SOTIF-Compliant Scenario Generation Using Semi-Concrete Scenarios and Parameter Sampling

Lukas Birkemeyer[1], Julian Fuchs[2], Alessio Gambi[3] and Ina Schaefer[4]

***Abstract*— The SOTIF standard (ISO 21448) requires scenario-based testing to verify and validate Advanced Driver Assistance Systems and Automated Driving Systems but does not suggest any practical way to do so effectively and efficiently. Existing scenario generation approaches either focus on exploring or exploiting the scenario space. This generally leads to test suites that cover many known cases but potentially miss edge cases or focused test suites that are effective but also contain less diverse scenarios. To generate SOTIF-compliant test suites that achieve higher coverage and find more faults, this paper proposes semi-concrete scenarios and combines them with parameter sampling to adequately balance scenario space exploration and exploitation. Semi-concrete scenarios enable combinatorial scenario generation techniques that systematically explore the scenario space, while parameter sampling allows for the exploitation of continuous parameters. Our experimental results show that the proposed concept can generate more effective test suites than state-of-the-art coverage-based sampling. Moreover, our results show that including a feedback mechanism to drive parameter sampling further increases test suites' effectiveness.**

I. Introduction

The Safety of the Intended Functionality (SOTIF)-standard (ISO 21448) [1] requires scenario-based testing to validate Advanced Driver Assistance Systems (ADASs) and Automated Driving Systems (ADSs). In scenario-based testing, scenarios precisely describe relevant environmental elements to include in the testing process and the System Under Test (SUT)'s initial state. *Logical scenarios* describe the main semantics of the scenarios and include parameters that influence their instantiation [2]. For example, a hypothetical logical scenario for testing an Adaptive Cruise Control (ACC) might occur on a highway; the ego vehicle travels at a given *speed* (v_{ego}) and approaches *another vehicle*.Watanabe [3] distinguishes between *continuous* parameters (e.g., vehicle speed) and *discrete* parameters (e.g., vehicle model). Continuous parameters are usually defined in terms of ranges (e.g., $v_{ego} = [120, 150]\ km/h$). Scenario-based testing requires instantiating logical scenarios in *concrete scenarios*, i.e., test cases, by assigning specific values to continuous and discrete parameters [4]. For instance, in a concrete scenario, the ego-vehicle travels at 123 km/h on a highway and approaches a *VW Beetle*. The set of all the generated concrete scenarios forms a test suite (i.e., scenario suite).

[1]Lukas Birkemeyer is with Technical University Braunschweig, Braunschweig, Germany l.birkemeyer@tu-braunschweig.de

[2]Julian Fuchs is with FZI Forschungszentrum Informatik, Karlsruhe, Germany fuchs@fzi.de

[3]Alessio Gambi is with IMC University of Applied Sciences, Krems, Austria alessio.gambi@fh-krems.ac.at

[4]Ina Schaefer is with Karlsruhe Institute of Technology, Karlsruhe, Germany ina.schaefer@kit.edu

The SOTIF standard requires scenario-based testing to ensure that an ADAS/ADS operates as intended within a specified Operational Design Domain (ODD) [1], i.e., a set of well-defined execution conditions and their possible interplay. However, SOTIF does not suggest any concrete strategy to select discrete parameters and sample continuous ones; thus, the challenge of instantiating logical scenarios into concrete ones remains open. According to Birkemeyer et al. [5], existing research on parameter selection and scenario generation includes: *data-driven* techniques that select parameters based on real-world data instances [6], [7] or distributions [8], [9]; *optimization* techniques that exploit parameter spaces using search-based methods [10]–[16] or machine learning [17]; and, *combinatorial* techniques that explore scenario spaces by systematically combining atomic scenario elements [18], [19]. SOTIF requires that scenario suites represent the ODD [1]. Thus, we argue that although data-driven scenario generation can generate realistic scenarios and optimization-based scenario generation can generate critical scenarios, only combinatorial scenario generation, which systematically covers all interactions of t atomic scenario elements, has the potential to generate SOTIF-compliant test suites. However, SOTIF also requires that the set of scenarios that lead to unsafe behavior of the SUT (i.e., critical scenarios) becomes minimal [1]. Since existing combinatorial scenario generation techniques cannot provide this property [19], they must be extended to balance scenario space exploration vs. exploitation.

To solve this issue, in this paper, we propose to generate effective, SOTIF-compliant test suites by combining combinatorial scenario generation techniques and parameter sampling. The key enabler of the proposed approach is semi-concrete scenarios that conceptually sit between logical and concrete scenarios. Semi-concrete scenarios enable covering discrete parameters (exploration) while optimizing continuous parameters (exploitation). We empirically evaluate the proposed approach by assessing the effectiveness of the test suites it generates, and we observed these test suites find more faults in the SUT than those generated by existing combinatorial scenario generation techniques.

This paper makes the following contributions:

1) A scenario generation technique based on the novel concept of semi-concrete scenarios that combines combinatorial testing and parameter sampling to generate effective test suites.
2) An empirical evaluation of the effectiveness of test suites generated from semi-concrete scenarios using combinatorial generation and parameter sampling.

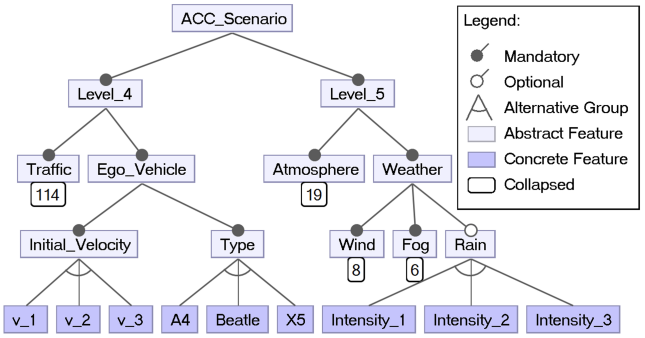


Fig. 1: Excerpt of the feature model proposed in [18]. The *initial velocity* and *type* of the ego-vehicle are mandatory features, while the feature *rain* is optional.

II. Variability Modeling Techniques

This section provides fundamental knowledge regarding feature modeling and sampling strategies to make the paper self-contained.

a) Feature Modeling: Variability modeling techniques are common methods to model highly configurable systems in software engineering [20], [21]. Configuration options are modeled as *features* in a *feature model*. A feature is a binary, user-visible system configuration option (e.g., the color or type of a vehicle). A feature model has a tree structure and describes characteristics of features (optional/mandatory) and dependencies between features (alternative/AND/OR as well as parent/child - relation). In Figure 1, we present an excerpt of a feature model that represents scenarios for scenario-based testing inspired by Birkemeyer et al. [18]. The feature model is structured according to the six-scenario levels proposed by Scholtes et al. [22] and covers atomic scenario elements. An atomic scenario element is a concrete entity of a scenario such as rain with the intensity of $1mm/h$ or the initial velocity of a vehicle $v_{ego} = 25km/h$. Thus, the scenario feature model introduced by Birkemeyer et al. [18] covers the space of possible scenarios.

b) Sampling Strategies: Due to the combinatorial explosion, the number of valid configurations represented by a feature model becomes extremely large [23], [24]; hence, testing all scenarios is practically infeasible and sampling strategies to select a subset of all valid configurations [25]–[27] must be used to generate concrete scenarios. Selecting a set of configurations that is representative of the overall configuration space makes it possible to derive sound assumptions about the adequacy of the generated test suites. Coverage-based sampling strategies select configurations so that each feature or interaction of t features is covered at least once in the representative subset. Considering the scenario feature model, coverage-based sampling explores the scenario space by ensuring that the resulting scenario suite contains each combination of t atomic scenario elements. State-of-the-art algorithms for coverage-based sampling are Chvatal [25], ICPL [26], and YASA [27].

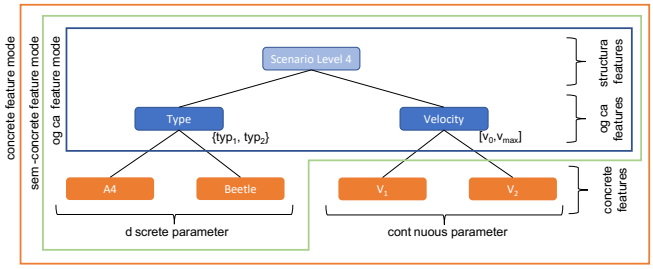


Fig. 2: Hybrid scenario feature model to represent logical (blue), semi-concrete (green), and concrete (orange) scenarios. The logical scenario feature model is part of the semi-concrete and part of the concrete scenario feature model.

III. Balancing Exploration and Exploitation

This paper aims to establish a novel concept for scenario generation that allows balancing exploration and exploitation of scenario spaces to improve the test effectiveness of SOTIF-compliant scenario-based testing. Klück et al. [19] state that combinatorial scenario generation (exploration) is more effective in detecting fault than optimization-based scenario generation (exploitation). Thus, we use combinatorial scenario generation as a starting point. However, we observed that standard combinatorial scenario generation, which uses equivalence classes, i.e., predefined value assignments, to discretize continuous parameters, is sub-optimal. The following example demonstrates that equivalence classes do not properly represent a continuous parameter space because one parameter might occur in different scenarios, possibly requiring it to take different values. In this example, we consider only the initial velocity of the ego vehicle v. On a straight road (*scenario-1*), v_1 represents an equivalence class. If we, however, replace the straight road with a sharp turn (*scenario-2*), v_2 represents an equivalence class that is different from the equivalence class defined for *scenario-1* ($v_1 \neq v_2$). To sum up, existing combinatorial scenario generation leads to effective scenario suites but has issues in discretizing continuous parameters. Optimization techniques, in contrast, are explicitly designed to select concrete values from continuous parameters that optimize a fitness function [28]. Alternative parameter sampling strategies select parameters randomly [19], equidistantly [29], or based on probability density functions [30]. In our scenario generation concept, we combine the best of both worlds: We explore scenario spaces by using a combinatorial approach for sampling discrete components and exploiting the parameter space by sampling continuous parameters.

A. Semi-Concrete Scenarios

To combine exploration and exploitation in the context of variability modeling, we introduce *semi-concrete* scenarios. A semi-concrete scenario combines logical and concrete scenarios by assigning concrete values for discrete parameters but leaves the value ranges of continuous ones. Consequently, generating semi-concrete scenarios that systematically cover all interactions of t discrete scenario elements allows us to explore the scenario space systematically, but it leaves open

the possibility to exploit each of the generated semi-concrete scenarios by sampling the continuous parameters. To this end, we extend the existing scenario feature model proposed by Birkemeyer et al. [18] to include semi-concrete scenarios. The novel scenario feature model structure can represent scenarios at different abstraction levels (i.e., logical, semi-concrete, and concrete). We refer to the resulting extended feature models as *hybrid scenario feature models*. Figure 2 exemplifies the hybrid scenario feature model.

The *hybrid scenario feature model* consists of (a) *structure features*, (b) *logical features*, and (c) *concrete features* (cf. Figure 2). The *structure features* are abstract and structure the scenario feature model in a tree structure, e.g., to allocate atomic scenario elements to the common six-scenario levels proposed in [22]. *Logical features* represent parameters and possible parameter ranges. In our example, (Vehicle-)*Type* and *Velocity* are logical features. The feature *Type* includes a list of discrete parameters (i.e., $type = [A4, Beetle, X5, \ldots]$), whereas velocity is a continuous parameter (i.e., $v = [0, 210]\ km/h$). Selecting scenarios from a feature model that contains structure and logical features results in logical scenarios; thus, we define it as a *logical feature model*. In Figure 2, we mark the *logical feature model* with a blue bounding box.

Using a *logical feature model*, we can add concrete features representing concrete parameter assignments and obtain concrete scenarios. For discrete parameters, such as the vehicle type, we add a concrete feature for each type; however, we need to discretize the range for continuous parameters. We do so by adding a feature for each equivalence class. Standard combinatorial approaches use expert-defined equivalence classes [18], [19]. Adding concrete features to a logical feature model results in a *concrete feature model*. The *logical feature model* models the same logical scenarios as the remaining *concrete feature model* (cf. Figure 2, orange bounding box). However, the *logical feature model* implicitly models a broader space of concrete scenarios wrt. to continuous scenario parameters.

Finally, to select semi-concrete scenarios, we use the *semi-concrete feature model* (cf. Figure 2, green bounding box). In this feature model, discrete parameter values are modeled as concrete features, but continuous parameters are represented with logical features. Sampling the continuous parameters can be done in various ways. For instance, one can sample concrete values for the continuous parameters within the entire range of possible values (see Sect. III-B) or within sub-parameter ranges (see Sect. III-C).

B. Parameter Range Sampling

During the modeling process of the semi-concrete feature models, we use logical features to represent representative upper and lower boundaries for continuous parameters. Those boundaries might be defined by expert knowledge or derived from real-world data. For example, the manufacturer specifies a vehicle's maximal velocity, whereas minimal and maximal rain intensity can be derived from real-world observations. These boundaries allow continuous parameters to be sampled over the entire range of values. In our evaluation, we opted to sample parameters randomly to avoid introducing any bias; however, continuous parameters could be sampled using optimization techniques or probability density functions.

C. Sub-Parameter Range Sampling

Similar to equivalence classes, we define sub-parameter ranges that partitions the range of continuous parameter values into smaller ranges that are combined to cover each interaction of t parameter ranges. In sub-parameter range sampling, concrete values are sampled from those ranges every time a new concrete scenario is generated. Thus, in contrast to equivalence classes, sub-parameter range sampling does not always select the predefined representative values defined by the domain expert. Nonetheless, it systematically covers (explores) parameter ranges. As for parameter range sampling, in our evaluation, we opted to sample parameters within each sub-parameter range randomly. Continuous parameters could be sampled using optimization techniques or probability density functions with or against a distribution.

Adopting sub-parameter range sampling requires (1) generating semi-concrete scenarios and (2) sampling concrete values for each of them. Parameter range sampling, instead, does not require covering each interaction of t parameter ranges; consequently, it is less expensive, as the parameters are sampled fewer times. However, because the value ranges considered by parameter range sampling are larger than the ones considered by sub-parameter range sampling, the generated test cases might be less effective. In summary, parameter range sampling only exploits the continuous parameters defined from the explored logical scenarios, whereas sub-parameter range sampling adds an intermediate level of exploration, i.e., exploring the interaction of t sub-parameter ranges, before the final parameters exploitation. By selecting one or the other option, developers can trade off the cost and effectiveness of the generated test suites.

IV. Evaluation

To assess the benefits of combining combinatorial testing and parameter sampling and to understand how parameter sampling affects the effectiveness of the generated test scenarios, we investigate the following main research questions:

RQ1 *Does combining combinatorial scenario generation and parameter sampling generate more effective test suites than standard combinatorial scenario generation?* Parameter sampling enables exploiting continuous parameters. Hence, in the sense of fault detection ability, we want to understand if this leads to generating more effective test suites than using expert-defined values.

RQ2 *How does a specific sampling technique impact test suite effectiveness?* Sampling parameters using sub-parameter range sampling is more expensive than parameter range sampling in the overall possible range. This raises the question of whether sub-parameter range sampling generates test suites that identify more faults than the one generated by parameter range sampling.

RQ3 *How does "sampling with feedback" impact the effectiveness of a test suite?* Optimization techniques use a feedback loop to guide the parameter sampling process. We are interested if parameter sampling *with feedback* outperforms purely random parameter sampling when combined with combinatorial scenario generation.

A. Experimental Setup

To evaluate the effectiveness of combining combinatorial scenario generation and parameter sampling, we implemented the proposed approach in a prototype[1] and used it to generate concrete scenarios. We executed the concrete scenarios in the virtual environment provided by IPG Carmaker version 11.0.1.[2] As a test subject, we implemented an adaption of the Adaptive Cruise Control (ACC) provided by IPG and Mathworks[3] in Simulink. The ACC adapts the ego vehicle's speed to the leading vehicle's speed. Following an iterative process that involved two researchers, we implemented a hybrid feature model representing relevant scenarios to test ACC functionality. The resulting hybrid feature model contains 193 features in total, with 31 parameter features and 93 expert-defined sub-parameter ranges, i.e., three sub-parameter ranges per continuous parameter. We share the feature model online.[1] Using the hybrid feature model and our prototype, we generated test suites by sampling semi-concrete scenarios with parameter ranges and sub-parameter ranges. In both cases, we used the state-of-the-art coverage-based sampling algorithm YASA [27] to cover the interaction of t parameters; we used YASA *t=1* (feature-wise) and YASA *t=2* (pair-wise). Specifically, *parameter ranges* produced 5 (YASA *t=1*) and 30 (YASA *t=2*) semi-concrete scenarios, whereas *sub-parameter ranges* produced 7 (YASA *t=1*) and 80 (YASA *t=2*) semi-concrete scenarios. To transfer semi-concrete scenarios into concrete scenarios, we used randomized parameter sampling, which we repeated 10 times to increase confidence in our conclusions. As a *baseline* for comparison, we considered the sub-parameter ranges as equivalence classes and selected scenarios with state-of-the-art combinatorial scenario generation. We generated test suites by using YASA (*t=1* and *t=2*) and assigned the expert-defined values to all the equivalence classes (i.e., sub-parameter ranges) instead of sampling them. Notably, since the baseline uses the same sub-parameter ranges used by sub-parameter range sampling, it produced the same amount of concrete scenarios, i.e., 7 (YASA *t=1*) and 80 (YASA *t=2*) concrete scenarios. Specifically, for each comparison, we measure *statistical significance* (p-value) using the Mann–Whitney U-test. Broadly speaking, p-values less than 0.05 indicate statistically significant results. In addition to assessing statistical significance, we measure the *effect size* using the Vargha and Delaney A_{12}, which captures the magnitude of the differences between two distributions (i.e., the effectiveness of two scenario suites). For example, a value of A_{12} equal to $= 0.5$ means that both suites are equally effective; thus, the difference in their effectiveness is *negligible*. However, as the A_{12} value moves away from the 0.5 threshold, the difference in test suite effectiveness becomes *small*, *medium*, or *large*.

Feedback Loop: Regarding RQ 3, we guide parameter sampling with a feedback loop trying to increase the test suites' effectiveness by limiting the occurrence of potentially irrelevant scenarios. The feedback loop is a first step and considered as a proof of concept in combining coverage-based sampling (exploration) and optimization techniques (exploitation). We introduce an independent Safety Relevance Controller (SRC) to determine whether a scenario is relevant. Regarding the ACC as SUT, we define a concrete scenario as relevant if at least one object occurs within the sensor's field of view within the first 10 seconds. We re-sample parameters up to 50 times if a scenario is irrelevant.

Test Suite Effectiveness: Inspired by Birkemeyer et al. [18], we assessed the test effectiveness of all generated test suites using fault-based testing [31]. Thus, to assess the ability of tests to expose problems in the SUT, we purposely injected faults in it and checked whether the tests identified them. We used the SIMULTATE Framework [32] to generate 50 random faulty versions of the ACC and counted the ones that are detected by the test suite. Intuitively, the more faults are found by a test suite, the more effective that test suite is. Therefore, as an effectiveness metric, we use the *mutation score*, i.e., the ratio of detected faulty versions to all.

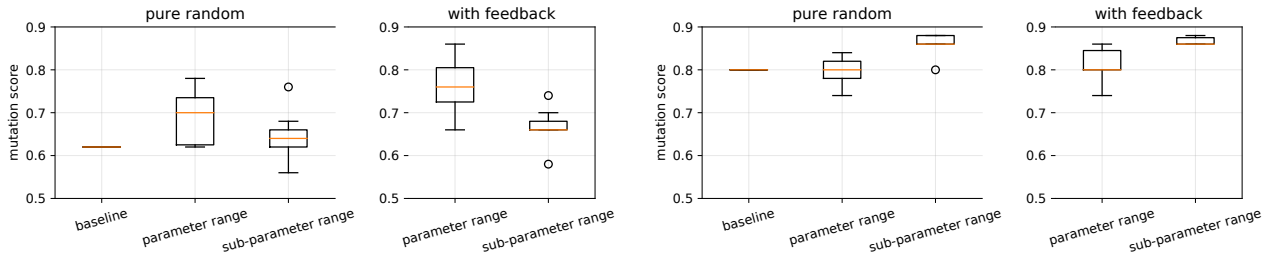
B. Results and Discussion

RQ1. Benefits of Combining Combinatorial Scenario Generation and Parameter Sampling: Figure 3 reports the number of faults found, i.e., *mutation score*, by each test suite we generated using YASA *t=1* (Figure 3-a) and YASA *t=2* (Figure 3-b). From the figure, we can observe that for YASA *t=1*, both sampling strategies generate more effective test suites than the baseline. Comparing the baseline and Parameter Range Sampling resulted in a statistically significant result (p-value$= 0.042$) and a *large* effect size while comparing the baseline and Sub-Parameter Range Sampling resulted in a less statistically significant result (p-value< 0.069) and *medium* effect size. For YASA *t=2*, we can observe that the baseline and Parameter Range Sampling achieved comparable results (p-value=1.0 and negligible effect size); however, Sub-Parameter Range Sampling generates statistically more effective test suites (p-value< 0.005 and *large* effect size). Finally, comparing the results achieved by covering single features (YASA *t=1*) and pair-wise interactions between them (YASA *t=2*), we can observe that -as expected- generating more scenarios lead to more effective test suites.

In light of those observations, we conclude that combining combinatorial scenario generation and parameter sampling is beneficial for scenario-based testing as it systematically covers all interactions of t features while outperforming basic combinatorial strategies discretizing continuous parameters.

RQ2. Impact of Sampling on Test Effectiveness: To address RQ2, we refer again to Figure 3 and compare

[1] https://doi.org/10.5281/zenodo.7940902
[2] https://ipg-automotive.com/
[3] https://mathworks.com/



(a) feature-wise interaction coverage YASA (t=1)

(b) pair-wise feature interaction coverage YASA (t=2)

Fig. 3: Test effectiveness of scenario suites generated using combinatorial scenario generation and parameter sampling for YASA *t=1* (a) and YASA *t=2* (b). We separate parameter sampling strategies that select parameters purely randomly or with an iterative feedback loop. For readability, we removed outliers that lead to mutation scores smaller than 0.5

the results obtained using Parameter Range Sampling and Sub-Parameter Range Sampling. We observe that none of the two methods always produces the most effective test suite. Specifically, for YASA *t=1*, Parameter Range Sampling produced a slightly better test suite (p-value$=0.3$ and *small* effect size). On the contrary, for YASA *t=2*, Sub-Parameter Range Sampling produced more effective test suites (p-value=<0.005 and *large* effect size). Although Parameter Range Sampling generated smaller test suites, it achieved good results (i.e., never worse than the baseline).

In light of those observations, we conclude that Parameter Range Sampling is a viable choice when scenario generation focuses on covering all scenario features or when scenario-based testing is budget-constrained, while Sub-Parameter Range Sampling might be a better choice when developers aim to fulfill stronger coverage criteria and can afford to generate larger test suites.

RQ3: Impact of Sampling with Feedback on Test Effectiveness: To answer RQ3, we compare the test effectiveness for scenario suites generated *with* and *without* feedback loop during the parameter sampling process. The results are shown in Figure 3 ("pure random" vs. "with feedback"). We assume that the test effectiveness increases for parameter sampling *with* feedback. Regarding YASA *t=1*, there is a clear increase of effectiveness for both sampling strategies: Parameter Range Sampling (p-value$=0.03$ and *large* effect size) and Sub-Parameter Range Sampling (p-value$=0.4$ and *small* effect size). Regarding YASA *t=2*, however, the increase is marginal; (p-value=0.5 and *small* effect size) for Parameter Range Sampling and (p-value=0.9 and *negligible* effect size) for Sub-Parameter Range Sampling. It is worth mentioning that both parameter sampling strategies *with* feedback perform similarly to each other for YASA *t=1* and *t=2* as both strategies *without* feedback: Parameter Range Sampling leads to more effective scenario suites for YASA *t=1*, while Sub-Parameter Range sampling is more effective for YASA *t=2* (cf. RQ 2).

In light of those observations, we conclude that adding feedback to parameter sampling slightly increases the test effectiveness of generated scenario suites. However, during our experiments, parameter sampling with feedback requires in median of 1.2% (overall parameter range) and 14% (sub-parameter range) additional simulation effort. For the sake of fairness in this comparison, we use simple resource allocation (50 tries for each (sub-)parameter range) [33]. But, relevant parameters might be rare in some (sub-)parameter ranges, which impacts the additional effort. However, depending on the use-case, test engineers need to carefully weigh costs and benefits while generating SOTIF-compliant scenario suites.

C. Threats to Validity

a) Internal validity: A threat to the internal validity of our study is relying on expert knowledge to define the feature model, equivalence classes, etc. To mitigate this threat, we rely on the knowledge of two independent experts. However, our results show that using the values defined by equivalence classes does not lead to equivalent behavior in all scenarios. We argue that the industry will face similar challenges; thus, our results are meaningful. Another threat comes with the definition of *relevant scenarios* for the feedback loop. We argue that the definition is arbitrary and a systematic study in the future could address this threat. Finally, a threat results from the (external) tools we used for evaluation. To mitigate this threat, we rely on established tools (Carmaker and Simulink) in the automotive industry and adapted an existing implementation of ACC.

b) External validity: Considering only ACC functionality in the evaluation cannot let us generalize the conclusions to other ADAS or ADS. However, we argue that an ACC is established in modern vehicles; hence, it must be tested.

V. CONCLUSION

This paper establishes a novel concept for semi-concrete scenario generation and parameter sampling to generate effective test suites for scenario-based testing of ADAS/ADS. The proposed concept allows developers to balance the trade-off between exploring and exploiting driving scenario spaces as intended by SOTIF (ISO 21448). Based on our results, we conclude that semi-concrete scenario generation and parameter sampling increases the effectiveness of SOTIF-compliant scenario generation to verify and validate ADAS/ADS. Moreover, depending on the parameter sampling strategy (full-/sub-parameter ranges, with/without feedback), our results indicate the potential for different testing strategies.

In the future, we aim to improve the test effectiveness of the generated scenario suites and address the major external

and internal threats to validity. We will add real-world data to define continuous parameter ranges and focus on strategies to define sub-parameter ranges automatically. Moreover, we will apply optimization techniques such as search-based or stochastic optimization to improve parameter sampling with feedback to balance resource allocations.

ACKNOWLEDGMENTS

We thank David Schultz for constructive discussions and for supporting the implementations. This work has been partially funded by the Ph.D. program "Responsible AI in the Digital Society" funded by the Ministry for Science and Culture of Lower Saxony, Germany, and the EU Project Flexcrash (Grant Agreement n. 101069674). This work has been partially supported by projects within the Innovations Campus Future Mobility (ICM) funded by the Ministry of Science, Research and Arts of the Federal State of Baden-Württemberg.

REFERENCES

[1] *Road vehicles — Safety of the intended functionality*, Std. ISO 21 448:2022, 2022.

[2] M. Steimle, T. Menzel, and M. Maurer, "Toward a consistent taxonomy for scenario-based development and test approaches for automated vehicles: A proposal for a structuring framework, a basic vocabulary, and its application," *IEEE Access*, vol. 9, pp. 147 828–147 854, 2021.

[3] H. Watanabe, *Methodik zur Determinierung repräsentativer und relevanter Testszenarien für prädiktive Sicherheitsfunktionen*. Cuvillier Verlag, 2022, vol. 21.

[4] G. Bagschik, T. Menzel, C. Körner, and M. Maurer, "Wissensbasierte szenariengenerierung für betriebsszenarien auf deutschen autobahnen," in *Workshop Fahrerassistenzsysteme und automatisiertes Fahren. Bd*, vol. 12, 2018, p. 12.

[5] L. Birkemeyer, C. King, and I. Schaefer, "Is scenario generation ready for SOTIF? a systematic literature review," in *2023 26th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2023.

[6] A. Gambi, T. Huynh, and G. Fraser, "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*, 2019, pp. 257–267.

[7] F. Montanari, C. Stadler, J. Sichermann, R. German, and A. Djanatliev, "Maneuver-based resimulation of driving scenarios based on real driving data," in *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2021, pp. 1124–1131.

[8] E. de Gelder and J.-P. Paardekooper, "Assessment of automated driving systems using real-life scenarios," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 589–594.

[9] R. Krajewski, T. Moers, A. Meister, and L. Eckstein, "Béziervae: Improved trajectory modeling using variational autoencoders for the safety validation of highly automated vehicles," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 3788–3795.

[10] A. Gambi, M. Mueller, and G. Fraser, "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*, 2019, pp. 318–328.

[11] A. Gambi, M. Müller, and G. Fraser, "Asfault: Testing self-driving car software using search-based procedural content generation," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 2019, pp. 27–30.

[12] R. B. Abdessalem, A. Panichella, S. Nejati, L. C. Briand, and T. Stifter, "Testing autonomous cars for feature interaction failures using many-objective search," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 143–154.

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

[14] Z. Ghodsi, S. K. S. Hari, I. Frosio, T. Tsai, A. Troccoli, S. W. Keckler, S. Garg, and A. Anandkumar, "Generating and characterizing scenarios for safety testing of autonomous vehicles," in *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2021, pp. 157–164.

[15] C. E. Tuncali and G. Fainekos, "Rapidly-exploring random trees for testing automated vehicles," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 661–666.

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

[17] D. Baumann, R. Pfeffer, and E. Sax, "Automatic generation of critical test cases for the development of highly automated driving functions," in *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*. IEEE, 2021, pp. 1–5.

[18] L. Birkemeyer, T. Pett, A. Vogelsang, C. Seidl, and I. Schaefer, "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*, 2022, pp. 1–10.

[19] F. Klück, Y. Li, J. Tao, and F. Wotawa, "An empirical comparison of combinatorial testing and search-based testing in the context of automated and autonomous driving systems," *Information and Software Technology*, p. 107225, 2023.

[20] S. Apel, D. Batory, C. Kästner, and G. Saake, *Feature-oriented software product lines*. Springer, 2016.

[21] C. Kastner, T. Thum, G. Saake, J. Feigenspan, T. Leich, F. Wielgorz, and S. Apel, "Featureide: A tool framework for feature-oriented software development," in *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 2009, pp. 611–614.

[22] M. Scholtes, L. Westhofen, L. R. Turner, K. Lotto, M. Schuldes, H. Weber, N. Wagener, C. Neurohr, M. H. Bollmann, F. Körtke *et al.*, "6-layer model for a structured description and categorization of urban traffic and environment," *IEEE Access*, vol. 9, pp. 59 131–59 147, 2021.

[23] T. Pett, T. Thüm, T. Runge, S. Krieter, M. Lochau, and I. Schaefer, "Product sampling for product lines: the scalability challenge," in *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A*, 2019, pp. 78–83.

[24] M. Varshosaz, M. Al-Hajjaji, T. Thüm, T. Runge, M. R. Mousavi, and I. Schaefer, "A classification of product sampling for software product lines," in *Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1*, 2018, pp. 1–13.

[25] M. F. Johansen, Ø. Haugen, and F. Fleurey, "Properties of realistic feature models make combinatorial testing of product lines feasible," in *Model Driven Engineering Languages and Systems: 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings 14*. Springer, 2011, pp. 638–652.

[26] M. F. Johansen, O. Haugen, and F. Fleurey, "An algorithm for generating t-wise covering arrays from large feature models," in *Proceedings of the 16th International Software Product Line Conference-Volume 1*, 2012, pp. 46–55.

[27] S. Krieter, T. Thüm, S. Schulze, G. Saake, and T. Leich, "Yasa: yet another sampling algorithm," in *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems*, 2020, pp. 1–10.

[28] M. Papageorgiou, M. Leibold, and M. Buss, *Optimierung*. Springer, 2015, vol. 4.

[29] P. Skruch, M. Szelest, and P. Kowalczyk, "An approach for evaluating the completeness of the test scenarios for the vehicle environmental perception-based systems," in *2021 25th International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE, 2021, pp. 133–138.

[30] H. Nakamura, H. Muslim, R. Kato, S. Préfontaine-Watanabe, H. Nakamura, H. Kaneko, H. Imanaga, J. Antona-Makoshi, S. Kitajima, N. Uchida *et al.*, "Defining reasonably foreseeable parameter ranges using real-world traffic data for scenario-based safety assessment of automated vehicles," *IEEE Access*, vol. 10, pp. 37 743–37 760, 2022.

[31] L. Morell, "A theory of fault-based testing," *IEEE Transactions on Software Engineering*, vol. 16, no. 8, pp. 844–857, 1990.

[32] I. Pill, I. Rubil, F. Wotawa, and M. Nica, "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, 2016, pp. 168–173.

[33] J. Campos, A. Arcuri, G. Fraser, and R. Abreu, "Continuous test generation: Enhancing continuous integration with automated test generation," in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, 2014, pp. 55–66.