Feature-Interaction Sampling for Scenario-based Testing of Advanced Driver Assistance Systems*

Lukas Birkemeyer
l.birkemeyer@tu-braunschweig.de
Technische Universität Braunschweig
Braunschweig, Germany

Tobias Pett
t.pett@tu-braunschweig.de
Technische Universität Braunschweig
Braunschweig, Germany

Andreas Vogelsang
vogelsang@cs.uni-koeln.de
University of Cologne
Cologne, Germany

Christoph Seidl
chse@itu.dk
IT University of Copenhagen
Copenhagen, Denmark

Ina Schaefer
i.schaefer@tu-braunschweig.de
Technische Universität Braunschweig
Braunschweig, Germany

ABSTRACT

Scenario-based testing is considered state-of-the-art to verify and validate Advanced Driver Assistance Systems. However, two essential unsolved challenges prevent the practical application of scenario-based testing according to the SOTIF-standard: (1) how to select a set of representative test scenarios, and (2) how to assess the effectiveness of a test scenario suite. In this paper, we leverage variability modelling techniques to select scenarios from a scenario space and assess the resulting scenario suites with a mutation score as metric. We capture the scenario space in a feature model and generate representative subsets with feature-interaction coverage sampling. The mutation score assesses the failure-finding effectiveness of these samples. We evaluate our concepts by sampling scenario suites for two independent Autonomous Emergency Braking function implementations and executing them on an industrial-strength simulator. Our results show that the feature model captures a scenario space that is relevant to identify all mutants. We show that sampling based on interaction coverage reduces the testing effort significantly while maintaining effectiveness in terms of mutation scores. Our results underline the potential of feature model sampling for testing in the automotive industry.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

Sampling Strategies, Scenario-based testing, ADAS, SOTIF

ACM Reference Format:
Lukas Birkemeyer, Tobias Pett, Andreas Vogelsang, Christoph Seidl, and Ina Schaefer. 2022. Feature-Interaction Sampling for Scenario-based Testing of Advanced Driver Assistance Systems. In *Proceedings of the 16th International Working Conference on Variability Modelling of Software-Intensive Systems (VAMOS '22), February 23–25, 2022, Florence, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3510466.3510474

*This work has been partially been funded by the PhD program "Responsible AI in the Digital Society" funded by the Ministry for Science and Culture of Lower Saxony.

VAMOS '22, February 23–25, 2022, Florence, Italy

1 INTRODUCTION

In recent years, Advanced Driver Assistance Systems (ADASs) have become an essential part of modern motor vehicles. Verification and validation of Advanced Driver Assistance Systems / Autonomous Driving (ADASs/AD) have to ensure increasing reliability for increasing autonomous functionality. ADAS are software components so that established software engineering methods have potential to improve their testing process. While real-world tests require a significant effort, both in terms of time and money, simulations reduce effort during the testing process. Simulation tools such as Carla [18] and Gazebo[1] stem from the open-source community, while e.g., CarMaker by IPG Automotive GmbH [23] and VTD by VIRES Simulationstechnologie GmbH [55] are simulation tools from industry. Based on simulations, scenario-based testing is a common approach to verify and validate ADAS/AD [12, 13, 26, 39, 49, 53]. A scenario [53] is a sequence of scenes, which describe the environment of a vehicle, e.g., road design, road users, or weather [6]. ISO 26262 is an essential standard within the automotive industry, which addresses functional safety of motor vehicles and avoids design faults [25]. The Safety of the intended functionality (SOTIF) standard (ISO 21448) focuses on failures that are not directly traceable to design faults but to a vehicle's environment [26, 34]. SOTIF employs scenarios as a basis and classifies scenarios into the classes *known* or *unknown* and into the classes *safe* or *unsafe*. Unsafe scenarios are of particular importance for the testing process of Highly Autonomous Vehicles (HAVs). However, SOTIF does not specify criteria to select scenarios, leaving test engineers to choose scenarios based on intuition. In addition, there is no metric to assess scenarios with respect to their suitability of testing and their effectiveness for failure-finding.

In this paper, we show that Software Product Line (SPL) engineering techniques such as feature modelling and sampling are not only beneficial for development of multi-variant products [4], but also for testing of ADAS/AD in the automotive industry. This study is based on the master's thesis [10]. We leverage variability modelling techniques to select scenarios from a scenario space. We apply sampling strategies from SPL engineering to generate

[1] http://gazebosim.org

scenario suites. Sampling strategies have potential to significantly reduce test effort for highly configurable systems by focussing on subsets that represent all possible configurations. Following this concept, we consider a scenario as a configuration of a simulation tool (e.g., CarMaker). The set of all possible scenarios constitutes the configuration space. In SPLs, the configuration space is commonly represented by a Feature Model (FM) [4]. We create an FM that represents the scenario space and use sampling strategies to generate sets of configurations to represent the overall configuration space of the FM [29, 30]. These sampling strategies aim for coverage of feature interactions relevant for practical testing representative subsets of the configuration space. A representative subset of configurations has similar properties as the overall configuration space. In contrast to existing approaches [2, 6, 12, 13, 27], the scenario selection using FM sampling is suitable to reduce the number of test scenarios. In evaluation experiments, we generate scenarios that trigger the Autonomous Emergency Braking (AEB) of a vehicle. We assess the resulting scenario suites with a mutation score as metric. Mutation testing [17, 28] is a common concept to assess test sets by seeding errors into the System Under Test (SUT), resulting in mutants, and performing tests on the mutants. Mutation testing has potential to effectively asses the quality of a test set in the sense of its ability to detect failures in a SUT. The mutation score indicates how many mutants are detected by a test set. The higher the mutation score, the stronger is the test set. We apply mutation testing to determine the ability of scenarios to detect failures and assess the strength of scenario sets.

In summary, we contribute the following:

- We apply variability modelling and management techniques to scenario-based ADAS testing:
  - We show that an FM is a suitable modelling formalism for configuration spaces of simulation tools.
  - We show that FM sampling is suitable to scenario-based ADAS testing by comparing coverage-based and random sampling with expert strategies.
  - We assess scenario suites with mutation testing and show strengths of FM sampling.
- We provide a prototypical implementation of our generic concepts in a tool chain using CarMaker as simulation tool and Simulink for ADAS functionality.
- We evaluate our concepts for scenario selection and scenario assessment in experiments using two AEB implementations. In our evaluation, we show that feature-interaction coverage sampling reduces the testing effort significantly while maintaining effectiveness in terms of mutation scores.

2 STATE-OF-THE-ART

In this section, we discuss work relevant to scenario generation and scenario assessment.

Scenario Generation. We identify three essential approaches for scenario generation: optimization-based, data-driven, and combinatorial scenario generation.

Optimization-based Scenario Generation. Optimization-based scenario generation approaches focus on a mathematical (non-linear) optimization, e.g., of a metric that rates how critical a scenario is [2, 8, 32, 33, 51, 52]. There is also search-based scenario generation [9, 22] using evolutionary algorithms. These approaches are used to generate scenarios considering a specifically tailored fitness function to test a specific SUT (e.g., pedestrian detection). These approaches generate scenarios that are critical according to the specification of the SUT encoded by the fitness function. The set that covers all possible scenarios is explicitly not modelled.

Data-Driven Scenario Generation. Data-Driven scenario generation approaches generate scenarios from existing scenario data [45]. This data may be information from real-world scenarios [41] or police reports [21]. Current approaches use machine learning [27] or stochastic parameterisation [16, 58] to generate scenarios. These approaches are restricted by available data and do not explicitly structure knowledge or configurability of scenarios. Thus, per concept, they do not allow to generate sets of scenarios that cover an overall scenario space exceeding available data or variations of it.

Combinatorial Scenario Generation. Combinatorial concepts generate or select scenarios based on a model that describes a scenario space. Rocklage et al. [47] introduce combinatorial interaction testing coverage using classical optimization by solvers for trajectory generation. Based on Schuldt [49], Bagschik et al. [7] develop a five-level model to structure knowledge of scenarios. Bagschik et al. [7] and Ulbrich et al. [54] use an ontology to model a scenario space. Fremont et al. [20] propose SCENIC, a probabilistic programming language to generate test scenes. They select scenes from scenarios with rejection sampling. To summarise, existing combinatorial scenario generation approaches do not generate sets of scenarios that satisfy a specific coverage criterion.

In this paper, we address the SOTIF scenario-based, black-box testing concept by generating scenarios independently of a specific SUT. We structurally model a scenario space and combinatorically select scenarios coverage-based. Instead of an ontology, we focus on an FM to model the scenario space which is suited for scenario generation with coverage-based sampling.

Assessment of Scenarios. Distance-based evaluation of scenarios considers statistically which critical scenarios occur in the real world [57]. However, testing effort is immense. Other approaches for scenario assessment define a criticality metric, e.g., by their driveable area [33] or accident velocities [27]. These criticality metrics focus on specific SUTs (e.g., trajectory planning); thus, they are not suited for a scenario-based black-box testing concept demanded by SOTIF.

In this paper, we use mutation testing to assess scenario suites in accordance with SOTIF. Mutation testing is a generic concept that rates the ability of detecting failures in an SUT.

3 BACKGROUND

In this section, we introduce variability modelling in the form of feature modelling and sampling as basis. We also establish the concept of mutation testing and terminology to describe scenarios.

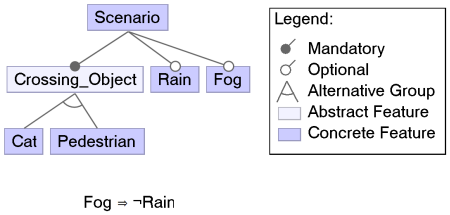

Figure 1: FM implementation for scenario generation

3.1 Variability Modelling

Feature modelling. Feature modelling is a common approach to describe variability in SPLs [4, 11, 31, 48, 50]. In a Feature Model (FM), features are arranged as nodes in a tree structure. Apel et al. [4] define a feature f as a behaviour of a system that is visible to the end-user. F describes the set of all features f. In the FM, features may be *mandatory* or *optional*. An *alternative* or an *or*-group permit selection of exactly one or at least one of the contained features, respectively. *Cross-tree constraints* express dependencies between features, which cannot be modelled as a tree structure. The FM represents the configuration space of a variable system, the related solution space is implemented as so-called 150%-model, i.e., a representation of the implementation that encompasses all possible variations. A valid configuration c is a selection of features that is permitted by the FM. In the solution space, a configuration serves as input for pruning the 150%-model to a 100%-model.

In Figure 1, we present an example FM that configures scenarios in the context of scenario-based testing [7, 49, 53]. In the example, there are the features *Crossing_Object*, *Cat*, *Pedestrian*, *Rain*, and *Fog*. Two valid example configurations are $c_1 = \{Crossing\_Object, Cat, Fog\}$ and $c_2 = \{Crossing\_Object, Cat, Rain\}$. In Figure 1, the cross-tree constraint $Fog \Rightarrow \neg\ Rain$ defines that, if the feature *Fog* is selected, the feature *Rain* must not be selected.

Sampling. The configuration space of an FM becomes large so that testing of all valid configurations becomes infeasible due to combinatorial explosion [19, 37]. In SPL engineering, so-called sampling strategies are used to select a representative sample which is a set of configurations. Uniform random sampling [42, 43] leads to an unbiased sample by selecting configurations that are uniformly distributed in the configuration space. Other sampling strategies focus on a Feature-Interaction Coverage (FIC) of T features which has been shown to be effective and efficient for failure detection [30, 36]. FIC-sampling generates samples that contain configurations with each combination of T features, e.g., for two-wise sampling ($T = 2$), each combination of two features occurs at least once in the sample. State-of-the-art FIC-sampling strategies are IncLing [1], ICPL [30], Chvatal [29], and YASA [35].

3.2 Mutation Testing

Mutation testing is a software engineering method to evaluate the quality of a test suite regarding effectiveness to discover failures [17, 28]. In mutation testing, errors are injected into a program P to generate so-called mutants $P_M = \{P_{M_1}, P_{M_2}, ..., P_{M_n}\}$. The quality of a test suite is determined by the number of mutants that are detected by the test suite. If a mutant does not pass the tests, it is classified as *killed*. If a mutant passes the tests, it is classified as *alive*. The kill criterion is derived from the specification of P as defined in the test oracle of a test t. The mutation score MS is the ratio of killed mutants P_{M_k} by the number of all mutants P_M. Thus, the mutation score is defined as $MS \in [0, 1]$. The higher the mutation score, the stronger is the test suite regarding its ability for detecting failures in an SUT.

3.3 Scenario Description

Ulbrich et al. [53] define the terms *scene* and *scenario* within the context of automotive testing: "A scene describes a snapshot of the environment including the scenery and dynamic elements [...]" [53]. The scenery of a scene contains static elements such as the road network, traffic lights, or crash barriers. Dynamic elements are, e.g., road users or obstacles that move. A sequence of scenes results in a *scenario* [53]. Bagschik et al. [6] follow this definition and structure the description of a scenario into five levels which we use to structure the FM. The first three levels define the road network. Level 1 (E1) describes the road itself, e.g., the route and surface. Level 2 (E2) defines the infrastructure such as traffic signs. Level 3 (E3) describes temporary adjustments of E1 and E2, e.g., construction sites. Level 4 (E4) describes dynamic and movable objects. Level 5 (E5) defines environmental conditions such as weather.

As a running example to motivate challenges of scenario-based testing and to illustrate our solutions, we introduce a concrete scenario: a vehicle drives through a city and a pedestrian crosses the street directly in front of the vehicle. The road is a straight road with asphalt (E1). The road has a sidewalk, guiding lines, and there are buildings next to the sidewalk (E2). No temporary adjustments are defined (E3). A yellow vehicle (ego-vehicle) drives on the road, and a pedestrian walks on the sidewalk. The pedestrian crosses the street directly in front of the yellow vehicle (E4). In the scenario, there is daylight and no rain (E5). The scenario is designed to trigger the AEB of the yellow vehicle. An AEB function identifies potential collisions and warns the driver. If the driver does not react, the AEB automatically performs a deceleration of up to $-10m/s^2$ to avoid, or at least to mitigate, a collision [5].

4 SCENARIO SELECTION AND ASSESSMENT

In the following, we introduce our concept for scenario modelling with variability modelling techniques and assessment of scenario suites with mutation score as metric. Figure 2 provides an overview of the process. Scenario modelling (4.1) is based on an FM (5.1.1) that we use to represent a scenario space. From this FM, we generate configurations by FM sampling (5.1.2) and transfer them into scenarios that are executable in a simulation tool (5.4). In Section 4.1, we provide detailed insights into our concept for designing an FM for scenario generation. The assessment of scenario suites (4.2) is based on mutation testing. We generate mutants (5.2) and use them as SUT in the simulation (5.4). We define a safety envelope (5.3) and derive a kill criterion for mutants from it. To assess the effectiveness of the generated scenario suites, we determine the mutation score (5.5) based on simulation results. In Section 4.2, we describe our ideas for scenario assessment.

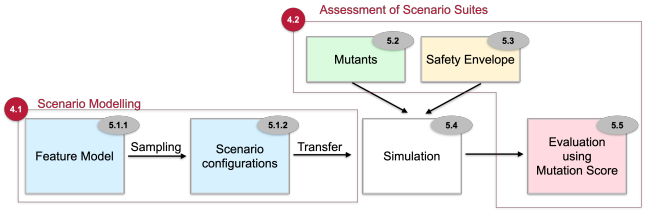

Figure 2: Process: Scenario modelling and assessment

4.1 Scenario Modelling

To capture a scenario space in an FM, we consider a scenario as a configuration of a simulation tool and model configurable elements as features. Thus, we assume that the simulation tool provides a realistic scenario space. This scenario space is finite, but extremely large. The selection of scenarios is an essential challenge because (1) each scenario has to be semantically valid and (2) the set of scenarios has to be representative. A scenario is semantically valid, if the scenario is (a) physically possible, (b) in accordance with traffic regulations, and (c) compilable in the simulation tool. Inspired by generative model-based software development [15], we devise a *template-based* FM design: A template is a fixed scenario with configurable parts that captures solution-space variability. The FM captures the problem-space with only valid scenarios.

Template-based Scenario Modelling. Scenarios that are relevant for the same SUT have a lot of similarities. Considering the running example, we might use the same road network to generate different scenarios where, e.g., the trajectory of a pedestrian or weather conditions differ. Scenarios have elements that are closely linked to each other and others that are independent. The temperature, e.g., does not affect the course of the road. However, the course of the road affects the trajectories of all road users. We model similarities and closely linked elements as non-configurable elements in the template. In contrast, we define an element as configurable if it is independent of other elements and freely configurable.

Following the five-level interpretation of scenarios according to Bagschik et al. [6], we consider elements of the scenario levels E1–E3 as non-configurable. These elements are often closely linked to each other and build the basis of the road network. This, in turn, affects elements of the levels E4–E5. Some elements of the scenario levels E4 and E5 are closely linked to elements of E1–E3. We consider those closely linked elements as non-configurable. In the running example, we define the position of a guiding line (E2) and start position of the road users (E4) as non-configurable because they are closely linked to the road course (E1). Examples for configurable elements of the example scenario are the type of the crossing object (E4), the trajectory of the crossing object (E4), or the weather (E5). For configurable elements, we consider presence or absence of the element in the scenario, but identify that configurable elements have parameters. We parameterise, e.g., the speed of the ego-vehicle. Some of these parameters are continuous.

To permit using FIC-sampling to select representative scenarios from one FM, we integrate templates, non-configurable elements, and parameters in the same FM. A scenario then is a configuration of the FM. We define a scenario $S = \{f_1, f_2, \ldots, f_n \mid f_i \in F \text{ and } i \in \{1, \ldots, n\}\}$ as a configuration of features f_i of F. Generating scenarios from the FM with all valid sampling results in a set of all possible scenarios $ScAll \subseteq \mathcal{P}(F)$. A scenario suite $ScSu \subseteq ScAll$ is a subset of all possible scenarios.

Scenario Feature Model. Simulation tools have numerous configuration options. Capturing all scenarios that can be simulated with a specific simulation tool leads to a large FM. Many of its configurations are not relevant for a specific SUT. We focus on relevant parts and reduce the size of the FM (i.e., complexity) by eliminating irrelevant parts.

We devise a concept to generically structure FMs to capture a template-based scenario space. Inspired by the five-level interpretation of scenarios according to Bagschik et al. [6], we structure an FM in three main subtrees: *Template*, *E4*, and *E5*. In subtree *Template*, we model each template as a *template feature* to select between different templates. In the subtree *E4*, we model all configurable elements of the scenario that belong to level E4.[2] In the subtree *E5*, we model configurable elements that belong to level E5.[3] Each configurable element is represented by a *scenario feature*. To simplify sampling by reducing the complexity that comes with continuous parameters, we define equivalence classes for parameters and add *parameter features* instead of using feature attributes [14, 50]. We use parameter features in the subtrees *E4* and *E5*.

A template requires specific configurable elements of *E4* and *E5*. We use cross-tree constraints to guarantee semantical validity of a scenario. Each valid configuration of the FM contains one template feature, multiple scenario features, and possibly multiple parameter features. Figure 3 presents an FM representing the scenario space of our running example. In addition to the generic structure, it is worth noting that the constraint $Simple\_Crossing \Rightarrow Crossing\_Obj \wedge Crossing\_Path$ ensures complete specification of the template *Simple_Crossing*. The constraint $Rain \Rightarrow \neg\, Fog$ stems from physical restrictions.

Scenario Selection. We select a representative sample of semantically valid scenarios from the FM with coverage-based FIC-sampling strategies. According to the template feature, we take the selected template as basis. We supplement the template with configurable elements and parameterise those with parameter features of the configuration. For the running example, a configuration, e.g., contains the template feature *Simple_Crossing*, the scenario features *Pedestrian_Male_Casual_01_IPG*, *Crossing_Path*, and *Speed*, and the parameter feature *Speed_50*.

4.2 Assessment of Scenario Suites

According to the SOTIF standard, a strong scenario suite reveals unintended behaviours of an SUT, which we define as the failure-finding capability of the scenario suite. Mutation testing [17, 28] is a method used in software testing to evaluate a test suite according to its failure-finding capabilities with a mutation score.

Kill Criterion. A challenge in mutation testing is the definition of a kill criterion for detecting mutants. Inspired by back-to-back testing [56], we compare the outputs of the original SUT to the

[2]Note: Scenario level E4 represents dynamic and movable objects.
[3]Note: Scenario level E5 represents environmental conditions.

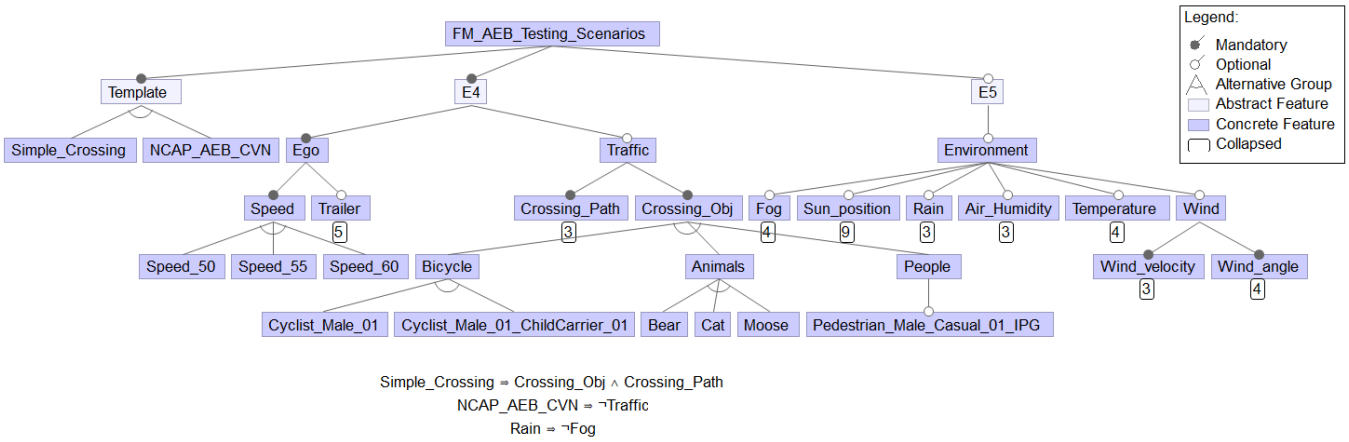

Figure 3: Example FM for scenario generation for AEB-system

outputs of the mutants. We consider the original SUT as correct, so that we classify a mutant as killed if its output is not the same as the original SUT, otherwise as alive. Considering all mutants that do not equally behave like the original as killed is a strict criterion, which we will relax in the next paragraph.

Safety Envelope. In ADAS/AD, modified behaviour of mutants with respect to the original system might be acceptable as long as the overall vehicle performs as intended. An error in an ADAS/AD-software component, e.g., may lead to a delayed, but more abrupt braking behaviour. Although the behaviour is changed, the overall vehicle still grinds to halts without collision. Thus, the error affects the comfort for the passengers of a car. From a safety point of view, however, this error is not relevant, but the Equal Behaviour (EB) kill criterion, as explained above, classifies this mutant as killed.

Since the SOTIF standard focuses on safety-relevance, we devise a Safety Envelope Controller (SEC), to capture this notion [40], and accept modified behaviour as long as it is not safety-relevant. As input, the SEC monitors parameters of the SUT (e.g., speed, detected objects, or steering angle) and its environment (e.g., weather, street conditions, or road users). Using the SEC as kill criterion, a challenge is the definition of undesired incidents. We arrange detectors for these undesired incidents in a fault-tree structure [38] and combine them with Boolean operators to a single output which indicates safety-relevance. Considering our example, we define a collision as an undesired incident and implement a collision detector.

5 IMPLEMENTATION AND TOOL SUPPORT

We have implemented our generic concepts prototypically in a tool chain to support our experimental evaluation. Our tool chain is publicly available to support replicability and further research activities[4]. The scenario generation process (5.1) uses template-based scenario suite generation to produce executable scenarios. We combine ADAS-mutants (5.2), SEC (5.3), and the scenario suites to test cases and execute them in the simulation tool (5.4). We assess scenario suites according to the mutation score (5.5). In the following, we describe the tools that we use to realise this workflow.

[4]https://doi.org/10.5281/zenodo.5422113

(5.1) Scenario Generation. We use FeatureIDE v3.6.3 [31, 50] to design an FM that represents a scenario space and to generate configurations from it. FeatureIDE is one of most established variability modelling tools and includes all sampling strategies required to evaluate our concepts. To transfer configurations from FeatureIDE into executable simulation files for a simulation tool, we implemented the Java tool C_2S.

(5.2) Mutant Generation. To evaluate our concepts, we implemented the SUT in Simulink – a widely applied tool in the automotive industry for model-based software development [3]. The generation of mutants is based on SIMULTATE by Pill et al. [44], which is open source and adapted by Mevenkamp [40]. We extract mutants with a Python-script by activating mutation operators.

(5.3) Safety Envelope Controller. We implement the SEC as Simulink model to reuse the interface of the SUT to communicate with the simulation tool. We separate the SEC from the ADAS-model to create a kill criterion that is independent of the SUT.

(5.4) Simulation. Within the automotive industry, there are a number of simulation tools for ADAS testing [23, 24, 55]. We focus on the simulation tool CarMaker [23] by IPG Automotive GmbH, which is widely used in the automotive industry. The CarMaker TestManager automatically executes and assess simulation scenarios. CarMaker also provides interfaces to connect Simulink models.

(5.5) Scenario Assessment. During the simulation, we log simulation results for each scenario. We determine the mutation score with the tool R [46].

6 EVALUATION AND DISCUSSION

In this section, we describe experiments that we perform with our prototypical implementation.

6.1 Research Questions

RQ1. As described in Section 4.1, we use an FM to represent a scenario space. We investigate: **RQ1: Does the FM capture the scenario space that is relevant to kill all mutants?** The objective is to determine the mutation score for all scenarios of the scenario space. If the mutation score is equal to one, the FM captures

scenarios that are relevant to kill all mutants. If the mutation score is less than one, the scenario space might be incomplete regarding relevant scenarios. FMs might not be suitable to represent scenario spaces or the abstraction of the FM may be too coarse in the sense that relevant variations of scenarios may not be represented.

RQ2. We generate scenario suites that represent the scenario space based on FIC-sampling to reduce test effort by reducing the number of test scenarios. To evaluate the quality of the scenario selection, we investigate research question **RQ2: What is the impact of sampling strategies on test effectiveness?** The objective of RQ2 is to compare measured mutation scores for independently generated scenario suites with various strategies. We determine if, for five runs each, the mutation scores' median is higher and its spread is smaller which would be deemed as a more effective scenario suite. The following three comparisons are of special interest: (1) We generate scenario suites with different sampling algorithms and degrees of FIC. We investigate their effectiveness in terms of mutation scores regarding scenarios from all valid sampling for RQ1. (2) We compare the FIC-sampled scenario suites with randomly sampled scenario suites. Consequently, we investigate the effectiveness of systematic scenario selection regarding unsystematic scenario selection. (3) We challenge sampling strategies against a scenario selection by expert knowledge to compare our concept with common evaluation methods of the automotive industry.

RQ3. As described in Section 4.2, we propose an SEC as kill criterion to assess scenario suites independently of an SUT. We define research question **RQ3: What is the influence of the SEC as kill criterion on the mutation score?** The objective of RQ3 is to compare measured mutation scores for generated scenario suites in combination with EB and SEC kill criterion. We determine if, for five runs, the mutation scores qualitatively performs similarly in terms of mutation scores.

6.2 Study Design

We designed an experiment to evaluate our concepts along the workflow presented in Figure 2. We use an AEB function as exemplary SUT. AEB functions are essential parts of modern vehicles and some implementations are publicly available. We use two independent AEB implementations to reduce a potential bias of one concrete implementation. As `aeb-1`, we use an implementation inspired by Arcidiacono [5]. The implementation of `aeb-2` is based on the Autonomous Driving Toolbox by Mathworks[5], which we adapted for CarMaker. For each AEB implementation, we generate sets of mutants M_{aeb-1} and M_{aeb-2} by randomly activating one or two mutation operators. We use SIMULTATE [44] to insert potential mutation operators before or after each block of the AEB Simulink model. We insert mutation operators of the types: *Absolute_mut*, *Zero_fault_mut*, *Inverter_mut*, *Negation_mut*, *Increment_mut*. Random activation of mutation operators reduces potential bias in the mutation process. Each set M_x contains 49 mutants. We consider the SUT implementation and their mutations as controlled variables.

We created an FM that represents the scenario space of relevant scenarios for the AEB. The resulting FM is an expansion of the FM in Figure 3 and contains seven templates. The templates relies on seven initial scenarios which are implemented and provided by IPG Automotive GmbH as scenarios for CarMaker to test AEB. Three of these scenarios are inspired by EuroNCAP[6] and are commonly used within the automotive industry to assess safety of AEBs. We identify configurable elements of the templates and model them as features and their dependencies as cross-tree constraints. This results in a FM containing 76 features and 10 cross-tree constraints. We use the scenario suite as input for a simulation with the simulation tool CarMaker. We simulate each scenario for all mutants for aeb-1 and aeb-2 as SUTs. We calculate the mutation score for the union of a scenario suite with Equation 1. The mutation score for each mutation set M is the dependent variable in our experiments. For **RQ1**, we generate the scenario suite $ScSu_{all}$, which contains 24,412 valid scenarios, with all valid sampling of FeatureIDE.

$$MS_{M_x}(ScSu) = \frac{|\{m \in M_x \mid m \textit{ killed by some } S \textit{ in } ScSu\}|}{|\{M_x\}|} \quad (1)$$

In research question **RQ2**, we aim to evaluate the impact of sampling strategies. We generate multiple scenario suites $ScSu_x$ using the sampling algorithms ICPL [30] and Chvatal [29]. Both sampling algorithms are parameterised with respect to their FIC of T features. We parametrise both algorithms with $T = 1$, $T = 2$, and $T = 3$. Thus, we can evaluate the influence of sampling algorithm and FIC separately. Both sampling strategies, ICPL and Chvatal are non-deterministic. We generate five scenario suite versions for each sampling strategy. In the evaluation, we investigate the median and distribution of mutation scores for each sampling strategy. The number of scenarios generated depends on the FIC of T features. For $T = 1$, a scenario suite contains 10 scenarios, while a scenario suite with $T = 2$ contains 118–121 scenarios and $T = 3$ leads to 979–991 scenarios. The resulting mutation score is impacted by the number of scenarios and the selection of scenarios. To assess the impact of scenario selection, we generate, for each scenario suite $ScSu_x$, a randomly sampled scenario suite $ScSu_{x\_rand}$ with the same number of scenarios $|ScSu_{x\_rand}| = |ScSu_x|$. A comparison of $MS_{M_x}(ScSu_x)$ and $MS_{M_x}(ScSu_{x\_rand})$ indicates whether FIC-sampling (systematic scenario selection) leads to stronger scenario suites than random sampling (unsystematic scenario selection). We generate 30 scenario suites that contain a total of 11,145 scenarios. For a comparison of our variability modelling concept and common evaluation methods, we also use the scenario suite $ScSu_{init}$ that contains the seven initial scenarios, from which we derive the FM. These scenarios were selected by expert knowledge.

Research question **RQ3** focuses on the influence of the kill criterion on the mutation score. We use both, EB and SEC as kill criterion to determine the mutation score for each scenario suite. To realise the EB kill criterion, we execute the original AEB and the mutant in parallel. We compare the desired acceleration of both using a relational operator in Simulink. The SEC is independent of the SUT and requires a definition of undesired incidents. For our experiments, we implemented a collision detector.

[5] https://www.mathworks.com/help/driving/ug/autonomous-emergency-braking-with-sensor-fusion.html

[6] https://www.euroncap.com

6.3 Study Results and Discussion

RQ1. We performed simulations with the scenario suite $ScSu_{all}$ for both AEB implementations aeb-1 and aeb-2. The scenario suite $ScSu_{all}$ reaches a mutation score of $MS(ScSu_{all}) = 1$ for both AEB implementations. In other words, after running all 24,412 valid scenarios that can be sampled from the FM, all mutants were identified as such by at least one scenario. For **research question RQ1**, we conclude, that our FM captures a scenario space that is relevant to kill all mutants and that our concept of capturing a scenario space in an FM is feasible. We argue that in general an FM is a suitable abstraction to represent a scenario space for failure detection in scenario-based testing.

RQ2. Figure 4 shows the mutation scores for the scenario suites that we generated with FIC-sampling strategies (red) in comparison with the mutation scores of the related randomly generated scenario suites (blue). We determine the mutation scores using both EB and SEC as kill criterion. We arrange the sampling strategies according to their FIC of T features on the x-axis. The dot represents the median value of the five samples for each sampling strategy and the line depicts minimum and maximum values.

Figure 4 shows that the median mutation scores increase with increasing FIC of T features. The median mutation score for ICPL with increasing coverage, e.g., is $MS(ScSu_{ICPL\_T1}) \approx 0.61$, $MS(ScSu_{ICPL\_T2}) \approx 0.67$, and $MS(ScSu_{ICPL\_T3}) \approx 1$ for aeb-1 with EB kill criterion. The diagrams show the same qualitative trend for any combination of Chvatal, abe-2 and SEC as kill criterion. The scenario suites Chvatal ($T = 3$) in combination with aeb-2 and ICPL ($T = 3$) lead to a median mutation score of one. We also recognise an increasing spread of the mutation scores for the five versions of each scenario suite for increasing T except for ICPL ($T = 2$). If we compare these results with randomly generated scenario suites, we recognise that randomly generated scenario suites lead to lower median and a broader spread of mutation scores. An exception for this is ICPL ($T = 2$) in combination with aeb-2. The mutation scores that we determine using the SEC as kill criterion lead to less or equal mutation scores in median than using the EB kill criterion. We recognise this trend for both AEB implementations. There are outliers for the randomly generated scenario suites related to ICPL ($T = 3$) in combination with aeb-1 although we use the same scenario suites in combination with aeb-2 for which we cannot pinpoint the reason. The scenario suite $ScSu_{ini}$ leads to a mutation score of $MS_{M_{aeb-1}}(ScSu_{ini}) = 0.62$ and $MS_{M_{aeb-2}}(ScSu_{ini}) = 0.52$ with EB kill criterion. The mutation scores determined with SEC as kill criterion are $MS_{M_{aeb-1}}(ScSu_{ini}) = 0.58$ and $MS_{M_{aeb-2}}(ScSu_{ini}) = 0.5$. Thus, the seven initial scenarios from which we derived the FM, kill less than 65% of the mutants.

(1) In our experiment, scenario suites generated by three-wise sampling lead to a detection of at least 75% of mutants. Depending on the AEB implementation, these scenario suites kill all mutants such as the scenario suite generated with all valid sampling. Scenario suites generated by three-wise sampling contain approx. 4% of $ScSu_{all}$ (991 scenarios instead of 24,412). Thus, in our experiment, three-wise sampling significantly reduces test effort in the number of test scenarios while maintaining effectiveness in terms of mutation scores. We conclude FIC-sampling is a reasonable strategy to reduce the number of scenarios in a test suite. However, to reach mutation scores close to 1, a coverage degree of $T > 2$ is necessary.

(2) We conclude that an influence of the sampling algorithm (Chvatal or ICPL) is low but we recognise that the FIC of T features influences the mutation score. The higher T, the stronger is the resulting scenario suite. This effect cannot be explained only by the increasing number of tested scenarios in a suite because a scenario suite with equally many randomly sampled scenarios performs worse in general. One exception is ICPL ($T = 3$) in combination with aeb-2, where the spread of random sampling is smaller than for FIC-sampling. We consider this exception as outlier in favour of randomly generated scenario suites.

(3) We compare the scenario suites generated with FIC-sampling to the scenario suite $ScSu_{ini}$, which consists of seven scenarios that are selected by expert knowledge. The mutation scores that we measure for $ScSu_{ini}$ are comparable to scenario suites sampled with feature sampling strategies. The mutation scores are influenced by both AEB implementation and kill criterion, but the initial scenarios kill less than 65% of the mutants. In contrast, all valid sampling and sampling strategies with $T > 2$ detect up to 100% of the mutants. In terms of mutation scores, FIC-sampling has the potential to be more effective than a scenario selection by expert knowledge.

Thus, for **research question RQ2**, our experiments show that FIC-sampling with $T > 2$ significantly reduces the test effort while maintaining effectiveness in terms of mutation scores. We also show that FIC-sampling leads to more effective scenario suites in terms of mutation scores compared to random sampling. The coverage T has an essential impact on the effectiveness of a scenario suite, while the sampling algorithm is negligible. ICPL is based on the Chvatal sampling algorithm; thus, the resulting scenario suites might be similar causing only a minimal influence. Comparing FIC-sampled scenario suites with a scenario selection by expert knowledge emphasises the potential of our concept with regard to common evaluation methods. Based on these findings, we conclude that systematic scenario selection has potential to outperform unsystematic scenario selection for scenario-based testing.

RQ3. We compare the mutation scores based on EB and SEC kill criterion to investigate the impact of the SEC. The mutation scores in Figure 4 show the same trend of mutation scores with regard to the sampling strategy. The median of the mutation score is equal or lower if we use the SEC as kill criterion instead of EB. This behaviour is plausible because the EB criterion detects whether the behaviour of SUT and mutant are identical. The SEC defines an abstraction of undesired incidences that must not occur. Hence, the SEC is more tolerant which is reflected in the experiment results. There are exceptions, where the SEC kills more mutants as the EB kill criterion. These exceptions occur for three scenario suites in combination with aeb-2: $ScSu_{ini}$, $ScSu_{Chvatal\_T1}$ (version v3), and $ScSu_{ICPL\_T1}$ (version v5). Each of these scenario suites kills less than 55% of the mutants, which is almost as bad as random. These scenario suites are not effective enough to properly kill mutants so that we assume that their results are not suitable to assess kill criteria. However, focussing on scenario suites with $T \geq 2$ the mutation scores of both kill criteria are similar. Moreover, the deviation of EB and SEC kill criterion is small. Regarding **research question RQ3**, we conclude that it is feasible to use an SEC as kill criterion.

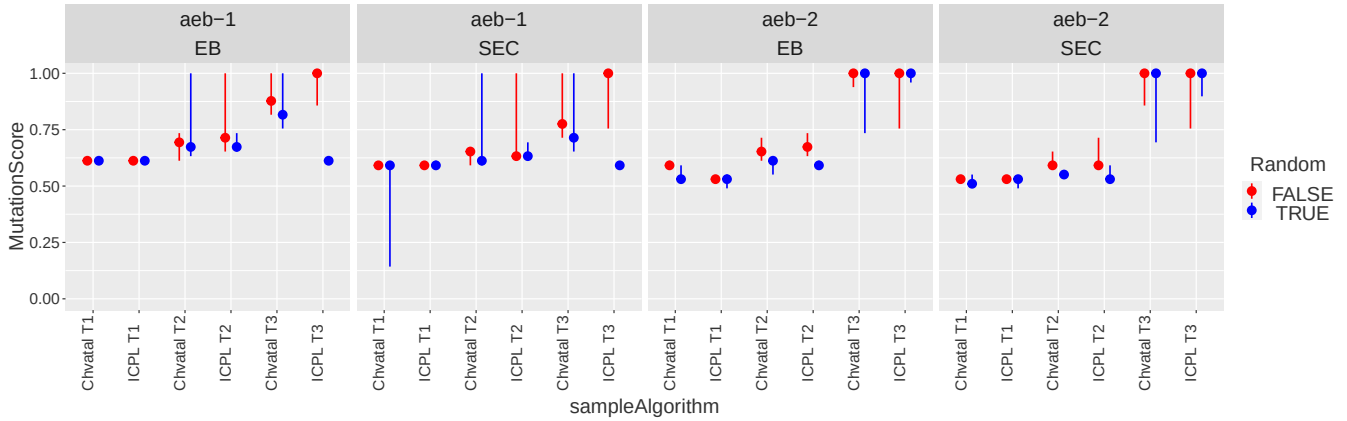

Figure 4: FM implementation for scenario generation to trigger the AEB-system

6.4 Threats to Validity

In the following, we discuss threats to internal, external, and construct validity.

Internal Validity. A threat to internal validity is a bias from the developed FM, which might result from the initial scenarios or the manual creation process. We address this by using EuroNCAP scenarios that are commonly used to evaluate safety of motor vehicles and demo scenarios that are provided by IPG Automotive GmbH for AEB-testing. Subsequently, we reviewed the FM with multiple experts in an iterative process to reduce researcher bias. Another threat to internal validity is non-determinism of FIC-sampling strategies. To mitigate this, we generate five versions of scenario suites and evaluate their median and distribution. A third threat to internal validity is that the implementation of the SUT might influence the mutation score. We address this by using two independent AEB implementations. Although these implementations are academic examples, we argue that SUTs with industrial strength lead to qualitatively similar results. Lower mutation scores are expected, due to more functionality and higher complexity. A last threat to internal validity is an influence of specific mutation operators on the mutation score. To address this, we use the same parameterisation of SIMULTATE for both AEB implementations.

External Validity. A threat to external validity is that the results may be influenced by the usage of specific simulation tools in our workflow. To mitigate this threat, we select the tools CarMaker and Simulink widely adapted in the automotive industry. We expect similar results for alternative solutions. A second threat results from the sampling algorithms used in our experiments, because ICPL is based on Chvatal. Future work is necessary to obtain a reliable, generalizable statement about the influence of the sampling algorithm on the mutation score. Another threat to external validity is that the results may not be generalizable to an FM that covers the overall scenario space of a simulation tool because in our experiments, we reduce the size of the FM by eliminating irrelevant parts to reduce complexity. This threat comes with the generalizability of our results to further ADAS. To address both, we argue that our variability modelling concept is designed to cover a scenario space for ADAS/AD testing. The identification of scenarios of interest is done by expert knowledge in an iterative process. An automated solution for identification of relevant parts and specialised FIC-sampling or slicing for specific ADAS is future work.

Construct Validity. A threat to construct validity is that the mutation score might not be suitable to assess the quality of scenario suites. Mutation testing is commonly accepted for evaluating test suites in software testing [28]. The implementation of ADAS/AD in the automotive context is based on software; thus, we expect using mutation testing for scenario-based testing to be valid.

7 CONCLUSION

In this paper, we leverage variability modelling techniques and mutation testing to solve two essential unsolved challenges of scenario-based testing in the automotive industry: (1) scenario selection and (2) scenario assessment. We select scenarios with coverage-based FM sampling from a scenario space and assess the resulting scenario suites with a mutation score as metric. Our experiments underline that it is feasible to use an FM to represent a scenario space in a template-based structure. FIC-sampling significantly reduces the testing effort while maintaining effectiveness in terms of mutation scores. In our experiments, these systematic sampling strategies outperform unsystematic random sampling and a scenario selection by expert knowledge. The coverage T has an essential impact on the effectiveness of a scenario suite while the selection of the sampling algorithm is negligible. Our experiments also show that it is feasible to use a tolerant kill criterion in form of a safety envelope controller to assess scenarios. These findings demonstrate the potential of variability management and FIC-sampling to improve scenario-based testing. In the future, we plan to optimize and expand template-based FM design focussing on the configurability of E1–E3 elements. We also plan to investigate the influences of SUT and mutation operations on the mutation score. Simulations using a scenario space defined by independent automotive experts will be used to evaluate the generalization of our concept. We also plan to investigate our concept in combination with further ADAS, autonomous driving and sampling strategies.

REFERENCES

[1] Mustafa Al-Hajjaji, Sebastian Krieter, Thomas Thüm, Malte Lochau, and Gunter Saake. 2016. IncLing: efficient product-line testing using incremental pairwise sampling. *ACM SIGPLAN Notices* 52, 3 (2016), 144–155.

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

[3] Harald Altinger, Franz Wotawa, and Markus Schurius. 2014. Testing methods used in the automotive industry: Results from a survey. In *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*. 1–6.

[4] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines*. Springer.

[5] Alberto Arcidiacono. 2018. *ADAS virtual validation: ACC and AEB case study with IPG CarMaker*. Master's thesis. Politecnico di Torino.

[6] G Bagschik, T Menzel, C Körner, and M Maurer. 2018. Wissensbasierte Szenariengenerierung für Betriebsszenarien auf deutschen Autobahnen. In *Workshop Fahrerassistenzsysteme und automatisiertes Fahren. Bd*, Vol. 12.

[7] Gerrit Bagschik, Till Menzel, and Markus Maurer. 2018. Ontology based scene creation for the development of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1813–1820.

[8] Halil Beglerovic, Michael Stolz, and Martin Horn. 2017. Testing of autonomous vehicles using surrogate models and stochastic optimization. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 1–6.

[9] Raja Ben Abdessalem, Shiva Nejati, Lionel C Briand, and Thomas Stifter. 2016. Testing advanced driver assistance systems using multi-objective search and neural networks. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. 63–74.

[10] Lukas Birkemeyer. 2021. *Sampling strategies for generating scenarios for simulation-based validation of advanced driver assistance systems*. Master's thesis. Braunschweig. https://doi.org/10.24355/dbbs.084-202102191146-0

[11] Andreas Classen, Quentin Boucher, and Patrick Heymans. 2011. A text-based approach to feature modelling: Syntax and semantics of TVL. *Science of Computer Programming* 76, 12 (2011), 1130–1143.

[12] ENABLE S3 Consortium. 2019. *Testing & Validation of highly automated systems: Summary of Results*. https://drive.google.com/file/d/15c1Oe69dpvW5dma8-uS8hev17x-6V3zU/view [accessed 2021-05-07].

[13] PEGASUS Consortium. 2020. *Schlussbericht für das Gesamtprojekt PEGASUS Projekt zur Etablierung von generell akzeptierten Gütekriterien, Werkzeugen und Methoden sowie Szenarien und Situationen zur Freigabe hochautomatisierter Fahrfunktionen*. https://www.pegasusprojekt.de/files/tmpl/pdf/PEGASUS_Abschlussbericht_Gesamtprojekt.PDF [accessed 2021-05-07].

[14] Maxime Cordy, Pierre-Yves Schobbens, Patrick Heymans, and Axel Legay. 2013. Beyond boolean product-line model checking: dealing with feature attributes and multi-features. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 472–481.

[15] Krzysztof Czarnecki and Simon Helsen. 2006. Feature-based survey of model transformation approaches. *IBM Systems Journal* 45, 3 (2006), 621–645.

[16] Erwin de Gelder and Jan-Pieter Paardekooper. 2017. Assessment of automated driving systems using real-life scenarios. In *Intelligent Vehicles Symposium (iv)*. IEEE, 589–594.

[17] Richard A DeMillo, Richard J Lipton, and Frederick G Sayward. 1978. Hints on test data selection: Help for the practicing programmer. *Computer* 11, 4 (1978), 34–41.

[18] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An open urban driving simulator. *arXiv preprint arXiv:1711.03938* (2017).

[19] Emelie Engström and Per Runeson. 2011. Software product line testing–a systematic mapping study. *Information and Software Technology* 53, 1 (2011), 2–13.

[20] Daniel J Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L Sangiovanni-Vincentelli, and Sanjit A Seshia. 2019. Scenic: a language for scenario specification and scene generation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 63–78.

[21] Alessio Gambi, Tri Huynh, and Gordon Fraser. 2019. Generating effective test cases for self-driving cars from police reports. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 257–267.

[22] Alessio Gambi, Marc Mueller, and Gordon Fraser. 2019. Automatically testing self-driving cars with search-based procedural content generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 318–328.

[23] IPG Automotive GmbH. [n. d.]. CarMaker. https://ipg-automotive.com/de/produkte-services/simulation-software/carmaker/. [accessed 2020-12-08].

[24] IPG Automotive Group [n. d.]. *User's Guide*. IPG Automotive Group.

[25] ISO. 2018. Road vehicles — Functional safety.

[26] ISO. 2019. Road vehicles — Safety of the intended functionality.

[27] Ian Rhys Jenkins, Ludvig Oliver Gee, Alessia Knauss, Hang Yin, and Jan Schroeder. 2018. Accident scenario generation with recurrent neural networks. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 3340–3345.

[28] Yue Jia and Mark Harman. 2010. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering* 37, 5 (2010), 649–678.

[29] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. 2011. Properties of realistic feature models make combinatorial testing of product lines feasible. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 638–652.

[30] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. 2012. An algorithm for generating t-wise covering arrays from large feature models. In *Proceedings of the 16th International Software Product Line Conference-Volume 1*. 46–55.

[31] Christian Kastner, Thomas Thum, Gunter Saake, Janet Feigenspan, Thomas Leich, Fabian Wielgorz, and Sven Apel. 2009. FeatureIDE: A tool framework for feature-oriented software development. In *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 611–614.

[32] BaekGyu Kim, Akshay Jarandikar, Jonathan Shum, Shinichi Shiraishi, and Masahiro Yamaura. 2016. The SMT-based automatic road network generation in vehicle simulation environment. In *2016 International Conference on Embedded Software (EMSOFT)*. IEEE, 1–10.

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

[34] Philip Koopman, Uma Ferrell, Frank Fratrik, and Michael Wagner. 2019. A safety standard approach for fully autonomous vehicles. In *International Conference on Computer Safety, Reliability, and Security*. Springer, 326–332.

[35] Sebastian Krieter, Thomas Thüm, Sandro Schulze, Gunter Saake, and Thomas Leich. 2020. YASA: yet another sampling algorithm. In *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems*. 1–10.

[36] D Richard Kuhn, Dolores R Wallace, and Albert M Gallo. 2004. Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering* 30, 6 (2004), 418–421.

[37] Jihyun Lee, Sungwon Kang, and Danhyung Lee. 2012. A survey on software product line testing. In *Proceedings of the 16th International Software Product Line Conference-Volume 1*. 31–40.

[38] Wen-Shing Lee, Doris L Grosh, Frank A Tillman, and Chang H Lie. 1985. Fault tree analysis, methods, and applications a review. *IEEE Transactions on Reliability* 34, 3 (1985), 194–203.

[39] Till Menzel, Gerrit Bagschik, and Markus Maurer. 2018. Scenarios for development, test and validation of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 1821–1827.

[40] Phillipp Mevenkamp. 2019. *Mutation Testing for Autonomous Vehicle*. Master's thesis. Technical University of Braunschweig.

[41] Pascal Minnerup, Tobias Kessler, and Alois Knoll. 2015. Collecting simulation scenarios by analyzing physical test drives. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. IEEE, 2915–2920.

[42] Daniel-Jesus Munoz, Jeho Oh, Mónica Pinto, Lidia Fuentes, and Don Batory. 2019. Uniform random sampling product configurations of feature models that have numerical features. In *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A*. 289–301.

[43] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding near-optimal configurations in product lines by random sampling. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 61–71.

[44] Ingo Pill, Ivan Rubil, Franz Wotawa, and Mihai Nica. 2016. SIMULTATE: A toolset for fault injection and mutation testing of Simulink models. In *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 168–173.

[45] Andreas Pütz, Adrian Zlocki, Julian Bock, and Lutz Eckstein. 2017. System validation of highly automated vehicles with a database of relevant traffic scenarios. *situations* 1 (2017), E5.

[46] R Core Team. 2020. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org

[47] Elias Rocklage, Heiko Kraft, Abdullah Karatas, and Jörg Seewig. 2017. Automated scenario generation for regression testing of autonomous vehicles. In *2017 ieee 20th international conference on intelligent transportation systems (itsc)*. IEEE, 476–483.

[48] Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. 2006. Feature diagrams: A survey and a formal semantics. In *14th IEEE International Requirements Engineering Conference (RE'06)*. IEEE, 139–148.

[49] Fabian Schuldt. 2017. *Ein Beitrag für den methodischen Test von automatisierten Fahrfunktionen mit Hilfe von virtuellen Umgebungen*. Ph. D. Dissertation. TU Braunschweig.

[50] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. 2014. FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming* 79 (2014), 70–85.

[51] Cumhur Erkan Tuncali and Georgios Fainekos. 2019. Rapidly-exploring random trees for testing automated vehicles. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 661–666.

[52] Cumhur Erkan Tuncali, Theodore P Pavlic, and Georgios Fainekos. 2016. Utilizing S-TaLiRo as an automatic test generation framework for autonomous vehicles. In *International Conference on Intelligent Transportation Systems (itsc)*. IEEE, 1470–1475.

[53] Simon Ulbrich, Till Menzel, Andreas Reschka, Fabian Schuldt, and Markus Maurer. 2015. Defining and substantiating the terms scene, situation, and scenario for automated driving. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*. IEEE, 982–988.

[54] Simon Ulbrich, Tobias Nothdurft, Markus Maurer, and Peter Hecker. 2014. Graph-based context representation, environment modeling and information aggregation for automated driving. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE, 541–547.

[55] MSC.Software GmbH VIRES Simulationstechnologie GmbH. [n. d.]. Virtual Test Drive. https://www.mscsoftware.com/de/virtual-test-drive. [accessed 2020-12-08].

[56] Mladen A Vouk. 1988. On back-to-back testing. In *Computer Assurance, 1988. COMPASS'88*. IEEE, 84–91.

[57] Walther Wachenfeld and Hermann Winner. 2016. The release of autonomous vehicles. In *Autonomous driving*. Springer, 425–449.

[58] Marc René Zofka, Florian Kuhnt, Ralf Kohlhaas, Christoph Rist, Thomas Schamm, and J Marius Zöllner. 2015. Data-driven simulation and parametrization of traffic scenarios for the development of advanced driver assistance systems. In *2015 18th International Conference on Information Fusion (Fusion)*. IEEE, 1422–1428.