A Dependency-based Combinatorial Approach for Reducing Effort for Scenario-based Safety Analysis of Autonomous Vehicles

Kaushik Madala[1], Hyunsook Do[1] and Carlos Avalos-Gonzalez[2]
[1]*Department of Computer Science and Engineering, University of North Texas, Denton, U.S.A.*
[2]*kVA by UL, Portland, U.S.A.*

Keywords: Scenario-based Analysis, Safety Verification, Operation Design Domain, Autonomous Vehicle Safety.

Abstract: For an autonomous vehicle, to assure safety, we need to perform a thorough analysis considering the vehicle's intended operational design domain (ODD). This requires analysts and engineers to consider various operating environments (OEs) that can occur in the ODD, and the various scenarios that are possible within each OE. However, the automotive safety standards ISO 26262 and ISO 21448 do not offer in-depth guidance on what and how many scenarios need to be analyzed to ensure safety of a vehicle. Moreover, many existing simulation tools and verification approaches consider limited OEs and generate test cases exhaustively for each scenario created by engineers within an OE. Such an analysis requires a significant amount of time and effort, but it still cannot ensure that various dependencies among ODD elements are covered. To address these limitations, we propose a dependency-based combinatorial approach (DBCA), which uses in-parameter-order-general (IPOG), a combinatorial testing algorithm to generate OEs and test cases for each scenario. To evaluate DBCA, we applied it to the ODD elements extracted from ISO 21448, and to a highway cut-in scenario. Our results show that DBCA reduced time and effort for analysis, and reduced the the number of OEs and test cases for the scenario without missing dependencies.

1 INTRODUCTION

Scenario-based analysis plays a vital role in assuring safety of an autonomous vehicle because various known and unknown hazardous scenarios that occur within a vehicle's operational design domain (ODD) can comprise the safety of the vehicle. Although the automotive industrial safety standards ISO 26262 (ISO, 2018) and ISO 21448 (ISO/PAS, 2020) offer guidance on how to analyze functional safety (FuSa) and safety of the intended functionality (SOTIF) of a vehicle respectively, they do not offer in-depth guidance on how to generate scenarios for analyzing safety of a vehicle, and what and how many scenarios will ensure that a vehicle is safe enough to be deployed.

The current methods (Kalra and Paddock, 2016; Akagi et al., 2019; Althoff and Lutz, 2018) for analyzing safety of autonomous vehicles rely on either the amount of miles traveled by the autonomous vehicles or by verifying the behavior of vehicles mostly on known safety-critical driving scenarios and then ensuring that the vehicles meet a predefined validation target (set by experts based on naturalistic driving data(VTTI, 2020b; VTTI, 2020a) or accident based data (NHTSA, 2018)). For example, Altoff and Lutz (Althoff and Lutz, 2018) proposed an approach for generating test cases for possible safety critical scenarios for a vehicle based on its drivable area. While these existing approaches aid in identifying potential situations in which an autonomous vehicle's safety might be compromised, they still suffer from the following limitations:

- Current approaches used in hazard analysis in both Part 3 of ISO 26262, and Part 6 of ISO 21448 rely on brainstorming of experts to identify scenarios and their corresponding operating environments. If some scenarios or environments are missed/overlooked during this analysis, then they are not tested during simulation.
- The amount of miles becomes an effective metric only if diverse and representative scenarios are covered. If vehicle fleets are tested on the same road in simulation or in the real world, it is possible to miss some scenarios and operating conditions that can occur within the defined ODD.
- Current safety-critical scenario generation ap-

Madala, K., Do, H. and Avalos-Gonzalez, C.
A Dependency-based Combinatorial Approach for Reducing Effort for Scenario-based Safety Analysis of Autonomous Vehicles.
DOI: 10.5220/0010495502350246
In *Proceedings of the 7th International Conference on Vehicle Technology and Intelligent Transport Systems (VEHITS 2021)*, pages 235-246
ISBN: 978-989-758-513-5

proaches take into account different possible driving scenarios for a fixed environment, i.e., for a fixed set of ODD elements and their attributes. However, driving scenarios are just a part of operating scenarios for an autonomous vehicle. For a vehicle with driving automation, as mentioned in ISO 21448, to identify a scenario, we should consider the combination of environmental factors (e.g., snowfall, rain), driving scenarios (e.g., overtaking a vehicle, performing cut-in in front of a vehicle), behaviors of agents involved (e.g., pedestrians, other vehicles), road geometry (e.g., straight road), road infrastructure (e.g., traffic signs) and goals/objectives of the scenario i.e., the tasks we want to accomplish in the scenario (e.g, an ego vehicle should stop when a pedestrian is crossing the road in city streets).

Moreover, the current simulation tools (e.g., CARLA (Dosovitskiy et al., 2017), Fortellix (Fortellix, 2020)) require to create operating environments before testing scenarios and most of the operating environments have been created based on customers' needs (Fadaie, 2019). However, whether these operating environments can cover the entire operational design domain (ODD) defined by engineers of autonomous vehicles is not verified. Also, most of the current simulation tools perform exhaustive testing, i.e., generating all possible test cases for each scenario with all discrete parameter values, their ranges and increments initialized by the engineers and running simulations within a fixed operating environment. While tools such as Fortellix (Fortellix, 2020) offers a means to combine scenarios and operating conditions, the scenario description is restricted to Open M-SDL (Fortellix, 2020) specification, which does not require to have all the essential ODD elements and attributes we aim to cover as a part of the ODD. Performing analysis with such tools does not account for overlooked ODD elements or their attributes. For example, if an attribute of a pedestrian such as race or gender cannot be set in a simulation tool despite the presence of the attribute in ODD, it may be ignored during the verification using simulation. Further, given the complexity of ODD for autonomous vehicles, simulating all potential scenarios with operating environments might not be feasible. This is because the need for resource (e.g., time and effort) increases as the complexity of the ODD increases.

For example, let us consider the following factors that are part of ODD (taken from Table B.3 of ISO 21448): climate, time of the day, road shape, road feature (e.g., tunnel, gate), condition of the road, lighting (e.g., glare), condition of the ego vehicle (e.g., a sensor covered by dust), operation of the ego vehicle (e.g., a vehicle is stopping), surrounding vehicles (e.g., a vehicle to the left of the ego vehicle), road participants (e.g., pedestrians), surrounding objects offroadway (e.g., a traffic sign), objects on the roadway (e.g., lane markings). Each of these factors can have multiple values. For example, for 'time of the day,' its values can be early morning, daytime, evening, and nighttime. Based on the values given in Table B.3 of ISO 21448, the total number of combinations for ODD factors is 169,554,739,200. Note that we have not considered properties of agents (e.g., gender of a pedestrian), vehicles (e.g., speed of the vehicle), environmental attributes (e.g., amount of snowfall) yet.

These combinations only represent operating environments and are still not complete as multiple agent types and vehicle types are not considered. For each of these combinations, we need to generate instances of scenarios to test by considering properties of the agents, environmental attributes, and an ego vehicle. Examples of properties are the amount of rainfall (environmental), the number of randomly initialized pedestrians (agents related), and a speed range of the vehicle (ego vehicle related). The properties values are manually selected/initialized by the simulation engineers and experts. Test cases for simulation are generated based on these properties by considering all possible combinations among the properties. In the example properties considered above, if we assume rainfall can range from 0 cm to 10 cm with an increment of 0.5 cm, the number of randomly initialized pedestrians can range from 0 to 20, with an increment of 1, and the speed range of a vehicle is considered to be between 30 mph and 90 mph, with an increment of 5 mph, then an exhaustive testing strategy result will be 21 (for rainfall)$\times$ 21 (for pedestrians)$\times$ 13 (for the speed range) = 5733 tests. As the number of properties, their ranges, and their increments change, this will result in a large number of tests. Creating a large number of operating environments and performing exhaustive testing can be very expensive and often it is not feasible to do so. Moreover, analyzing test cases that expose collisions and near misses from a significantly large number of test cases is difficult as often experts need to manually analyze the causes of collisions and near misses.

To address these limitations, we propose a dependency-based combinatorial approach (DBCA) for operating environment identification and test suite optimization for analysis of scenarios. DBCA utilizes IPOG (Lei et al., 2007; Lei et al., 2008), a widely used combinatorial testing algorithm, to generate t-way combinations of operating environments and test cases for each scenario defined in those operating en-

vironments. We chose a combinatorial approach because it is found to be effective and efficient in testing highly configurable systems (Kuhn et al., 2010; Wotawa, 2017), which require to run without failures on many thousands of configurations. Because autonomous vehicles also need to ensure safety of the system under various operating conditions, we believe a combinatorial approach will be a perfect solution for reducing the complexity of their safety analysis. The 't' value in t-way combinations is manually chosen based on the dependencies among factors being considered to generate operating environments and instances of scenarios. To evaluate whether our approach can aid in identifying as well as reducing the number of operating environments and instances of scenarios that needs to be considered for verifying safety of an autonomous vehicle, we performed a case study using Metamoto, a simulation tool. Our results show that our approach is effective at reducing the number of test cases for a scenario without affecting the detection of collisions when compared to exhaustive approach. We also found that it is important to consider the different operating environments that need to be considered based on ODD defined for an autonomous vehicle as we found that often only limited ODD is considered in simulation thereby missing potential unknown scenarios. Note that in this paper, we restrict our analysis to vehicle level behavior and do not discuss machine learning safety-based verification.

The rest of the paper is organized as follows. Section 2 discusses background and related work. Section 3 explains DBCA approach and Section 4 discusses the empirical study, its results, and threats. Section 5 details the insights from results and limitations of DBCA. Finally, we conclude in Section 6.

2 TERMINOLOGY AND RELATED WORK

In this section, we discuss the various terminologies we use throughout the paper and the related literature to DBCA.

2.1 Terminology and Their Descriptions

1. *Ego Vehicle:* It is the autonomous system for which we perform analysis.
2. *Operational Design Domain (ODD):* It is defined as a set of conditions that includes environmental factors, road infrastructural elements, agents around vehicles, and various driving scenarios for which a system or a feature of the system is designed (BSI/PAS, 2020; ISO/PAS, 2020).
3. *ODD Element:* An entity or agent that is part of the ODD. For example, weather and pedestrians are ODD elements. Every ODD element has corresponding values. For example, weather can have multiple values: rainy, snowy, and sunny. Every ODD element and every value of an ODD element can have associated properties. For example, when the weather is snowy, we can have a property such as the amount of snowfall.
4. *Operating Environment:* The environment in which an ego vehicle operates. It usually includes the agents, road infrastructure and environmental factors. An operating environment is a subset of ODD, i.e., for a given ODD, we can identify many operating environments. An example of an operating environment is a wet and straight highway road with 3 lanes that has trucks and vehicles on it, where the ego vehicle is moving forward.
5. *Scenario:* A scenario is a temporal sequence of situations with actions and events in an operating environment where an ego vehicle aims to achieve goals and objectives defined by the engineers (ISO/PAS, 2020). In a given operating environment, multiple scenarios can occur. For the highway operating example discussed previously, a scenario can be for the ego vehicle to reach destination B without a collision while a lead vehicle brakes suddenly. Note that every scenario can have multiple instances. For the scenario example discussed previously, we can generate various instances by considering different separating distances between the lead and the ego vehicles, and various braking force values for the lead vehicle.
6. *IPOG Algorithm (Lei et al., 2007; Lei et al., 2008):* A combinatorial testing algorithm used to generate t-way combinations of parameter values. The value of 't' is smaller when compared to the total number of parameters. A t-way combination implies all combinations between any 't' elements are considered. For example, if we have four parameters p1, p2, p3 and p4 with two values each, 2-way combinations of the parameters represent the combinations in which all possible pairs of values between every two parameters are considered. In DBCA, we use IPOG algorithm to generate combinations of ODD elements and combinations of properties to generate instances of a scenario.
7. *Functional Safety (FuSa):* It is a characteristic of a system in which malfunctions of electrical and electronic components do not result in an unrea-

sonable risk (ISO, 2018). The process of functional safety analysis for automotive systems is detailed in ISO 26262 (ISO, 2018).

8. *Safety Of The Intended Functionality (SOTIF):* Unlike functional safety, safety of the intended functionality is an absence of unreasonable risk due to insufficiencies in defining the functionalities of the system (ISO/PAS, 2020). SOTIF aids in reducing unknown risks as well as gaps in requirements and design. The SOTIF analysis process for vehicles with driving automation is detailed in ISO 21448 (ISO/PAS, 2020).
9. *Exhaustive Simulation:* If we run all possible combinations of the parameters' values for a given scenario in a simulation tool, then we refer to it as an exhaustive simulation. It takes significantly a high amount of time and resources.

2.2 Related Work

Generating scenarios for testing autonomous vehicles is a challenging task. Koopman and Wagner (Koopman and Wagner, 2016) have indicated that complete testing is infeasible for an autonomous vehicle. To address this problem, to date, many researchers (Li et al., 2020; Mullins et al., 2018; Nonnengart et al., 2019; Akagi et al., 2019) have proposed techniques to generate scenarios for simulation.

Some researchers (Li et al., 2020; Rocklage et al., 2017; Duan et al., 2020; Gannous and Andrews, 2019) used combinatorial approaches to generate scenarios. For example, Li et al. (Li et al., 2020) proposed an ontology-based combinatorial testing approach, in which ontology is used to identify parameters and their values, which are then given as input to the combinatorial algorithm that generates n-way combinations. The authors further refine these combinations to identify test cases corresponding to critical scenarios by using a machine learning model. Another example is the static and hybrid scenario generation approach proposed by Rocklage et al. (Rocklage et al., 2017), in which the authors use combinatorial interaction testing to identify feasible trajectories for motion planning in a given scenario using a backtracking algorithm for solving the constraints defined for checking feasibility of trajectory.

Other researchers (Calò et al., 2020; Mullins et al., 2018) have proposed search-based techniques to generate scenarios. For example, Mullins et al. (Mullins et al., 2018) proposed a framework for generating scenarios using search strategy that performs adaptive sampling using surrogate optimization to identify high quality scenarios, followed by density based clustering to group the scenarios in the order of their effectiveness.

A few researchers (Nonnengart et al., 2019; Althoff and Lutz, 2018) have proposed formal approaches to generate scenarios for autonomous vehicles. For example, Nonnengart et al. (Nonnengart et al., 2019) proposed a formal-method based approach called CriSGen, which is used to generate critical scenarios by formalizing abstract scenarios and maneuvers and then performing a forward reachability analysis to identify if a maneuver can result in an unsafe state.

In addition to the aforementioned methodologies, some researchers (Akagi et al., 2019; Koopman and Wagner, 2016) have also used naturalistic driving data (VTTI, 2020b; VTTI, 2020a; van Nes et al., 2019) to generate scenarios. An example is the framework by Akagi et al. (Akagi et al., 2019), in which the authors generate scenarios from a probabilistic model built using naturalistic driving data (e.g., SHRP2 (VTTI, 2020b)), and sampling vehicle kinetic information and traffic risk index values.

While these techniques aid in identifying critical scenarios, they do not consider whether the operating environments they use cover all the ODD elements. Moreover, to test a scenario, the techniques either produce test cases of parameters defined for the scenario either randomly or exhaustively. However, random generation of tests does not guarantee to cover the dependencies among systems' components and ODD elements, and exhaustive testing requires a significant amount of effort, time, and resources. Using DBCA, we aim to overcome these limitations.

3 APPROACH

The overview of dependency-based combinatorial approach (DBCA) is shown in Figure 1. The ovals represent the steps in the approach and the rectangles represent the outputs from the steps which can be used as inputs to their next steps. The parallelogram represents the existing combinatorial algorithm we use in DBCA. The numbers in the ovals represent the step numbers. Each step is described in detail as follows.

Step 1. Define ODD and Dependencies. As a first step of DBCA, we (stakeholders) define ODD of the autonomous vehicle being analyzed. While ODD is usually refined when performing the hazard analysis and risk assessment conducted during the SOTIF analysis, in this paper, we assume that ODD has already been refined and considered, and we focus more on identifying operating environments and optimizing test cases to verify a scenario in simulation. As

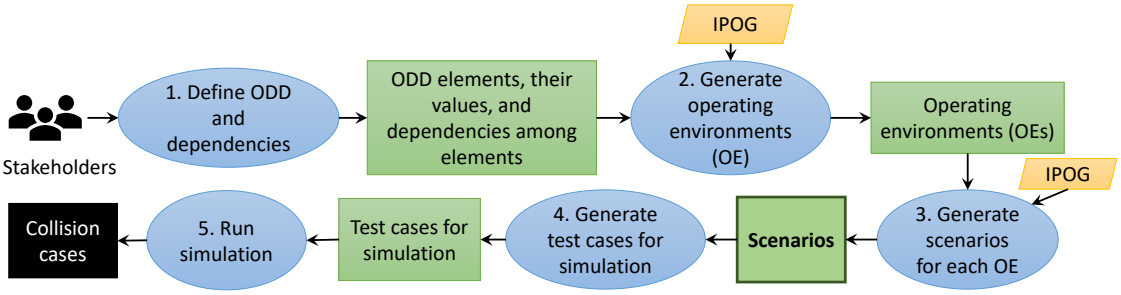

Figure 1: Overview of dependency-based combinatorial approach (DBCA).

mentioned in Section 2, ODD comprises of ODD elements, their values, and properties. ODD elements include environmental factors (e.g., weather, time of the day), agents (e.g., surrounding vehicles on the road), road infrastructure (e.g., traffic signs, zones, junctions) and operations of the autonomous vehicle under analysis, i.e., an ego vehicle (e.g., a vehicle is making a left turn). A sample of the detailed list of ODD can be found in BSI:PAS 1883 (BSI/PAS, 2020).

Every ODD element has corresponding values. For example, weather can have values such as rainy, snowy, cloudy, and clear. Every ODD element value can have its associated properties/parameters. For example, when the ODD element is weather and its value is rainy, we can have a property such as the amount of rain. After the ODD elements, their values, and properties are defined, we identify dependencies among ODD elements by considering their values. For example, let us consider three ODD elements: weather, road condition, and time of the day. Weather can affect a road condition. For example, a road that is not leveled properly might have puddles when the weather is rainy. Hence, we consider a dependency between weather and a road condition. However, a road condition and weather do not affect time of the day. Hence, we do not consider them to be having any dependencies. We identify all such dependencies and the absence of dependencies among ODD elements.

In this paper, due to a lack of support for automatic dependency identifications by the state-of-the-art techniques, we identify dependencies manually because it helps us in providing sufficient evidence and justification during safety assessment of the vehicle that our combinatorial approach reduced the number of operating environments and test cases for scenarios without overlooking any dependencies among ODD elements.

Step 2. Generate Operating Environments. After the dependencies among ODD elements are identified, we find the maximum number of ODD elements that are dependent on each other. We identify this maximum number as 't' that is applied to IPOG (Lei et al., 2007; Lei et al., 2008), a widely used combinatorial testing algorithm to produce t-way combinations of ODD elements along with their values. These combinations represent the potential operating environments.

Let us consider the example of ODD elements we have chosen in the previous step. The elements are weather, a road condition, and time of the day. Let us assume each of the elements to be having the following values: weather - {rainy, snowy, cloudy, clear} (4 values), road condition - {dry, wet with puddles, wet with no puddles} (3 values), time of the day - {day, dusk/dawn, night} (3 values). For this simple set of ODD elements, if we plan to generate all possible combinations, we get $4 \times 3 \times 3 = 36$ combinations. However, as mentioned in the previous step, only weather and road condition have a dependency (e.g., a rainy weather can result in a wet road with puddles). So, as long as we are able to cover all possible combinations between weather and road condition, we will be considering all possible dependencies among elements. Hence, the 't' values will be 2 for this example. When we generate 2-way combinations of the three ODD elements, 36 combinations reduce to 12 combinations. Figure 2 illustrates the reduction in the combinations. It can be observed from the figure that the 2-way combinations not only covered all possible combinations between {weather, road condition}, but also covers all possible combinations between {road condition, time of the day}, and {time of the day, weather}.

We then go through these combinations that represent operating environments and check if the simulation tool has environments that match the required operating environments. If the tool does not offer such environments, we request the engineers who are responsible for the simulation tools to develop re-

All possible combinations between the three ODD elements

Weather	Road condition	Time of the day
Rainy	Wet without puddles	Day
Rainy	Wet without puddles	Dawn/Dusk
Rainy	Wet without puddles	Night
Rainy	Wet with puddles	Day
.....	.....	.....
.....	.....	.....
Clear	Wet with puddles	Day
Clear	Wet with puddles	Dawn/Dusk
Clear	Wet with puddles	Night
Clear	Dry	Day
Clear	Dry	Dawn/Dusk
Clear	Dry	Night

Total combinations = 36

2-way IPOG

All possible combinations between every two ODD elements are covered

Weather	Road condition	Time of the day
Rainy	Wet without puddles	Day
Rainy	Wet with puddles	Night
Rainy	Dry	Dusk/Dawn
Cloudy	Dry	Night
Cloudy	Wet with puddles	Day
Cloudy	Wet without puddles	Dusk/Dawn
Snowy	Dry	Day
Snowy	Wet with puddles	Dusk/Dawn
Snowy	Wet without puddles	Night
Clear	Dry	Day
Clear	Wet with puddles	Dusk/Dawn
Clear	Wet without puddles	Night

2-way combinations = 12

Figure 2: Example of illustrating the reduction in the number of combinations of ODD elements using combinatorial approach.

quired environments by providing information on the elements that need to be present in those environments respectively.

Step 3. Generate Scenarios for Each Operating Environment (OE). As mentioned above, we consider each combination of ODD elements along with their corresponding values as an operating environment. In a given operating environment, various scenarios can occur. This is because according to ISO 21448 (ISO/PAS, 2020), every scenario requires to have a goal or objective defined. An example of a goal can be "an ego vehicle travels from location A to location B in the map in 5 minutes without any collision." Because this approach relies on expertise of engineers, in our approach, defining goals and objectives is manual.

Each scenario can have multiple unique instances because the various properties and parameters can be associated with each ODD element. For example, if we consider the weather to be rainy, the road condition to be wet with puddles, and time of the day to be night, then their corresponding properties can be the amount of rainfall, the density of the road puddles (the number of puddles per square meter in a road), and time in PM/AM. Note that the properties of the ODD elements must be chosen based on the goals/objectives of the scenario.

Let us assume, that we have a rainfall range established from 0 cm to 15 cm with an increment of 1 cm (a total of 16 values), the density of road puddles to be ranging from 0 to 1, with an increment of 0.1 (a total of 11 values), and time in PM/AM to be ranging from 7 PM to 4 AM, with an one hour increment (10 values). Let us also assume that we have considered the ego vehicle to be traveling in a straight line at 60 kmph. For this circumstance and given property values, the total number of possible instances of a high-level scenario that can occur within an operating environment is $16 \times 11 \times 10 = 1760$ combinations.

However, as the number of ODD elements, their values, their properties and the ranges of the properties increase, the number of combinations rise drastically. Hence, similar to previous step, we identify dependencies among properties/parameters of the ODD elements and identify a 'p' value, where 'p' is the maximum number of properties/parameters which can affect each other. In the above example, the amount of rainfall, time in PM/AM and density of puddles are not dependent on each other. However, the amount of the rainfall might affect the puddle water level. Hence, we can define 'p' as 2. When we generate 2-way combinations for the three properties, the number of combinations is reduced from 1760 to 176.

Step 4. Generate Test Cases for Simulation. Once the various instances that need to be considered for each scenario along with its corresponding goals/objectives are identified, test cases are generated. This is done by using simulation tools, where we can specify the property values, their range and associated increment. If such a functionality is not present, the test cases need to be created manually or we need to write a script which offers automated support to run simulation for various cases we want to evaluate. In DBCA approach, for the previous scenario that has 176 instances, we generate corresponding 176 test cases. A test case will fail if the scenario's goal/objective fails.

Step 5. Run Simulation. After the test cases for simulation are generated, we evaluate them in a simulation tool. If the objective defined is violated or there is a potential crash, the simulation tool will fail the test case. Based on these failures, we identify the potential causes for collisions and undesired behaviors and discuss corresponding solutions (e.g, design modification, enhancement of functions) among stakeholders including domain experts and engineers.

4 EMPIRICAL STUDY

To evaluate DBCA, we conducted an empirical study using Metamoto (Metamoto, 2020), a cloud-based simulation tool, with the focus on the following research questions:

RQ1. How efficient is DBCA over exhaustive simulation?

RQ2. Does reduction in the number test cases affect DBCA's ability to identify root causes of collisions over exhaustive simulation?

4.1 Object of Analysis

To analyze DBCA, we considered a list of scenario factors listed in Table B.3 in ISO 21448 (ISO/PAS, 2020) for defining ODD. The table contains values for 12 ODD elements such as climate (8 values), time of the day (4 values), road shape (16 values), feature of the road (8 values), condition of the road (6 values), lighting (5 values), ego vehicle's perception of condition (10 values), operation of the ego vehicle (14 values), surrounding vehicles (14 values), other road participants (4 values), objects in the surroundings (16 values), and objects on the road-way (11 values). The list of elements and their values is shown in Table 1. It can be observed that many of the values in the table can be further subdivided. For example, for ODD element "surrounding vehicle," we have "bicycle" as a value. However, a bicycle itself can be moving along the vehicle, crossing across the road, or coming in the opposite direction but in a bike lane. However, we only consider the values defined in Table B.3 in ISO 21448 to limit the scope of the analysis.

For test case generation, we focused on a vehicle cut-in scenario (illustrated in Figure 3), where a lead vehicle tries to cut-in to the path of the ego-vehicle, i.e., an autonomous vehicle being analyzed. The ego-vehicle we analyzed is a SAE level 2 system (Committee et al., 2014), which has a lane keeping assist and a cruise control. We have also considered a city street with passengers scenario for a SAE level 4 vehicle (Committee et al., 2014) with cameras and a LIDAR. However, for the level 4 system, the simulation tool gave time-out errors for most of the test cases. Upon analysis, we found that it was a simulation tool issue. Hence, we do not discuss results of a SAE level 4 vehicle in this paper. Rather we focus on the level 2 system. Table 2 shows a list of parameters we used for generating test cases for the level 2 system along with their ranges, step size (how much increment we have used between two values when generating test case), and the total number of values we considered for each of the parameter, respectively.

Figure 3: Example illustrations of cut-in cases possible within the highway cut-in scenario.

4.2 Variables

Independent Variable. The independent variable is the type of technique we used to generate operating environments and instances of scenarios.

We have one heuristic technique and one control technique as follows.

Heuristic: The heuristic technique we used in our study is our proposed DBCA approach, which uses IPOG (Lei et al., 2007; Lei et al., 2008), a combinatorial algorithm to reduce the number of combinations for operating environments as well as instances of a scenario.

Control: The control technique used in our study is the exhaustive technique where all possible combinations of values of ODD elements are considered for operating environments, and all possible combinations of property or parameter values are considered for generating instances of a scenario.

Dependent Variables. The dependent variables for RQ1 are time taken for creating operating environments and time taken for running instances of simulation. and for RQ2 is the number of root causes result-

Table 1: ODD elements and their values from Table B.3 of ISO 21448 (ISO/PAS, 2020).

ODD element	**Values**
Climate	fine, cloudy, rainy, sleet, snow, hail, fog, wind
Time of the day	early morning, daytime, evening, nighttime
Road shape	straight, curve, downhill, uphill, banked road, step difference, uneven spot, Belgian brick road, narrow road, wide road, median, manhole cover, tollgate, merging on the roadway, branching, pothole
Feature of the road	tunnel, underpass, bridges, skyways, cloverleaf, diamond, toll booth, gate
Condition of the road	dry, wet, low μ path, crossover road, water trough, gravel road
Lighting	direct sunlight, moon light, street lamp, backlight, twilight
Ego vehicle's perception of condition	irregular disturbance by sensor, variation of sensor, sensor fogged up, dirtied sensor, vehicle posture, vehicle situation (e.g,. towing), real vehicle weight, distribution of weight, tire, brake pad
Operation of the ego vehicle	accelerating, decelerating, driving at constant speed, stopping of the vehicle, drive at high speed, drive at low speed, making a turn, making a sudden traversing, passing, right or left turn, construction zone detour, approaching intersection, roundabout, crossing railroad track
Surrounding vehicles	position of surrounding car, preceding vehicle makes sudden deceleration, preceding vehicle makes deceleration, preceding vehicle makes acceleration, preceding vehicle makes a sudden acceleration, interrupting vehicle, trailing vehicle in stop and go traffic, vehicle to the right of ego vehicle going in same direction, vehicle to the left of ego vehicle going in same direction, oncoming vehicle, high beam from oncoming vehicle, a motor cycle passing by, bicycle, heavy interferences from other vehicles
Other road participants	pedestrian walking across, truck, three-wheeled motorcycle, peculiar vehicle
Objects in surroundings	sidewall, upside sign, side sign, pole, tunnel, multi-storey parking space, beneath a viaduct, kerb, guardrail, pylon, vehicle stopping on the side of the road, animal jumping out, railway crossing, construction site, marked crosswalk, water alongside road
Objects on road-way	reflectors, solid lines, dashed lines, cross walk, rumble strips, speed bumps, informational (e.g., arrow, speed limit), none, interrupted, degraded lane markings, multiple lane markings

ing in collision. Note that the number of root causes resulting in collision is different from the number of collisions detected. For example, let us consider an autonomous vehicle that has eight collisions detected, four due to incorrect sensor reading, and four due to heavy rain. In this case, while the number of collisions is eight, the number of root causes is only two.

4.3 Experimental Procedure

In our study, we utilized advanced combinatorial testing for software (ACTS) tool (Yu et al., 2013), which uses IPOG to generate combinations. For evaluating operating environments, creating scenarios and test cases, we used a proprietary simulation tool, which runs simulations on a cloud.

To conduct our study, we began with the ODD elements defined in Table 1. For the exhaustive technique, we calculated all possible combinations of ODD elements that need to be considered for operating environments. In the meanwhile, we also identified the dependencies among 12 ODD elements. We have considered both direct dependencies and indirect dependencies to identify the 't' value for generating t-way combinations of ODD elements with ACTS tool. For example, climate can affect the behaviors of sensors in the ego vehicle thereby affecting its perception of the condition, i.e., being unable to assess the correct situation which the ego vehicle encounters with respect to its operating environment. We consider this to be a direct dependency. Climate can also affect operation of the ego vehicle because of its impact on the ego vehicle's perception of its condition with respect to operating environment. Hence, we can consider climate and the ego vehicle's operation to have an indirect dependency. Based on the dependencies, we decided to vary t-value from 2 to 4 to see how significantly the number of operating environments might

Table 2: Parameters considered for test cases.

Parameter	Range	Step size	# of values
Road marking deterioration	0–1	1	2
Road wetness	0–1	1	2
Road puddles	0–1	1	2
# of random vehicles	0–20	5	5
Ego vehicle's start distance	500–2000	100	16

Table 3: Fixed Parameters and their respective values.

Parameter	Range	Fixed value
Density of clouds	0–1	0.2
Amount of rain	0–1	0
Lead vehicle's collision avoidance	0–1	0
Lead vehicle's start distance	500–2000	910
Initial lead vehicle's start distance	500-2000	880
Ego vehicle's collision avoidance	0–1	0

differ.

Once the operating environments analysis is complete, we performed scenario analysis. As mentioned earlier in our study, we focus on analyzing instances of a scenario in which a lead vehicle tries to cut into the path of the level 2 ego vehicle. Thus, we considered a simulation environment that meets the needs of the required operating environment. If the operating environment is not present then we should discuss with simulation engineers and create a corresponding environment to run simulation. The goal we defined for the scenario being simulated is for the ego-vehicle to reach a pre-defined location without collision. Since the simulation tool we used has a 800 jobs limit, i.e., we cannot submit more than 800 test cases at a time. Therefore, we fixed values of some parameters and varied values of other parameters. The fixed parameters and their corresponding values are shown in Table 3 and the range values for parameters are found in Table 2 along with the step size and total number of parameter values we used for each parameter. We chose larger step sizes because we believe that once we identify a problematic range for the parameter, we can focus on the specific range and use a smaller step size for the problematic range. This paves a path for hierarchical analysis rather than performing an exhaustive analysis. It can be observed from Table 3 that the start distance of the lead vehicle (the vehicle in front of the ego vehicle) is fixed to 910 and the start distance of the initial lead vehicle (the vehicle which is in front of all vehicles in the scenario) is fixed to 880. These values are chosen arbitrarily to see how the vehicles will be initialized if the ego vehicle's position is set to be before the lead vehicle in the test case. We chose both the lead vehicle's and the ego vehicle's collision avoidance as zero due to the following reasons: (1) we wanted to minimize the simulation runs due to resource constraints and job limits (the number of test cases that can be submitted at once, which is 800 in our case). Since we already have a randomized parameter "# of random vehicles", adding more randomness to the instances of scenarios such as making collision avoidance as 0.5, where a vehicle can collide another vehicle with a 50% probability (chosen randomly), will require more runs to gain sufficient confidence in the results. (2) by making collision avoidance of both ego vehicle and lead vehicle zero, we can identify if the DBCA technique can identify same collision causes as exhaustive technique even when vehicles are randomly initialized.

After fixing the parameter values, we generated an exhaustive set of test cases for simulation using the simulation tool. For the combinatorial analysis, we created combinations of parameter values using ACTS tool, and then created test cases in the simulation tool. For parameter values, we chose a 'p' value of 2 to generate p-way combinations. This is because the only parameters that have a partial dependency are road wetness and road puddles. We then ran these test cases in the simulation tool to identify collision cases. Note that, since we have some randomness present for each test case due to a parameter "random number of vehicles", we ran test suite for each approach 10 times. Once the results of test cases are collected, we analyzed the failed test cases to identify causes of collision and discussed with stakeholders about possible solutions and next steps. The results reported are the average of results found over 10 iterations of simulations.

4.4 Results

RQ1 Results. For RQ1, we analyzed the efficiency of DBCA over exhaustive simulation. For 12 ODD elements defined in Table 1, the total number of combinations that is required to be considered for creating operating environments for each technique is shown in Table 4. It can be observed from the table that the exhaustive technique generated 169,554,739,200 combinations for operating environments; the number of combinations is significantly larger than that of 2-way, 3-way or 4-way combinations generated using DBCA. Each of the combinations represents

Table 4: Number of combinations generated for operating environments using heuristic and control techniques.

Type	Technique	# of combinations
Heuristic	DBCA	266 (2-way), 4032 (3-way), 55075 (4-way)
Control	Exhaustive	169554739200

Table 5: Number of test cases generated (represented as # of TC) using heuristic and control techniques and time taken for running them in the cloud (represented as TT).

Type	Technique	# of TC	TT
Heuristic	DBCA	80 (2-way)	104 min
Control	Exhaustive	640	225 min

a potential operating environment and an environment corresponding to this combination needs to be checked for its presence in the simulation tool manually. If no such environment is present in the simulation tool, then the environment needs to be created manually, which can take from a few minutes to days depending on the complexity. Hence, we can conclude that DBCA requires less effort over the exhaustive technique.

Similarly, for the 5 parameter values in Table 2, the number of test cases produced using DBCA and the exhaustive approach and the approximate time taken for each of their iteration using the simulation tool are shown in Table 5. We can observe that DBCA reduces the number of test cases by 87.5% and reduces the time required for running simulations in cloud by ≈51% compared to the exhaustive approach.

RQ2 Results. For RQ2, we analyzed the number of root causes for collision identified for DBCA and the exhaustive approach, respectively. The results are shown in the Table 6. While we have 63 collision cases stemming from the results of 640 test cases for the exhaustive approach, and only 10 collision cases stemming from the results of 80 test cases for DBCA, the number of root causes for collisions remained same for both approaches. We found that the collision occurred irrespective of other parameter values when the ego vehicle's start distance is 800 and 1000. This is because of lack of sufficient distance between the lead vehicle and ego vehicle, there by resulting in collision. When the ego vehicle's starting distance is 900, then the collision is nearly missed, hence the test cases are passed. Although the number of collision cases reported for the exhaustive approach is 63, the expected count is 80. The remaining 17 are reported under the different category due to inherent problems in the simulation tool, where a surrounding vehicle's hard braking infraction can result in a failure of a test case, even if it is not related to an ego vehicle.

Table 6: Number of root causes for collisions found from the test cases generated using heuristic and control techniques.

Type	Technique	# of root causes
Heuristic	DBCA	2
Control	Exhaustive	2

4.5 Threats to Validity

First, the combinations generated by DBCA or the exhaustive approach can have feasible or unfeasible combinations. While this can be easily resolved by adding constraints to the tools while generating combinations, it is essential to analyze both feasible and unfeasible combinations as they might aid in uncovering unknown safety issues. This helps in reducing the unknown risks as required by ISO 21448 standard. Second, random initialization of agents or vehicles in a scenario can affect the results. To reduce the effect of randomization, we ran each test suite 10 times using the simulation tool, and averaged the results. Third, our results are based on Metamoto simulation tool. So, they might not be generalized to other industrial tools or open source tools. Nevertheless, the insights we provide based on the results can be useful for other simulation tools that are intended for autonomous vehicles.

5 DISCUSSION

From the results of our study, we found that DBCA approach reduces the effort and time for analysis in simulation tools over exhaustive approach. We also found that we can identify the same number of root causes for collisions as exhaustive approach using DBCA, making it reliable to use for generating test cases for simulation for autonomous vehicles.

Importance of 't' and 'p' Value in DBCA: While combinatorial approach is found to be useful, it can be effective only if we choose the right 't' and 'p' values for generating combinations. For combinatorial testing of software applications it is found that 2-way interactions can expose approximately 62% to 97% of the faults, 3-way interactions can expose 87% to 99% of the faults, and 4-way interactions can expose 96% to 100% of faults. However, whether such is the case with safety-critical and complex systems such as autonomous vehicle is yet to be verified.

To identify the right 't' value used to generate t-way combinations of ODD elements, it is essential to perform a thorough analysis of dependencies among ODD elements. Missing dependencies can result in missing some important combinations thereby re-

sulting in overlooking some operating environments, which can increase the chances of facing unknown risks for autonomous vehicles. Similarly, 'p' value used to generate p-way combinations of properties of elements, i.e., parameters for simulation, must be chosen based on dependencies among parameters. Missing dependencies among properties of the elements considered for a scenario can result in missing critical test cases, thereby overlooking safety issues. A potential future direction to explore in this area is to study how percentages of collision causes that can be found vary with increment in 't' and 'p' values, and if the required 't' an 'p' values are less than the ones defined by identifying dependencies among ODD elements, and properties/parameters of scenario respectively.

Insights on Simulation Tools for Autonomous Vehicles: During the analysis of our results, we found that it is important to ensure proper working of simulation tools before running simulations. We found that often in the case of the Metamoto tool we used in our study, even though we are focusing on identifying collision of the ego vehicle, a randomly initialized vehicle's hard brake infraction also resulted in failure of test cases. Moreover, the simulation tool does not generate any metrics for ODD elements coverage based on operating environments we define or scenario coverage for each operating environment. Such metrics should be considered as a part of simulation tool.

A major finding we identified for the simulation tool is that there is not effect of randomly initialized vehicles on the results of our test cases. We analyzed the results further to understand why we have such an unexpected behavior among the results. We found that simulation tools by default assume vehicles when being initialized follows the traffic rules and maintain minimal separation. However, this should be left to the user, as a separate mechanism to specify scenario rules or simulation rules, as assuming vehicles always follow rules can result in overlooking unknown scenarios, there by increasing unknown risks for an autonomous vehicle.

Practical Implications for Industry: Using DBCA, automotive engineers and safety analysts can identify the various operating environments that need to be created as a part of simulation for analysis of autonomous vehicle's behaviors with less effort and time. This also helps in simplifying the complexity of hazard and risk assessments performed as a part of FuSa and SOTIF analyses, where we need to consider various possible scenarios a vehicle can be in. Similarly DBCA can also reduce the effort and time required for running simulation by optimizing the number of test cases while simultaneously ensuring all dependencies among the properties/parameters are satisfied.

Limitations: Despite the advantages of DBCA approach, it suffers from limitations. First, the identification of dependencies among ODD elements and among properties/parameters used for generating instances of scenarios is a manual process. We can address this limitation by automating the process using property relation tables, and generating inferences from the tables. Another limitation is that we did not define any systematic procedure for defining goals or objectives for scenarios in this paper. We plan to address this by suggesting various goals and objectives that can be used for a scenario based on its ODD elements and properties by using a recommender system.

6 CONCLUSION AND FUTURE WORK

In this paper, we proposed dependency-based combinatorial approach (DBCA) to reduce the effort for identifying the various operating environments and various instances for a scenario in an operating environment for autonomous vehicles. DBCA utilizes the combinatorial algorithm IPOG (Lei et al., 2007; Lei et al., 2008) to generate t-way combinations of ODD elements to generate combinations that represent operating environments, and p-way combinations of properties (or parameters) of elements chosen for a scenario, where 't' and 'p' values are defined by identifying dependencies among ODD elements and properties of elements, respectively. To evaluate DBCA approach, we conducted an empirical study by considering ODD elements listed in Table B.3 of ISO 21448 (ISO/PAS, 2020) and a highway cut-in scenario, and compared DBCA with the exhaustive technique, which generates all possible combinations of values of ODD elements to generate operating environments, and all possible combinations of values of properties to generate instances for a scenario. We found that DBCA reduces the time requires for running simulations significantly up to 51% without affecting the number of collision causes identified.

As a part of future work, we plan to address the limitations discussed in Section 5 and incorporate ethical assessment of safety into our approach based on industrial standards such as ISO/AWI 39003 (ISO/AWi, 2021) We also plan to define dependency-based coverage criteria that aid in analyzing whether the engineers overlooked any of the operating environments or test cases for scenarios. We also plan to conduct a comprehensive study

for various scenarios using multiple simulation tools (both industrial and open source) and study the effectiveness of DBCA approach under a broader context.

REFERENCES

Akagi, Y., Kato, R., Kitajima, S., Antona-Makoshi, J., and Uchida, N. (2019). A risk-index based sampling method to generate scenarios for the evaluation of automated driving vehicle safety*. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 667–672.

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

BSI/PAS (2020). 1883:2020. *Operational Design Domain (ODD) taxonomy for an automated driving system (ADS) – Specification*.

Calò, A., Arcaini, P., Ali, S., Hauer, F., and Ishikawa, F. (2020). Generating avoidable collision scenarios for testing autonomous driving systems. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 375–386.

Committee, S. O.-R. A. V. S. et al. (2014). Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems. *SAE Standard J*, 3016:1–16.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). Carla: An open urban driving simulator. In *Conference on Robot Learning*, pages 1–16.

Duan, J., Gao, F., and He, Y. (2020). Test scenario generation and optimization technology for intelligent driving systems. *IEEE Intelligent Transportation Systems Magazine*.

Fadaie, J. (2019). The state of modeling, simulation, and data utilization within industry: An autonomous vehicles perspective. *CoRR*, abs/1910.06075.

Fortellix (2020). Open M-SDL, Measurable Scenario Description Language.

Gannous, A. and Andrews, A. (2019). Integrating safety certification into model-based testing of safety-critical systems. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 250–260. IEEE.

ISO (2018). 26262. *Road Vehicles – Functional Safety*.

ISO/AWi (2021). 39003. *Road Traffic Safety (RTS) — Guidance on safety ethical considerations for autonomous vehicles*.

ISO/PAS (2020). 21448. *Road vehicles - Safety of the intended functionality*.

Kalra, N. and Paddock, S. M. (2016). Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice*, 94:182 – 193.

Koopman, P. and Wagner, M. (2016). Challenges in autonomous vehicle testing and validation.

Kuhn, D. R., Kacker, R. N., and Lei, Y. (2010). Practical combinatorial testing. *NIST special Publication*, 800(142):142.

Lei, Y., Kacker, R., Kuhn, D. R., Okun, V., and Lawrence, J. (2007). Ipog: A general strategy for t-way software testing. In *14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, pages 549–556. IEEE.

Lei, Y., Kacker, R., Kuhn, D. R., Okun, V., and Lawrence, J. (2008). Ipog/ipog-d: efficient test generation for multi-way combinatorial testing. *Software Testing, Verification and Reliability*, 18(3):125–148.

Li, Y., Tao, J., and Wotawa, F. (2020). Ontology-based test generation for automated and autonomous driving functions. *Information and Software Technology*, 117:106200.

Metamoto (2020). Simulation as a service for autonomous systems.

Mullins, G. E., Stankiewicz, P. G., Hawthorne, R. C., and Gupta, S. K. (2018). Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles. *Journal of Systems and Software*, 137:197 – 215.

NHTSA (2018). Fatality Analysis and Reporting System (FARS) Data Tables.

Nonnengart, A., Klusch, M., and Müller, C. (2019). Crisgen: Constraint-based generation of critical scenarios for autonomous vehicles. In *International Symposium on Formal Methods*, pages 233–248. Springer.

Rocklage, E., Kraft, H., Karatas, A., and Seewig, J. (2017). Automated scenario generation for regression testing of autonomous vehicles. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 476–483.

van Nes, N., Bärgman, J., Christoph, M., and van Schagen, I. (2019). The potential of naturalistic driving for in-depth understanding of driver behavior: Udrive results and beyond. *Safety Science*, 119:11 – 20.

VTTI (2020a). Canadian NDS Data.

VTTI (2020b). SHRP2 NDS Data.

Wotawa, F. (2017). Testing autonomous and highly configurable systems: Challenges and feasible solutions. In *Automated Driving*, pages 519–532. Springer.

Yu, L., Lei, Y., Kacker, R. N., and Kuhn, D. R. (2013). Acts: A combinatorial test generation tool. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 370–375. IEEE.