Scenario-based validation of self-learning systems using metamorphic relations

Marco Stang, Martin Sommer and Eric Sax

{marco.stang, ma.sommer, eric.sax}@kit.edu

Abstract: Numerous applications in our everyday life use artificial intelligence (AI) methods for speech and image recognition, as well as the recognition of human behavior. Especially the latter application represents an interesting research field for self-learning systems based on AI methods in the automotive domain. Human driving behavior is determined by routines that an AI system can learn, thereby predicting future actions. However, the methods and tools for validating these systems are insufficient and need to be adapted to the new types of self-learning algorithms. In this paper, we present a concept for validating self-learning systems based on behavioral scenarios, which was extended by metamorphic relations. The concept integrates use case testing, state transition, interface and requirement testing with metamorphic relations to create initial and subsequent test cases. A proof of concept is performed using the example of a self-learning comfort function in a vehicle. The correct functionality is shown by comparing the generated test cases. The concept addresses the main challenges in testing self-learning systems, in particular the generation of test inputs and the creation of a test oracle.

1 Introduction and Motivation

With the growing use of self-learning system (SLS) in different applications in our daily lives [SSM+22], we are increasingly confronted with the decisions made by the systems and the resulting consequences that can have far-reaching impact on us [J.M20]. Therefore, the explainability through testing is of high significance to generate trust with the customer. Traditional testing methods do not apply for SLS as they follow data-driven programming paradigms where the decision logic is not implemented in program code but extracted from datasets. This means that code can be tested extensively, and the SLS still makes decisions that are not comprehensible because the input datasets for training the SLS have not been considered in the testing process. This fundamentally different nature and construction of SLS compared to traditional, deterministic and less statistically orientated software systems poses new challenges in the field of software testing which we are trying to tackle with the combination of metamorphic and scenario based testing.

2 State of the Art

2.1 Scenario-Based Testing

Scenario-based testing has become a central method in testing automotive systems. Thereby, scenarios are defined as the temporal development of scene elements. Within a scenario, the scenes consist of a pre-defined sequence that begins with a start scene. The different actions of road users represent the link between the scenes. Defined scenarios form the basis for generating relevant test cases for the System under Test (SUT) [Sur18]. In the research work in the PEGASUS project, the concept of scenarios was further detailed and concretized [PEG18]. A concept was developed that makes scenarios representable on three levels of abstraction. A distinction is made between functional, logical and concrete scenarios [ea17].While functional scenarios represent a level at which relevant properties are recorded and described in language, logical scenarios define parameter areas in the state space for these properties in the form of entities and relationships of the scenario. Concrete scenarios are ultimately actual manifestations of a logical scenario, i.e., they represent a fixed, selected set of parameters. A logical scenario can therefore be transformed into at least one concrete scenario [ea17]. Generating test cases is the main part of the scenario-based testing concept. Test cases are used to check concrete properties of the SUT against boundary conditions, such as input data, expected reactions or test instructions, which are checked before, during or after the test execution [Pfe20]. In the context of scenario-based testing, a test case can be understood as the concretization of a scenario, which contains evaluation criteria as further components that decide under which conditions a test case was passed [Sch17]. Furthermore, a test case includes the description of the requirements for the test execution, such as the requirement for accuracy or the time required to execute the test cases.

2.2 Metamorphic Testing

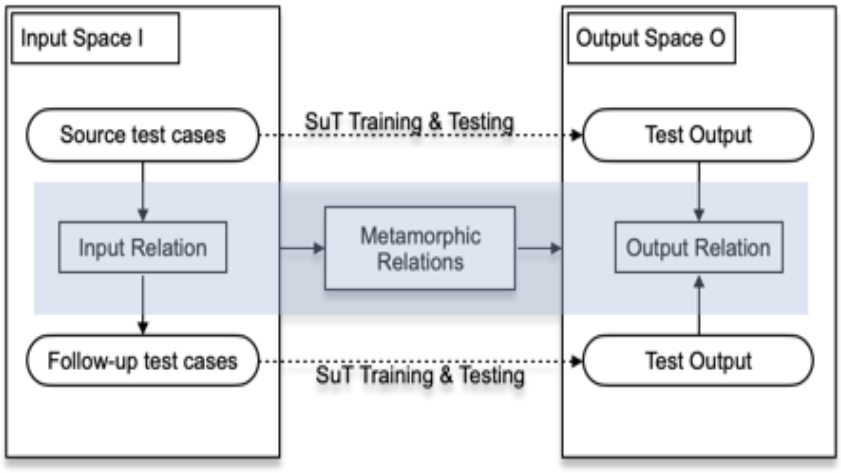

Figure 1: Principle of metamorphic testing [Z.H13]

The test procedure of metamorphic testing (MT) was first introduced in the work of Chen

et al. in 1998 with the aim of reusing successful test cases through transformation [T.Y98]. Figure 1 shows the principle of metamorphic testing with an Input-Dataset X (Original test case in Figure 1), which consists of the data points t of class I. This dataset is then split in a certain ratio into a training dataset D and a testing dataset K. Both datasets are used to train and test the model M in the SUT. The corresponding output of the ML system is called f(X). In the next step, a transformation is to be applied to this source input. This transformation is determined with the help of the so-called *metamorphic relations* (MR) and is expressed in the form of a transformation function T. Then, depending on the desired metamorphic relations, the transformation is applied only to the training dataset D or additionally to the test dataset K as well. Thus, the transformed datasets T(D) and T(K) result. Together, these form the new input dataset X', also known as follow-up input (cf. *follow-up test case* in Figure 1), for the new model M'. Accordingly, the result of the follow-up output is denoted as f(X'). The metamorphic relations describe expectations regarding the output relation between the source output and the follow-up output, more precisely the input relation between the source input and the follow-up input with respect to the applied transformation (cf. figure 1). These expectations, which also serve as a kind of pseudo-oracle [J.M20], must be met in order for the metamorphic test to be successful. For example, one expectation may be that the follow-up output after the transformation should not differ from the source output.

3 Concept

The concept aims to solve the challenges in testing self-learning systems with focus on user-specific behavior. In the following, the test process is presented and tested using the example of a commuter using a self-learning automotive seat massage function.

3.1 Scenario Generation

In the early phases of the software development process, real data is rarely available in sufficient quantity and quality. Prototypes for data acquisition or structures, such as automated data pipelines, do not yet exist or are under development. Since it is not goal-oriented to wait for a sufficient database during development, it is common practice to use data generation methods to accelerate the development of self-learning systems [SLP11]. However, the blind generation of large amounts of data is not sufficient to test self-learning software, as it might lead to the generation of scenarios not meaningful for verifying the functionality. Therefore, the requirements analysis for the generation of relevant scenarios became the first step of our flowchart illustrating the scenario-based validation of a self-learning system using metamorphic relations (Figure 2). This step summarizes the definition of logical and functional scenarios. For example, a logical scenario can be a typical commuter who drives from home to his workplace every morning. The following step defines the features to be generated synthetically and the APIs to be used for this scenario. For the challenge of data generation, a concept named CAGEN has been developed in a previous work by the authors [SMS21]. CAGEN is an acronym for Context Action

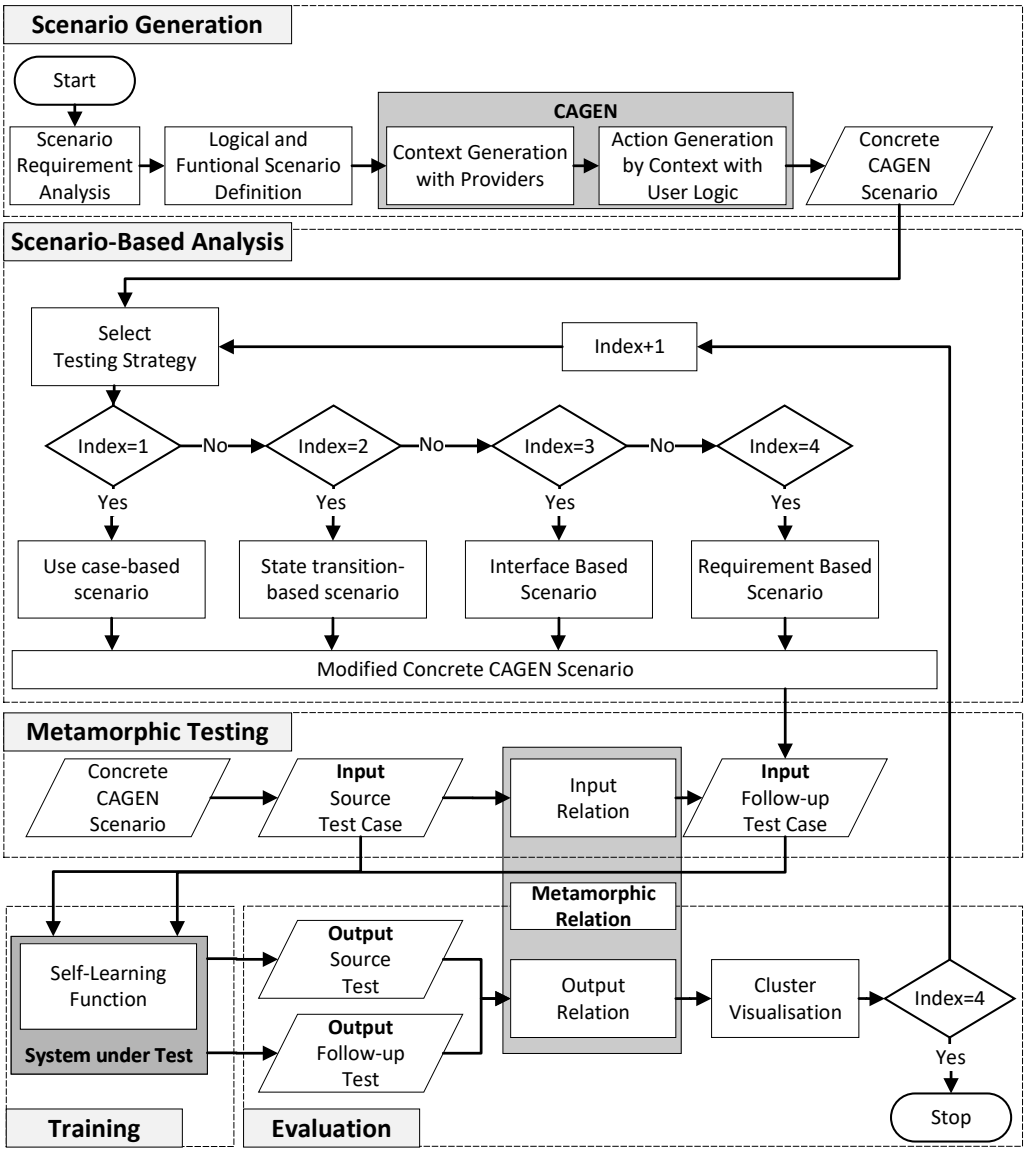

Figure 2: Overall Flowchart of the scenario-based validation of a self-learning systems using metamorphic relations

GENeration. Context-based data refers to any information about the environment, for example GPS-coordinates, weather or time. Action generation designates user-specific (i.e. commuter) and context-based (i.e. weather) interactions with the system. For the context generation, so-called providers are used. A provider can provide data through Web APIs or recorded datasets. The providers have the ability to build on one another, i.e., Provider B gets input from Provider A. To give an example, the tunnel provider simulates the loss of the GPS signal while driving through a tunnel and the recovery of the signal at the tunnel exit. The tunnel provider therefore needs information about the position of the simulated vehicle, which is supplied by the position provider. The position provider is not dependent on any other provider, but provides the longitude and latitude of the vehicle through an external API (OpenRouteService[1]). Based on this context data, user-specific

[1]https://openrouteservice.org/

actions are generated. The most straightforward approach uses random actions at random positions on the route. This way, it is possible to create multiple actions with a low time effort, but without the possibility to influence the events. The second approach is the generation of actions by predefined rules. An event is created if a certain rule is met or not (e.g., if temperature is higher than 30 °C, the massage will be activated). With this approach, actions can be tailored to a specific type of user. The most time-consuming type of event creation is the complete manual generation of actions. In this case, a human operator takes into account the given context and creates realistic actions according to the expectations.

3.2 Scenario-Based Analysis

Once the concrete CAGEN scenarios have been created, the following four test strategies are presented with the goal of modifying the dataset. The modifications affect the context, but also the user-specific actions (user logic).

1. **Use case-based scenario:** The use case describes the behavior of the system (test object) from an external perspective, e.g., the user perspective. Such a test technique covers more than one feature of the test object, allowing the interaction with this characteristic to be tested. The technique describes the behavior of the system in situations of user interaction. The different cases include:
    - Regular Cases - the behavior as intended
    - Special Cases - alternative or optional behavior. For example, if a child seat is placed on the car seat
    - Error Cases - the behavior when errors occur. For example, if the user activates massage level 1 and the system activates massage level 5
    - Misuse Cases - the user operates the system in a way it was not designed for. For example, turning the massage on and off multiple times within a few minutes.
2. **State transition-based scenario:** Technical systems assume certain states, e.g., on, off, startup, shutdown, normal operation, etc. The behavior of a system usually differs between several states. Therefore, it is necessary to test the system in all possible states. The transition between states is also of interest, as there are several ways to achieve or change it. The goal is to check whether states and transitions are implemented logically or not. Considering the window as a test object, the event would be the opening and closing of the window. The interface can be considered as a door system that reports the state of the driver's door (open/closed).
3. **Interface Based Scenario:** Interface Based Testing focuses on the test object's input and output interfaces. The test bases include interface requirements, CAN catalog, diagnostic database, etc. Interface, in particular, refers to the dependencies in features: context or events, pair-wise, three-wise combinations, etc. For example, the window lifter interface, defined by a CAN communication matrix for an ECU,

consists of eight boolean inputs/parameters. A single window was considered, but the CAN interface compels to test all four windows and its eight value-derived test cases.

4. **Requirement Based Scenario:** The approach is to analyze requirements (n) and develop one or more test cases (m) to check the fulfillment of the requirement by the test object using n to m mapping. Requirements can generally be found in several artifacts, e.g., specifications of design, models, and standards. It can be derived intuitively from the most straightforward scenarios, starting from an assumption about what a person would do at a particular location. Considering data as a sea of input, the events can be regarded as a class, and at least one test case should be picked to conform to the behavior while evaluating the same. This technique is called Equivalence class partitioning.

After one of the test strategies described above has been executed, a modified concrete CAGEN scenario is available (Table 1). This dataset will be referred to as the input follow-up test case in the following phase. The concrete CAGEN scenario from the "Scenario Generation" section is used as the input source test case.

Latitude	Longitude	Time	Temperature	Passenger Count	Massage Level
48.710226	8.99562	28.04.2022 06:34:13	5.1	1	0
48.710147	8.99551	28.04.2022 06:36:13	5.6	1	3
48.710031	8.99543	28.04.2022 06:38:16	5.9	1	3

Table 1: Exemplary modified concrete dataset based on CAGEN and User-Logic. Latitude, Longitude, Time, Temperature, and Passenger Count represents context data. Massage level is an example of a user event.

There are various MRs which include different transformations, for example, addition or subtraction of the data points, adding data points to the training set or removing one of the action labels etc. The proof of the correctness of the system under test is provided by verifying the validity of MRs. As an example of an Input Relation, the change of time context can be considered. The Modified Concrete Scenario consists of generated context data and the behavior of a commuter (Input Source Test Case). Using scenario-based analysis (i.e., use cases), the MR was defined as the commuter behaving the same way on different workdays.

3.3 Training and Evaluation

Once the metamorphic testing phase is complete, the self-learning system (in this context the SUT) is trained with both test input datasets (source and follow-up test case). The output of the SUT contains two datasets, grouping data points into classes based on the provided dataset. The test outputs form an output relation to each other.

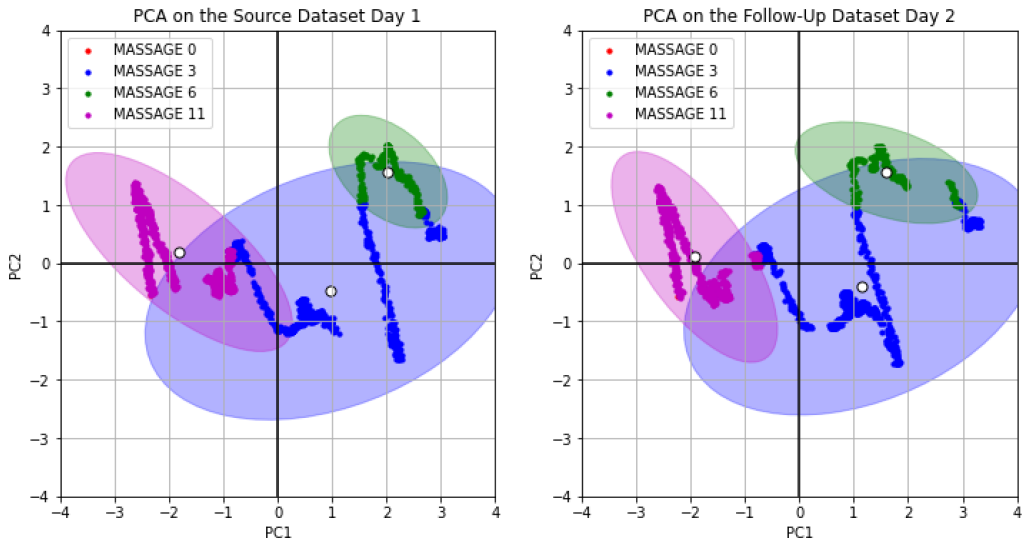

Figure 3: PCA of the Output Source Test (left, Monday) and the Output Follow Up Test (right, Wednesday) for a massage function. The values following the massage stand for the different intensities of the massage

The applied transformation (see section 2.2) that leads to the visualization of Figure 3 is the change of the time context between source and follow-up test case by two days. This behavior can be defined for example via the test strategy "Use case-based Scenario" from Phase Scenario Based Analysis and states that the behavior of the simulated driver is identical on both days, namely the source test case on a Monday and the shifted follow-up on a Wednesday. For the visualization, a Principal component analysis (PCA) was chosen that is capable of showing the predicted data clusters even for higher dimensional datasets. Since the set of contexts contains five features (see Table 1), the visualization of the data in the form of two-dimensional diagrams is not possible. Accordingly, PCA maps the data into a less dimensional coordinate system that attempts to consider as much information as possible. It can be observed in Figure 3 that the clusters, i.e. the behavior of the self-learning system, are almost identical for the source and the follow-up test case, from which it can be deduced that the metamorphic relation is fulfilled.

4 Conclusion and Summary

A workflow called 'Scenario-based metamorphic testing' using scenario-based and metamorphic testing has been presented. The concept focuses on overcoming the two major challenges in testing self-learning systems, namely the generation of test inputs and the test oracle problem. The test input generation was investigated by the scenario-based generation of data sets by CAGEN. The advantage of combining these two techniques is the use of defined tests through scenarios and the flexibility of metamorphic tests which do not require any labels. The evaluation based on the visualization of the source and follow-up datasets allows considering whether a metamorphic relation has been revealed or broken.

5 Acknowledgements

We would like to thank the student assistant Neeti Kumari for her work and commitment to this project.

References

[ea17] Menzel et al. Szenarien für entwicklung, absicherung und test von automatisierten fahrzeugen. In *11. Workshop Fahrerassistenzsysteme. Hrsg. von UniDAS e. V*, pages 125–135, 2017.

[J.M20] J.M. Zhang, M. Harman, L. Ma, Y. Liu. Machine learning testing: Survey, landscapes and horizons. In *IEEE Trans. Softw. Eng.*, 2020.

[PEG18] PEGASUS. https://www.pegasusprojekt.de/files/tmpl/PDF-Symposium/04_Scenario-Description.pdf, 2018. Accessed: 2022-09-02.

[Pfe20] Raphael Pfeffer. *Szenariobasierte simulationsgestützte funktionale Absicherung hochautomatisierter Fahrfunktionen durch Nutzung von Realdaten*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2020.

[Sch17] Fabian Schuldt. *Ein Beitrag für den methodischen Test von automatisierten Fahrfunktionen mit Hilfe von virtuellen Umgebungen*. PhD thesis, Technische Universität Braunschweig, Apr 2017.

[SLP11] Kaarthik Sivashanmugam, Da Lin, and Senthil Palanisamy. Scenario Driven Testing. In *2011 Eighth International Conference on Information Technology: New Generations*, pages 299–303, 2011.

[SMS21] Marco Stang, Maria Guinea Marquez, and Eric Sax. CAGEN - Context-Action Generation for Testing Self-learning Functions. pages 12–19, Cham, 2021. Springer International Publishing.

[SSM+22] Marco Stang, Simon Stock, Simon Müller, Eric Sax, and Wilhelm Stork. Development of a self-learning automotive comfort function: an adaptive gesture control with few-shot-learning. In *2022 International Conference on Connected Vehicle and Expo (ICCVE)*, pages 1–8, 2022.

[Sur18] Sebastian et al. Surmund. Neue Szenarien für autonome Fahrsysteme. *ATZextra*, 23(2):42–45, 2018.

[T.Y98] T.Y. Chen, S.C. Cheung, S. Yiu. Metamorphic testing: a new approach for generating next test cases. In *Tech. Rep. HKUST-CS98-01, Dept. of Computer Science, Hong Kong University of Science and Technology*, 1998.

[Z.H13] Z.Hui, S.Huang, Z.Ren, Y.Yao. Metamorphic testing integer overflow faults of mission critical program: A case study. In *Mathematical Problems in Engineering*, 2013.