle, called *car*, moving around in the scene towards a goal or destination position, trying to avoid any collision with the *obstacles*. The car features sensors (e.g., camera, LiDAR, etc.) enabling it to perceive its environment. Obstacles represent the different hazards that may disturb the progression of the car. There are two kinds of obstacles: the static, fixed ones (e.g., buildings, trees, parked cars, etc.), and the dynamic, moving ones (e.g., cars, cyclists, pedestrians, etc.). The trajectories of dynamic obstacles can be

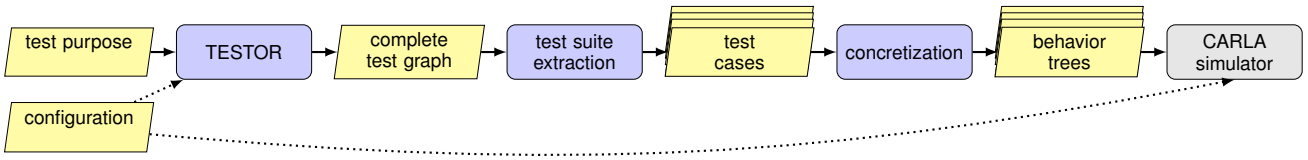
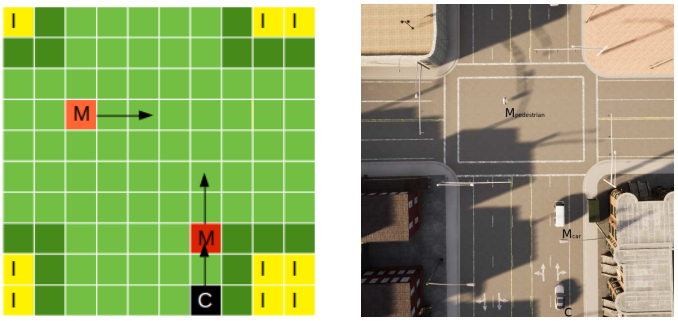


Fig. 1: Overview of the proposed approach to generate behavior trees from a configuration and a test purpose



(a) Ground truth grid

(b) CARLA simulation

Fig. 2: Scene map of a urban road crossing

predefined or (partially) left random to leave more room for varying behavior. Both kinds of obstacles can be transparent, in which case the sensors of the car can perceive the status of the cells behind the obstacle, so as to enable taking into account the difference between, e.g., a ball and a truck.

Considering a discrete, two-dimensional projection, we represent the map of a scene as a grid composed of cells, which can have one of the three different values: free, occupied by the car, or occupied by an obstacle. This discrete modeling helps to keep the size of the model tractable (note that the grid resolution can be made finer at the price of increased computing resources). The grid represents the ground truth under ideal conditions and is updated at each move of one of the actors (the car or a moving obstacle). Figure 2a shows the grid of a road-crossing scene. The buildings delimiting the two roads are represented as immobile obstacles (cells marked "I") and the car is denoted by the cell marked "C". The scene also comprises two moving obstacles (cells marked "M"): another vehicle in front of the car, and a pedestrian crossing the trajectory of the two vehicles. The cell of the pedestrian is depicted in a lighter color than the one of the other vehicle to indicate that the pedestrian is considered transparent and does not obstruct the view on the cells behind. A screenshot of a corresponding CARLA simulation is shown in Figure 2b.

We consider a car equipped with a LiDAR. For each configuration of the ground truth, the model can compute the perception grid, corresponding to the ideal expected output of the perception algorithm fed with the car's LiDAR input. We model the perception as a grid centered around the car, indicating whether a cell is free, occupied, or unknown (hidden from view). For each occupied cell, the perception also contains information whether the cell was occupied before, enabling to detect car or obstacle moves.

To formally model the behavior of such a scene, we use LNT [4], [11], a language equipped with a formal semantics rooted in concurrency theory and a syntax close to classical programming languages. Actors (car and moving obstacles) are represented as concurrent LNT processes that interact by means of multiway rendezvous [5], [12], enabling synchronization and communication. Each actor process has a local copy of the map, to ensure that its moves obey the laws of physics. The current status of the map is handled by a map manager process, in charge of maintaining a consistent ground truth. When moving, each actor sends its new position to the map manager, which updates the map and broadcasts it to all actors. This allows to represent moves to any adjacent cells (including diagonals) or even farther cells. The scenario terminates when either the car arrives at its destination, or a collision occurs: in these cases, the model will perform one of the special actions ARRIVAL or COLLISION and then stop immediately. This implies that an execution sequence of the model will never contain both, a collision and the arrival of the car at its destination.

LNT has an interleaving semantics, in which actions are considered atomic and two different actions cannot be observed simultaneously. To connect to a simulator where moves take some time and can thus happen concurrently, we introduce a notion of discrete time and add a scheduler process, which generates special TICK actions and ensures that all actors move at most once between two ticks. Thus, all moves between two TICK actions can be executed in parallel in the simulator. The scheduler also enforces a fixed order of execution of the actors between two TICK actions, which prunes redundant interleaving of actor executions and makes the subsequent analyses more efficient.

To facilitate the handling of various configurations, the LNT model of a configuration is split in two different parts: a generic part, defining types, functions, and processes common to all configurations, and a particular part, defining the constants characterizing the considered configuration. The latter part defines in particular the two dimensions of the rectangular grid, the initial grid (indicating any static obstacles), the number of obstacles, as well as the initial position and behavior of all actors (car and obstacles).

The behavior of the car is defined as a sequence of actions, specified by the speed and direction of move. When all actions of the sequence have been performed, the car has arrived at its destination. The behavior of an obstacle is defined in the same way, with three extensions. First, an obstacle may choose not to move (between two TICK actions). Second, an obstacle may perform a random move, where the direction is chosen arbitrarily (respecting the physical constraints of the scene and avoiding collisions). Third, rather than stopping, an obstacle

Fig. 3: Example test purpose "collision_pedestrian"

may restart the execution of its behavior sequence. In the current version of the model, the actors occupy a single cell and can turn as they wish. In principle, it is possible to refine the model and add constraints to represent rules of the dynamics (e.g., turning in at least x cells).

The LNT model is compiled into an automaton (state-transition graph) using the compilers of the CADP toolbox [10]. To generate tests from this automaton, we must specify which of its actions (i.e., transition labels) are considered as inputs or outputs. In our case, only the moves of the actors are inputs (to control the simulation), all the other actions being outputs (to observe the progress of the simulation), e.g., ground truth map, perception, detection of a collision, and arrival of the car. In the sequel, we will denote by "model" either the LNT description or its underlying automaton, since both represent the same behavior, albeit at different abstraction levels.

The model was validated using the CADP tools to ensure that it represents the desired scene and behaviors of the actors. Besides interactive simulation of the model (exploring back and forth the transition sequences from the initial state), we also checked the temporal logic property stating that a terminal state (either a collision, or the arrival of the car at its destination) will be eventually reached.

B. Computation of a Complete Test Graph

To obtain test scenarios focused on specific situations, we apply the conformance testing tool, which generates interaction scenarios with the simulator—in their simplest form, a sequence of transitions. As shown in Figure 1, we first extract a CTG (complete test graph) from the model and a TP (test purpose) using the TESTOR tool [18]. A TP is an automaton with special "ACCEPT" labels characterizing the states to be reached by the scenario, and a CTG is an automaton that contains *all* transition sequences leading to these states.

As an example, Figure 3 shows a TP to generate scenarios leading to a collision of the car with a pedestrian. This TP has two states and two transitions, requesting to reach (after an arbitrary number of transitions) a collision, represented by a transition labeled with "COLLISION !PEDESTRIAN".

When computing a CTG, only the transitions corresponding to a controllable input or observable output of the SUT (system under test, in our case the CARLA simulator) are necessary. Thus, we can hide—and reduce the model—all other transitions (e.g., the broadcast of the ground truth map) that are useful for validation, but irrelevant for test generation.

C. Extraction of test cases

In general, a CTG contains states for which several inputs can lead to a successful run. Thus, we apply the techniques of [19] to extract a test suite, i.e., a set of TCs (test cases) covering all transitions of the CTG. Each TC is an automaton interacting with the SUT to drive it towards the accepting states specified by the TP. Thus, for a given model, using even a simple TP as the one shown in Figure 3, several different TCs (and hence, scenarios) can be generated. In our setting, all generated TCs are sequences, since from a given state there cannot be two different outputs.

D. Translation of test cases into behavior trees

The TCs extracted from the model and a TP are represented in an abstract form as automata, and must be transformed into a more concrete form to be used as simulation scenarios. This last step is dependent on the simulator considered; we describe below the procedure we developed for the CARLA simulator. A simulation scenario in CARLA begins by initializing the configuration (scene map and placement of actors) symmetrically w.r.t. the formal model, and then specifying the behaviors of the actors. To run a simulation scenario, we use (an extended version of) the scenario_runner[2] feature of CARLA, which takes as input a behavior tree and uses it to control the actor moves during the scenario.

Our procedure to translate a TC into a behavior tree consists of three steps. Firstly, the TC is converted to a textual form, more suitable for parsing. Given that in our setting a TC is always a sequence of transitions, its corresponding text file contains one action per line, ending with a PASS action that denotes the end of the sequence (i.e., the accepting state of the TP). Secondly, the text file is analyzed line-by-line using a shell script, which extracts the necessary information about the moves and new positions of the actors, and produces a JSON file containing an initialization part (declaration of the actors with their initial placement and orientation) and a behavior part describing the sequence of moves for each actor. Thirdly, this JSON file is fed as input to a Python program that parses it, builds a behavior tree in memory using the scenario_runner primitives, and traverses this behavior tree to drive the simulation in CARLA by sending the commands associated to the tree nodes.

Figure 4 shows an example of TC and Figure 5 shows its corresponding behavior tree. To make the simulation smooth and realistic, the moves of the actors between two consecutive TICK actions in the TC are executed simultaneously by encapsulating them in *parallel* nodes of the behavior tree. These parallel nodes are then encapsulated into one *sequential* node (the root of the behavior tree) representing the sequence of all simultaneous moves in the scenario.

IV. Experiments

We illustrate our approach on several configurations, with different maps and obstacle behaviors. The experiments were performed on the `troll` cluster of the Grid'5000 platform, i.e., servers with an Intel Xeon Gold 5218 processor and 384 GB of RAM. The CARLA simulations were run on a laptop with an Intel i5-7440HQ, 16 GB of RAM, and an Nvidia Quadro P4000. The simulation was synchronized with CARLA's scenario_runner to ensure that, for each simulation step, the simulation time was constant and the scenario_runner completed its control loop.

[2]https://github.com/carla-simulator/scenario_runner

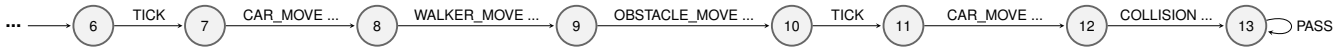


Fig. 4: Excerpt of a test case generated for model "X-crossing1" (see Fig. 2) and test purpose "collision_pedestrian" (see Fig. 3)

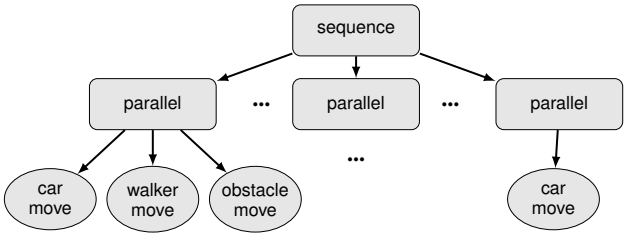


Fig. 5: Excerpt of the generated behavior tree for Figure 4

model	#obst.	#lines	states	transitions	time	memory
T-cross	1	96	258	449	10	651
highway	2	97	1277	2431	13	599
X-cross-1	2	107	4860	9403	24	744
X-cross-2	2	104	1848	3432	19	745
X-cross-3	2	104	3232	6166	19	745
X-cross-4	2	105	1865	3684	16	745
near_miss	3	121	18359	38753	252	842
immobile_car	3	114	345166	707363	7497	14831
X-cross-rand1	2	107	111401	226558	1408	3741
X-cross-rand2	2	107	144045	296559	1761	4490

TABLE I: Formal models of the considered configurations

Table I gives statistics about the formal models (LNT descriptions and their underlying automata) of the considered configurations. For each model, the table indicates the number of obstacles, the number of LNT lines specific to the configuration (the generic part, common to all models, has 1130 LNT lines) and the size (number of states and transitions) of the minimized automaton. The execution time in seconds (resp., memory requirements in MB) corresponds to the sum (resp., peak) of both generation and minimization of the automaton.

All models except "highway" and "T-cross" use the map shown in Figure 2a. Models "X-cross-N" ($N \in \{1, 2, 3, 4\}$) focus on a collision between the ego car and a pedestrian, featuring also another moving car; differing in the initial positions and behavior, these configurations were designed as simple tests of the approach. In model "near_miss" the itinerary of the car is crossed by different obstacles, leading to near misses (i.e., an obstacle passing very close to the car, but without colliding with it). Model "immobile_car" features the car stopped in the middle of the crossroad, with various obstacles moving around, but without causing any collision. Model "X-cross-rand1" introduces additional variability by enabling random moves of the pedestrian and the other car; their precise moves in a TC will be chosen according to the TP. Model "X-cross-rand2" is similar to "X-cross-rand1", but features a different and longer trajectory of the ego car (11 moves instead of 7). Model "T-cross" uses a map with a T-shaped crossroad, with a collision between the ego car and another car ignoring priority rules. Model "highway" uses a rectangular map representing a long horizontal highway with the ego car on the middle lane, a faster car trying to overtake it, and a slower car ahead of it, but leaving the lane to let the ego car pass.

The difference in the model sizes can be explained by the varying number of obstacles and the number of moves of the actors. Models "immobile_car" and "near_miss" feature three, instead of two, obstacles, with a large number of moves (in total, 12 moves for "near_miss" and 23 for "immobile_car"). In particular, a random obstacle move amounts to possibly five transitions, corresponding to each direction it can go (up, down, right, left, or none). Also, as the model stops when the ego car has finished, the longer the car trajectory, the larger the size of the underlying automaton, as can be seen by comparing models "X-cross-rand1" and "X-cross-rand2".

We consider different test purposes for these models. Purpose "arrival" requires the car to complete its behavior without any collision. Purpose "collision_X" requests a collision with obstacle X ($X \in \{\text{car}, \text{pedestrian}\}$). Purpose "lidar" requests the LiDAR to perceive an obstacle right next to the car.

Table II gives statistics about the test suite extraction for various pairs of a model and a test purpose. For each pair, columns three to six give statistics about the computation of the complete test graph: its size, generation time, and peak memory requirement. Columns seven to eleven give statistics about the generated test suite: number of test cases/scenarios, total size of the test suite (sum of states and transitions), generation time, and peak memory requirement. Column twelve gives the standard deviation (σ) for the number of test case states, showing that the test cases are of comparable size. Column 13 gives the total number of lines of the generated behavior trees.

In our experiment, we generated 9700 scenarios in total for testing AV's behavior in ten different configurations involving four TPs. Importantly, this did not require any prior knowledge about the configurations that lead to realistic scenarios. Instead, such configurations were found iteratively. Indeed, after running the behavior trees in CARLA, we observed unrealistic behaviors (e.g., actors performing discrete moves rather than continuous ones), which allowed us to improve the model by introducing a scheduler.

Another improvement concerns the translation of the discrete (grid-based) moves to the continuous environment of CARLA. When improving our model (for example by adding lane changing and turns at intersection) we had to smooth the trajectories (the motion control of the actors) for more realism. To do so, we represented in the scenario_runner the sequence of grid-based moves as a sequence of waypoints the actors must pass through, following a realistic trajectory obtained by applying PID (Proportional Integral Derivative) controllers at each simulation step.

V. Conclusion

We proposed an approach that exploits behavioral conformance testing to automatically generate, from a formal model and test purposes, scenarios in the form of behavior trees for the

model	test purpose	complete test graph				test suite (all test cases / scenarios)						behavior
		states	transitions	time	mem.	nb	states	transitions	time	mem.	σ	JSON lines
T-cross	collision_car	37	64	3	68	8	151	165	119	37	2	656
T-cross	arrival	238	453	3	69	71	4170	4847	1010	39	16	9761
highway	arrival	1105	2197	3	75	312	28078	29743	4341	38	41	78556
highway	collision_car	67	118	3	68	3	298	320	1	36	18	735
X-cross-1	collision_pedestrian	177	338	3	70	48	1803	1908	680	37	8	7296
X-cross-2	collision_pedestrian	206	396	3	72	55	2042	2128	778	38	12	8432
X-cross-3	collision_pedestrian	490	923	3	73	110	5666	5940	1549	39	17	25194
X-cross-4	collision_pedestrian	299	580	3	72	87	2752	2856	1225	38	6	13964
near_miss	lidar	2035	4459	3	940	688	26617	29325	9950	46	9	125200
near_miss	arrival	16385	36403	3	978	3603	746616	836030	50400	63	40	898978
immobile_car	lidar	5301	10210	4	262	1294	126640	131776	18637	50	14	312633
X-cross-rand1	lidar	2278	4805	4	378	981	41465	43127	14118	57	5	219734
X-cross-rand2	collision_pedestrian	2667	5970	3	459	1668	103589	105951	24128	63	11	454210
X-cross-rand2	collision_car	1701	3680	3	456	772	92085	96953	11191	59	8	189752

TABLE II: Test suite extraction statistics

CARLA simulator. Besides being to a large extent automatic, the generated scenarios are guaranteed to focus on specified test purposes, which provide a versatile manner of targeting corner-case situations. The approach has been illustrated on several examples of configurations (scenes and moving actors).

Our approach makes possible the formalization (as test purposes) of safety requirements defined in standards, such as ISO 26262, and the generation of test cases, similarly to [17]. The test cases can be used both for producing simulation scenarios (as we illustrated here) and for assessing the correct functioning of AVs in their environment.

Concerning future work, we plan to extend our approach to take advantage of new AV simulation standards gaining popularity, such as OpenDRIVE[3] for map representation and OpenSCENARIO[4] for scenario description (both already used by CARLA). We also plan to enhance the formal model by including near miss thresholds [20] and the traffic-sign rules [2], which would make possible a refined scenario selection according to different levels of criticality.

REFERENCES

[1] M. Althoff and S. Lutz. Automatic generation of safety-critical test scenarios for collision avoidance of road vehicles. *IEEE Intelligent Vehicles Symposium (IV)*, pages 1326–1333, 2018.

[2] G. Bagschik, T. Menzel, and M. Maurer. Ontology based scene creation for the development of automated vehicles. In *IEEE Intelligent Vehicles Symposium, IV*, pages 1813–1820, 2018.

[3] N. Boudette. 'It Happened So Fast': Inside a Fatal Tesla Autopilot Accident. https://www.nytimes.com/2021/08/17/business/tesla-autopilot-accident.html, Aug. 2021.

[4] D. Champelovier, X. Clerc, H. Garavel, Y. Guerte, C. McKinty, V. Powazny, F. Lang, W. Serwe, and G. Smeding. Reference Manual of the LNT to LOTOS Translator (Version 7.0). INRIA, Grenoble, Mar. 2021.

[5] A. Charlesworth. The Multiway Rendezvous. *ACM Transactions on Programming Languages and Systems*, 9(3):350–366, July 1987.

[6] W. Ding, B. Chen, M. Xu, and D. Zhao. Learning to collide: An adaptive safety-critical scenarios generating method. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2243–2250. IEEE, 2020.

[7] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[3]https://www.asam.net/standards/detail/opendrive/

[4]https://www.asam.net/standards/detail/openscenario/

[8] W. F. for the Harmonization of Vehicle Regulations. New assessment/test method for automated driving (natm) - master document. Technical report, UNECE, 2021.

[9] B. Gangopadhyay, S. Khastgir, S. Dey, P. Dasgupta, G. Montana, and P. A. Jennings. Identification of test cases for automated driving systems using bayesian optimization. In *Intelligent Transportation Systems Conference (ITSC)*, pages 1961–1967. IEEE, 2019.

[10] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. *Springer Int. J. on Software Tools for Technology Transfer (STTT)*, 15(2):89–107, 2013.

[11] H. Garavel, F. Lang, and W. Serwe. From LOTOS to LNT. In *ModelEd, TestEd, TrustEd – Essays Dedicated to Ed Brinksma on the Occasion of His 60th Birthday*, volume 10500 of *LNCS*, pages 3–26. Springer, 2017.

[12] H. Garavel and W. Serwe. The Unheralded Value of the Multiway Rendezvous: Illustration with the Production Cell Benchmark. In *2nd Workshop on Models for Formal Analysis of Real Systems (MARS'17)*, volume 244 of *EPTCS*, pages 230–270, 2017.

[13] S. Khastgir, G. Dhadyalla, S. Birrell, S. Redmond, R. Addinall, and P. Jennings. Test scenario generation for driving simulators using constrained randomization technique. Technical report, SAE Technical Paper, 2017.

[14] M. Klischat and M. Althoff. Generating critical test scenarios for automated vehicles with evolutionary algorithms. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 2352–2358, 2019.

[15] R. Krajewski, T. Moers, D. Nerger, and L. Eckstein. Data-driven maneuver modeling using generative adversarial networks and variational autoencoders for safety validation of highly automated vehicles. In W. Zhang, A. M. Bayen, J. J. S. Medina, and M. J. Barth, editors, *International Conference on Intelligent Transportation Systems (ITSC), Maui, HI, USA*, pages 2383–2390. IEEE, 2018.

[16] Y. Li, J. Tao, and F. Wotawa. Ontology-based test generation for automated and autonomous driving functions. *Information and software technology*, 117:106200, 2020.

[17] D. Makartetskiy, G. Marchetto, R. Sisto, F. Valenza, M. Virgilio, D. Leri, P. Denti, and R. Finizio. (User-friendly) formal requirements verification in the context of ISO26262. *Engineering Science and Technology, an International Journal*, 23:494–506, 2020.

[18] L. Marsso, R. Mateescu, and W. Serwe. TESTOR: A Modular Tool for On-the-Fly Conformance Test Case Generation. In *24th Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'18)*, volume 10806 of *LNCS*, pages 211–228. Springer, 2018.

[19] L. Marsso, R. Mateescu, and W. Serwe. Automated Transition Coverage in Behavioural Conformance Testing. In *32nd IFIP Int. Conference on Testing Software and Systems (ICTSS'20)*, pages 219–235, 2020.

[20] A. Pierson, W. Schwarting, S. Karaman, and D. Rus. Learning risk level set parameters from data sets for safer driving. In *IEEE Intelligent Vehicles Symposium, IV 2019, Paris, France*, pages 273–280. IEEE, 2019.

[21] S. Riedmaier, T. Ponn, D. Ludwig, B. Schick, and F. Diermeyer. Survey on scenario-based safety assessment of automated vehicles. *IEEE Access*, 8:87456–87477, 2020.

[22] C. E. Tuncali, T. P. Pavlic, and G. E. Fainekos. Utilizing s-taliro as an automatic test generation framework for autonomous vehicles. In *International Conference on Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil*, pages 1470–1475. IEEE, 2016.