








TSC2CARLA: An abstract scenario-based verification toolchain for automated driving systems

Philipp Borchers, Tjark Koopmann , Lukas Westhofen *, Jan Steffen Becker ,
Lina Putze , Dominik Grundt , Thies de Graaff , Vincent Kalwa,
Christian Neurohr 

German Aerospace Center (DLR) e.V., Institute of Systems Engineering for Future Mobility, Oldenburg, Germany

ARTICLE INFO

Keywords:

Automated driving systems
Verification
Abstract scenarios
Simulation
Constraint solving

ABSTRACT

Transitioning automated driving systems to complex operational domains disproportionately increases demands on verification activities. In the worst case, the operational domain can not be covered by a manageable set of logical scenarios. An anticipated solution is to use abstract scenarios, which increase coverage while still enabling formal methods. However, established verification approaches must be adapted for abstract scenarios. In this work, we consider the generation of simulatable test suites from abstract scenarios. For this, we use Traffic Sequence Charts (TSCs), a visual yet formal scenario description language based on first order logic. We propose an SMT-based process for generating concrete test cases that can be simulated in e.g. CARLA. This theoretical framework is compiled into an architecture and a prototypical implementation called TSC2CARLA. An evaluation on a set of non-trivial examples yields initial evidence for the feasibility of our approach.

Code metadata

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v0.1
C2	Permanent link to code/repository used for this code version	For inquiries on accessing the software, please send a message to the authors
C3	Permanent link to Reproducible Capsule	https://doi.org/10.5281/zenodo.14502996
C4	Legal Code License	see above
C5	Code versioning system used	Git
C6	Software code languages, tools, and services used	Python, Java
C7	Compilation requirements, operating environments and dependencies	Windows, Z3, CARLA
C8	If available, link to developer documentation/manual	No additional documentation available
C9	Support email for questions	jan.becker@dlr.de

* Corresponding author.

E-mail addresses: philipp.borchers@dlr.de (P. Borchers), tjark.koopmann@dlr.de (T. Koopmann), lukas.westhofen@dlr.de (L. Westhofen), jan.becker@dlr.de (J.S. Becker), lina.putze@dlr.de (L. Putze), dominik.grundt@dlr.de (D. Grundt), thies.degraaff@dlr.de (T. de Graaff), vincent.kalwa@dlr.de (V. Kalwa), christian.neurohr@dlr.de (C. Neurohr).

<https://doi.org/10.1016/j.scico.2024.103256>

Received 21 June 2024; Received in revised form 5 November 2024; Accepted 9 December 2024

Available online 13 December 2024

0167-6423/© 2024 The Author(s).

Published by Elsevier B.V. This is an open access article under the CC BY license

(<http://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In the automotive domain, testing is an integral part of verification and validation. This also applies to the emerging, increasingly complex driving automation that is built into vehicles [1]. With automated driving systems (ADSs), i.e. SAE Level ≥ 3 [2], drivers can delegate the responsibility for the driving task entirely to the automated vehicle within the limits of the ADS’s operational design domain (ODD) – with ODD being defined as the “operating conditions under which a given driving automation system [...] is specifically designed to function” [2]. However, this increases the burden on testing: It can not be assumed that the driver will act as a fallback in case of system failure. Thus, tests have to be vastly more comprehensive than for advanced driver assistance systems (ADAS), i.e. SAE Level ≤ 2 . As state-of-the-art ADS-equipped vehicles rely on various sensing technologies for perception, they receive a large part of the real world as input over time. Even ODDs that are spatially quite limited, such as a small district of a densely populated urban area, can lead to an uncountable infinite amount of situations and scenarios [3] that can not be enumerated on a concrete level.

If the ODD or the functionality is simple enough, logical scenarios can often be used to finitely represent the infinite set of concrete scenarios contained within the ODD. Logical scenarios, as defined Menzel et al. [4, Sect. IV.B], represent scenarios on a state space level using real-valued intervals as parameter ranges and, optionally, probability distributions for these parameters. However, with the aspired complex and ever-changing ODDs, the problem of having infinitely many (even logical) test cases can persist. This is due to the imperative semantics employed by logical scenarios – if something is not specified within the scenario, it is not included in the test case. This leads to the necessity of knowing the full ODD during specification, which is, for complex and changing ODDs, often impossible.

Therefore, it is imperative to use a declarative test case specification mechanism that is able to cover complex test spaces with a finite number of test cases but still, just like logical scenarios, enables an efficient derivation of concrete tests. To this, testing based on abstract scenarios (in the sense of Neurohr et al. [5]) is a viable solution. While their declarative semantics trivially allows to finitely cover all possible test cases (by not constraining anything at all), generating concrete test cases from them is not as straight-forward as for logical scenarios [6]. Moreover, the generated samples must vary sufficiently from each other to provide meaningful evidence for test evaluation. Naturally, only few efforts have been observed in testing based on abstract scenarios [7,8].

For addressing this gap, our work tackles the following research questions:

RQ1 How can we devise an efficient test suite generation procedure based on abstract scenarios?

RQ2 How can we measure test suite quality prior to its execution?

RQ3 How can we ensure a high degree of quality of the generated test suite?

For answering these questions, we rely on Traffic Sequence Charts (TSCs) as a visual yet formal specification language for abstract scenarios [9–11]. Our main contribution is an automated generation and execution of test cases based on TSCs. More specifically, we contribute:

1. A formal method to sample concrete scenarios from abstract scenarios based on a translation from (a subset of) TSCs to a Satisfiability Module Theories (SMT) formula (named $TSC2SMT$), which extends prior work [12,13], and a procedure to create and execute test suites from TSCs based on $TSC2SMT$.
2. Means for measuring and increasing the quality of the generated test suite, based on a formal notion of distances between concrete scenarios.
3. The $TSC2CARLA$ toolchain, an architecture and prototypical implementation of the prior contributions, which is evaluated on a non-trivial, abstract test scenario in an urban setting and an ADS implemented for the CARLA simulator.

We put our work into the context of the current scientific literature in section 2 and continue with the preliminaries in section 3, including scenario-based testing and TSCs as our formal foundation. The theoretical method for test suite generation and execution based on TSCs is subsequently presented in section 4. We introduce in section 5 the $TSC2CARLA$ toolchain as an architecture together with technical details on its prototypical implementation. In section 6 we first evaluate test suite generation for three abstract urban traffic scenarios by comparing different variation methods by means of a test suite quality measure. Two of these test suites are subsequently executed with an ADS implemented for the CARLA simulator and evaluated using criticality metrics to demonstrate the applicability of our implementation for testing purposes. Current limitations of $TSC2CARLA$ and future work are addressed in section 7, before concluding with section 8.

2. Related work

Sampling logical scenarios for test scenario generation is state of the art [6, Section III D. 2]. While their parameter spaces can, in theory, be sampled in a Monte Carlo approach, the sheer size makes such a procedure highly inefficient. Therefore, much research effort has been put into finding effective sampling methods, e.g. based on reinforcement learning [14], rare-event simulation [15] or evolutionary algorithms [16]. There even exist implementations that can perform testing based on a full suite of search procedures [17], including various genetic algorithms.

However, with increasing complexity of the ODD, the number of logical scenarios required to cover it exhaustively is likely to grow exponentially. The underlying assumption with logical scenarios is that either the given set of logical scenarios is incomplete

w.r.t. the ODD or the ODD is small enough to be covered completely by a manageable amount of logical scenarios. Thus, if we aim for increased test coverage in large and complex ODDs, we require a more abstract notion of scenarios. For example, Bock et al. [18] propose a controlled natural language for abstract scenario specification and Song et al. use Natural Language Processing to directly create concrete scenarios from natural language specifications [19]. Moreover, the System Co-Design for Open Context Analysis [20] uses topological graphs for abstracting concrete geometries. These approaches do not use formal logics underlying their scenario description language. Most prominently, this is currently addressed by ASAM OpenSCENARIO DSL (originating from the Measurable Scenario Description Language) [21]. It partially defines a declarative trace acceptance semantics, however, details are not yet clear. To the best of the authors' knowledge, there are only two fully-defined formal approaches: The first is Scenic [22,8], in which abstract scenes – and with version 2, also dynamic behavior and temporal constraints – can be defined based on a structural operational semantics (SOS) similar to probabilistic programs. It thus incorporates probabilistic-imperative and declarative aspects, where any abstract scenario specified in Scenic represents a distribution over scenes and behaviors. Due to its structural operational semantics, reasoning over Scenic scenarios is inherently coupled to the ability of sampling behaviors. The second approach, TSCs as introduced in section 3 [11], are a visual yet formal language and rely on a denotational semantics by translation to first order logic. In contrast to Scenic, TSCs allow for concise, graphical scenario specifications as well as formal analyses even if there is no access to simulation models [12]. Having a visual formalism allows experts with diverse backgrounds to incorporate knowledge in the design process early on while keeping precise semantics. This is in contrast to textual specifications, used in Scenic for example.

Naturally, the idea arises to sample from abstract scenarios similarly to logical ones. For informal languages, sampling is typically based on combining discrete sets of options given by an ontology together with a set of rules on creating valid scenarios [23,24]. For formal languages, sampling can be directly based on the semantics of the underlying formalism. To the best of our knowledge, four approaches are currently pursued: First, prior work of the authors describes how sampling of TSCs can be performed using linear SMT solving [13]. A second strategy is using non-linear SMT solving, which has already shown promising results for runtime verification of requirements in concrete scenarios [25]. This was examined by Eggers et al. and Scheibler et al., who propose to sample from scenarios described as constraint systems [7,26]. Their work focuses on sampling concrete scenarios and only briefly mentions challenges when extending this approach for testing. Thirdly and in contrast to the SMT-based approaches, Klischat, Finkeldei and Althoff propose the use of (mixed-integer) quadratic programming for generating concrete scenarios from non-linear constraint systems, yet focus solely on sampling without further integration into downstream (e.g. testing) procedures [27,28]. Finally, Scenic was designed with scene generation as its main purpose and thus sampling from an abstract Scenic scene is reduced to performing rejection sampling on the distribution defined by the program (after pruning invalid states) [22].

We are aware of only one published article on this work's main subjective, i.e. incorporating abstract scenario sampling into a testing toolchain. Fremont et al. used Scenic version 2 to sample, at each point in time, a valid scene configuration [8]. This simulation state is then fed into probabilistic agents specified in Scenic (that may constrain an underlying simulation model), which are executed in the simulator to generate a new state. If the new state does not adhere to user-specified (possibly temporal) constraints, it is rejected. Finally, all non-rejected runs are compared against a safety property specified in a temporal logic for test evaluation. Our work explores an orthogonal approach leveraging the independence of TSCs from simulators. Instead of intertwining scene-by-scene sampling with the execution of simulation models we separate sampling from simulation. This means, we first sample from the abstract scenario as a whole and subsequently evaluate the system under test (SuT) in a simulation. In contrast to the works of Fremont et al., our approach is able to generate stand-alone suites of concrete scenarios (without access to a high-fidelity simulation model), which can be reused when testing different systems.

3. Preliminaries

Here, we provide the necessary background knowledge for our approach. The concept of scenario-based testing for automated driving is summarized in subsection 3.1, followed by a brief introduction to the formalism of TSCs in subsection 3.2.

3.1. Scenario-based testing of automated driving systems

In the automotive domain, any safety-critical system requires passing a systematic verification and validation process. Testing, defined as the 'process of planning, preparing, and operating or exercising an item or element to verify that it satisfies specified requirements, to detect safety anomalies, to validate that requirements are suitable in the given context and to create confidence in its behavior' [29], constitutes a central element for both verification and validation. With the increasing degree of automation, the responsibility during the operation is gradually shifted from the driver to the vehicle. ADSs, i.e. SAE Level ≥ 3 , need to be capable of handling the complexity of arbitrary situations in an open, real-world environment. This implies enormous challenges for testing such systems. As argued by Kalra et al., a purely distance-based approach is not feasible due to the amount of miles that would be required to provide significant statistical evidence [30]. Scenario-based testing offers a promising alternative and has been recently pursued by various research projects such as VVM¹ and SET Level.² Here, the complexity of the open context is managed by decomposition into representative scenario classes.

Actual strategies for scenario-based testing are diverse, but in general they correlate with some overarching steps [6,31–33]. Fig. 1 shows a simplified framework by Neurohr et al. [6], which highlights steps that are subject of this work. First, appropriate

¹ www.vvm-projekt.de/en.

² www.setlevel.de/en.

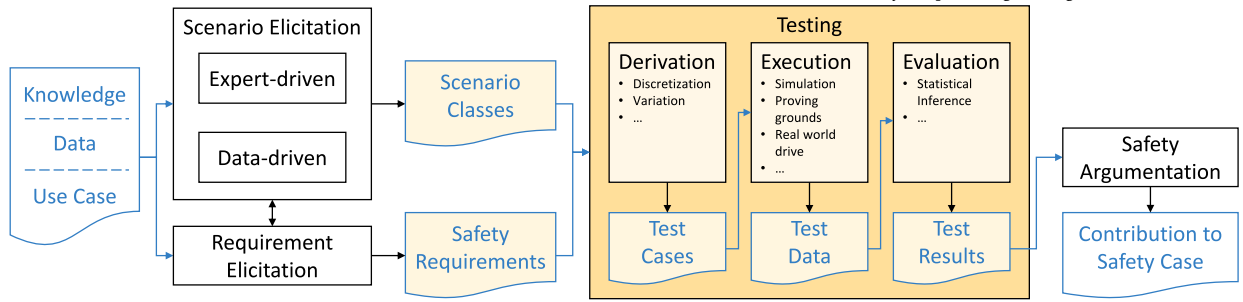


Fig. 1. Scenario-based testing workflow by Neurohr et al. [6], with steps examined in this work highlighted in yellow. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Table 1
Abstraction levels of scenario description based on [4,5].

Abstraction Level	Characteristics
Functional	<ul style="list-style-type: none"> • linguistic, behavior-based description • human readable
Abstract	<ul style="list-style-type: none"> • declarative description based on ontologies • formal, machine readable • efficient description of relations (e.g. cause-effect)
Logical	<ul style="list-style-type: none"> • imperative description • using parameter ranges and distributions over a state space • formal, machine-readable
Concrete	<ul style="list-style-type: none"> • imperative description • using concrete values for each parameter in the state space • formal, machine-readable, e.g. as OpenDRIVE/SCENARIO

scenario classes and requirements need to be derived. Both expert/data-driven approaches can be applied to identify relevant classes of scenarios as part of a comprehensive risk analysis. Testing within these scenario classes is divided into three steps: the derivation of test cases, their execution and an evaluation. Finally, the results are embedded into the safety argumentation.

It is evident that the term scenario is central to the overall process of scenario-based testing. Prior work has therefore given a common definition as the ‘temporal development between several scenes in a sequence of scenes [...]’ where a *scene* is ‘a snapshot of the environment including the scenery and dynamic elements, as well as all actors’ and observers’ self-representations, and the relationships among those entities [...]’ [3]. Moreover, several degrees of abstractions were introduced, as shown in Table 1. A *functional scenario* is given by a linguistic description while a *logical scenario* is defined in terms of parameter ranges in a given state space and a *concrete scenario* represents a single instance of a logical scenario with concrete values for each parameter [4]. Neurohr et al. complement these scenario categories by a fourth scenario category called *abstract scenarios* which is defined as a ‘formalized, declarative description of a traffic scenario’ [5]. Abstract scenarios enable the representation of complex relations which cannot be covered by a purely parameterized description, but unlike functional scenarios this scenario category ensures machine-readability and enables formal arguments. Thus, abstract scenarios fill the gap between functional and logical scenarios.

Depending on the stage of the development process, there exist different requirements on scenarios: In the concept phase, the development of a functional safety concept requires a common understanding across various collaborators. This already necessitates a semi-formal notion of scenarios. For the derivation of technical requirements during system development, a representation via parameter ranges is needed, while for the test process additional information about different relations like physical laws is necessary [4]. Due to concise specification, abstract scenarios have the advantage that they can be created early in the concept phase. Additionally, due to their formal semantics [34], they remain useful throughout the remainder of the development process, e.g. in system-level testing. Their concise representation is specifically ensured by modular compositions of multiple abstract scenarios and thus the scenario space can be described with comparatively low specification effort.

The remainder of this work focuses on two of these scenario abstraction levels. We employ 1) *abstract scenarios* for the specification of abstract test cases, cf. Definition 5, and derive from them 2) *concrete scenarios* as input for concrete test cases in the sense of Definition 1. Functional scenarios can be seen as an input for the creation of abstract scenario, as indicated by Fig. 8, but this is not central to this work. Logical scenarios are not used at all within TSC2CARLA.

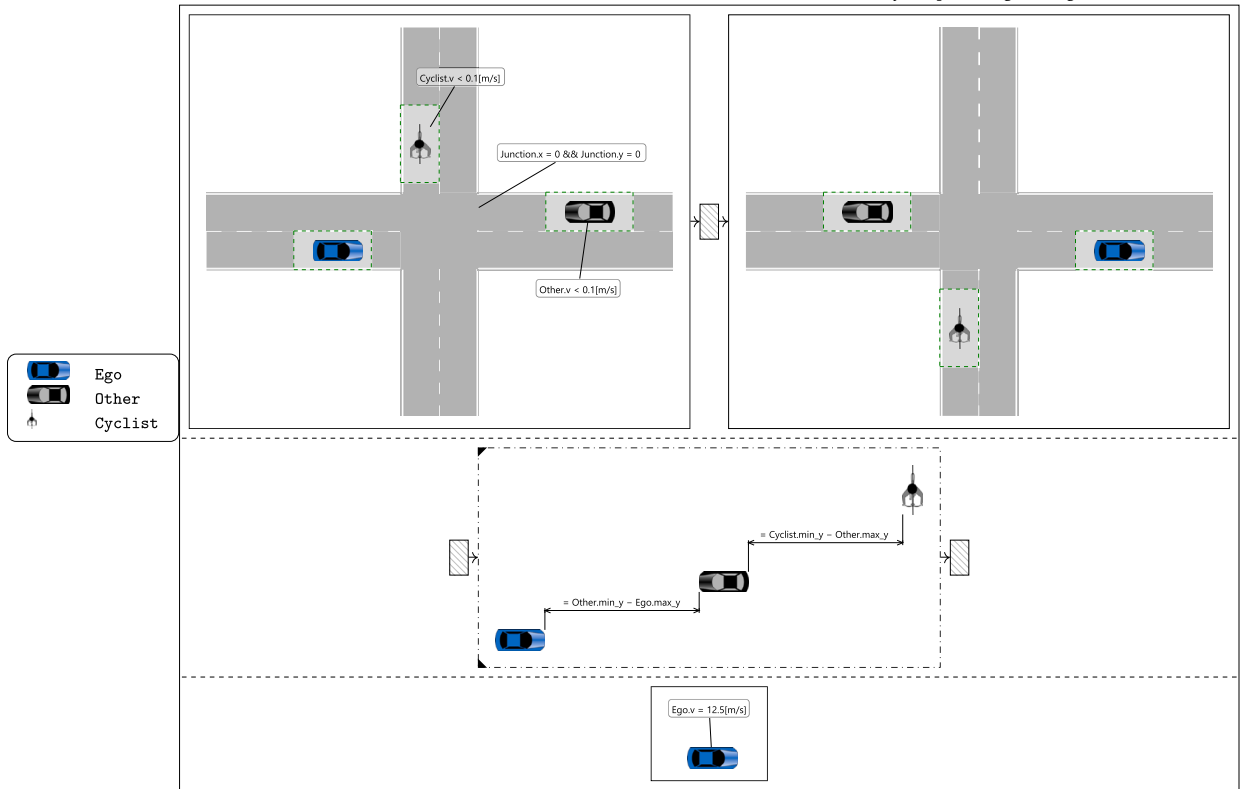



Fig. 2. A Traffic Sequence Chart depicting a dynamic occlusion between a bicyclist and a vehicle at a four-armed intersection.

3.2. Introduction to traffic sequence charts

TSCs are a formal specification language for abstract scenarios [11,10,35] and have been developed with the goal of creating a description language that connects the intuitiveness of depicting traffic situations graphically with well-defined, formal semantics. A specification with TSCs consists of three parts:

- The world model describes the domain ontology. This includes for example the different types of traffic participants, infrastructure elements and road types.
- The symbol dictionary defines the symbols that are used to denote the world model concepts in TSCs.
- The last part is the collection of TSCs itself that describes a traffic scenario with related constraints and requirements for the traffic objects defined in the world model and linked via the symbol dictionary.

In this paper, we use a single form of TSCs, called *scenario TSCs* or *existential TSCs*. A scenario TSC consists of a *bulletin board* that lists the objects used in the TSC and a *basic chart* describing the temporal and spatial aspects of the scenario. The interested reader can find a formal semantics definition for the used subset of the TSC language in Appendix A.

The simplest form of a basic chart is an *invariant node*. The single invariant nodes are inductively assembled to basic charts by combining them using operators like *sequence*, *choice* and *concurrency* (parallel composition). The operators can be arbitrarily nested. Furthermore, it is possible to add timing annotations. Fig. 2 shows a TSC specifying a scenario with a dynamic occlusion that serves as running example within this work. The basic chart of this TSC is a parallel composition of three other basic charts, which are graphically separated by the two horizontal dashed lines. The first two ones are sequences of three invariant nodes each, and the third one is a single invariant node. The top sequence describes a crossing scenario with two cars (Ego and Other) and a bicyclist (Cyclist). In the first invariant node of this sequence, all vehicles are in front of the crossroad (from their perspective). In the last invariant node, they are all behind the crossing. The hatched rectangle  in the middle denotes an empty invariant that allows any behavior. Because sequences in a TSC are seamless, undefined behavior must be marked explicitly. The second sequence in the middle describes that at some time during the scenario (from the perspective of Ego) an occlusion of the Cyclist by the vehicle Other occurs. By default, invariant nodes have a non-zero duration. The dash-dotted border of the middle node indicates that the depicted situation holds for a single time point. The invariant node at the bottom describes that the speed of the ego vehicle is constant with 12.5 m/s.

Inside an invariant node, a traffic situation can be defined as a so-called *spatial view*. Spatial relations are expressed graphically by placing symbols in a 2D space. The relative placement of the symbols to each other mimics the spatial relations in the traffic situations. Spatial views have well-defined semantics which enable their translation into mathematical formulae [11]. Definition 10 in Appendix A shows how to construct these formulae for the spatial views that appear in this work.

For example, the spatial view in the middle sequence of the running example can be translated to

$$\text{Other.min}_x - \text{Ego.max}_x = \text{Other.min}_y - \text{Ego.max}_y > 0 \wedge$$

$$\text{Cyclist.min}_x - \text{Other.max}_x = \text{Cyclist.min}_y - \text{Other.max}_y > 0$$

where the attributes $\text{min}_x, \text{max}_x, \text{min}_y, \text{max}_y$ denote the extrema of the vehicle's axis aligned bounding box for each vehicle $\text{Ego}, \text{Other}, \text{Cyclist}$.

4. Automated TSC-based test suite generation and execution

In this section, we delineate how TSCs can be used for generating and executing test suites for the verification and validation of ADSs in virtual environments.

4.1. General scope

The problem can be stated as follows: An ADS – the SuT – is to be virtually tested in the system-level verification and validation phase of a development process. For this, we assume as input a scenario catalog that can be specified with TSCs, originated from prior stages of the development process [34]. Due to the large input space for the SuT, we require a large amount of test cases to be executed in an automated fashion. These test cases are based on a finite and manageable set of abstract scenarios resulting e.g. from a hazard analysis [36] or a criticality analysis [5,37]. Moreover, the virtual testing shall be done in a high-fidelity simulation s.t. effects of system components, e.g. from sensors, can be considered as well.

4.2. Employed terminology

We first define the relevant terms for our testing approach. In what follows, we differentiate between concrete scenarios, i.e. simulatable inputs to the simulator (e.g. OpenSCENARIO XML files), and data traces, i.e. outputs of the simulator. A trajectory is a sequence of object states (e.g. position) as a function of time. It can be present in both concrete scenarios and data traces. More precisely, data traces consist only of trajectories, whereas concrete scenarios may additionally have maneuvers or other, more abstract instructions.

For our purpose, we adapt the test terminology provided by Pretschner and Leucker [38], which define a (concrete) test case as a tuple of input and expected output.

Definition 1 (Test Case). A (concrete) *test case* is a tuple of input and expected output. a concrete scenario with a possibly partial trajectory for the SuT (e.g. initial and target position). The expected output consists of a set of requirements, i.e. functions mapping from a data trace to $\{0, 1\}$.

Note that the possibly partial trajectory gives the SuT freedom to react to its environment (and thus enables to check whether the SuT adheres to the requirements). This definition can also be extended to allow the same for the other agents present in the concrete scenario. As a simplification, we assume that the input already contains a complete trajectory for the other agents and thus only the SuT behavior has to be simulated.

The main purpose of test cases is their execution, i.e. in general, the simulation of the SuT given the input of the test case.

Definition 2 (Test Execution). A test execution is the simulation of a test case input together with the SuT, which is given its partial trajectory as initial and time-dependent target values. The result of the executed test is a data trace.

After execution, we require a verdict: an evaluation of the execution of a single test case (e.g. pass, fail, unknown).

Definition 3 (Test Verdict). For an executed test case, a test verdict is the evaluation of the test case's requirements on the data trace resulting from the test execution.

Finally, we are interested in evaluating multiple test cases, which is called a *test suite*.

Definition 4 (Test Suite). A test suite is a finite set of test cases.

Instead of enumerating all test cases in a test suite, it can also be more concisely depicted, in its simplest form, by some requirements and a single TSC, which we call an abstract test case.

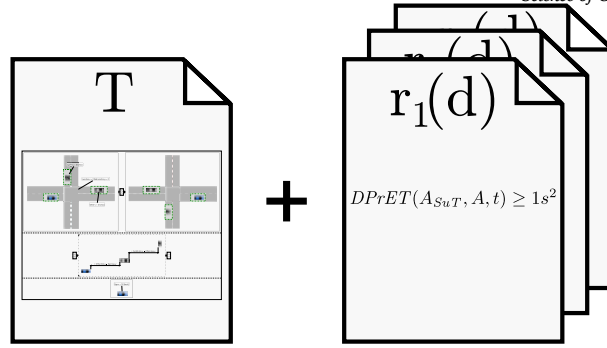


Fig. 3. Visualization of an abstract test case consisting of the Traffic Sequence Chart T and the requirements $r_1(d)$, $r_2(d)$, and $r_3(d)$. The Traffic Sequence Chart T represents the (infinite) set of concrete scenarios at four-armed intersections with a dynamic occlusion between a bicyclist and a vehicle, as introduced in subsection 3.2.

Definition 5 (Abstract Test Case). An abstract test case is a tuple consisting of an abstract scenario (the test input), i.e. a TSC, and a set of requirements, i.e. functions mapping from a data trace to $\{0, 1\}$ (the expected test output).

An abstract test case can be seen as a TSC that encompasses an (infinite) number of concrete scenarios, which represent the test suite. Note that in any of the concrete scenarios represented by the TSC, the SuT trajectory is fully defined. Therefore, an additional method for partialization (i.e. removing parts) of SuT trajectories can be supplied in an abstract test case. As an example, a partialization method could simply remove all but the initial and final state. Finally, we aim to test the SuT with a variety of TSCs, for which we use the term abstract test suite.

Definition 6 (Abstract Test Suite). An abstract test suite is a finite set of abstract test cases.

We now illustrate these definitions by an example and assume that our abstract test suite contains only one abstract test case. Recall that an abstract test case is a concise representation of a test suite, which again consists of concrete scenarios and requirements. We therefore require two ingredients, as shown in Fig. 3: A concise description of a set of concrete scenarios and requirements for the expected outcome.

For the former, we use the already introduced TSC of Fig. 2 representing a dynamic occlusion at a four-armed intersection. This TSC abstractly describes a set of concrete scenarios, i.e. the inputs to our simulator for test execution. Above, we highlighted that this TSC will lead to concrete scenarios where also the SuT trajectory is fully explicated, which leaves nothing for the simulator to simulate. In order to introduce degrees of freedom to the SuT trajectory, we additionally supply the simple partialization method of removing everything from the SuT trajectory except for the initial and final position.

The test requirements can also be specified as TSCs (by employing a logical implication of the form ‘if *situation* then *behavior*’ in the TSC chart structure) or as mathematical functions over a data trace d . For simplicity, the requirements in our example are defined on the basis of criticality metrics and respective threshold values:

$$r_1(d) := \min_{A \in \text{Actors}(d) \setminus \{A_{SuT}\}} \min_{t \in \text{Times}(d)} a_{\text{req}}(A_{SuT}, A, t) \geq -3.5 \frac{\text{m}}{\text{s}^2}, \quad (1)$$

$$r_2(d) := \min_{A \in \text{Actors}(d) \setminus \{A_{SuT}\}} \min_{t \in \text{Times}(d)} \text{DPrET}(A_{SuT}, A, t) \geq 1 \text{ s}^2, \quad (2)$$

$$r_3(d) := \min_{A \in \text{Actors}(d) \setminus \{A_{SuT}\}} \min_{t \in \text{Times}(d)} \text{TTC}(A_{SuT}, A, t) \geq 1 \text{ s}. \quad (3)$$

Here, A_{SuT} denotes the SuT, $\text{Actors}(d)$ all actors in d and $\text{Times}(d)$ the time points of d . As criticality metrics we rely on the Required Acceleration (a_{req}), the Duration-dependent Predictive Encroachment Time (DPrET), and the Time-To-Collision (TTC) defined according to Westhofen et al. [39, Sec. 5.2] and Fehnker [16, Section 4.3.1]. Intuitively, a_{req} measures the maximum required negative acceleration to reach zero velocity at a predicted collision. The criticality metric DPrET is a predictive version of the Post Encroachment Time which measures the time gap between two actors entering and leaving a conflict area, and scales it with a time duration, resulting in a s^2 unit. Lastly, the widely known TTC refers to the time remaining until a predicted collision, based on some dynamic motion models for the actors, e.g. a constant velocity model.

4.3. Process for TSC-based test suite generation and execution

As sketched above, the main idea behind our approach is to

- (1) automatically generate a test suite from a TSC-based abstract test case,
- (2) execute and evaluate such a test suite for an SuT,
- (3) aggregate all test verdicts.

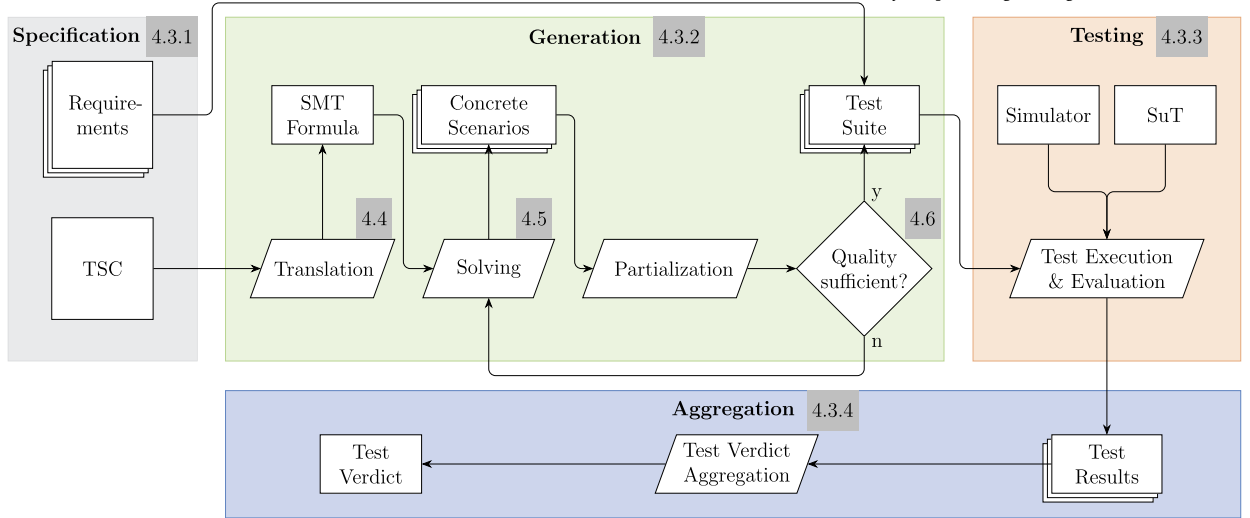


Fig. 4. The testing method proposed in this work. The Traffic Sequence Chart (TSC) is translated to a Satisfiability Module Theories (SMT) formula from which concrete scenarios are generated iteratively and prepared for testing until an exit condition holds. The resulting test suite is then executed in simulation and the performance of the system under test (SuT) is evaluated with regard to the requirements. Finally, the test verdicts are aggregated. Corresponding subsections of section 4 are annotated with a gray background.

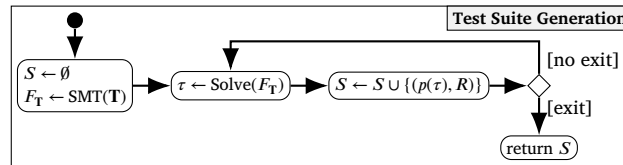


Fig. 5. Activity diagram for generating a test suite from an abstract test case. Expected inputs are a TSC T , a set of requirements R , a partialization function p , and an exit condition. $SMT(\cdot)$ creates an SMT formula from the given TSC (described in subsection 4.4), and $Solve(\cdot)$ returns a satisfiable solution to an SMT formula (described in subsection 4.5). The procedure returns a test suite S consisting of tuples of partialized trajectories and requirements.

Given a TSC-based abstract test case as input, our method outputs a test verdict (a comparison between the actual and required SuT behavior) for the abstract test case based on the test verdicts of the executed test cases within the generated test suite. This output is subsequently delivered to a test engineer. Our proposed method is shown in Fig. 4, along with the corresponding subsections that describe their concept. Note that an implementation of this method is presented in section 5.

4.3.1. Abstract test case specification

We assume that a functional scenario is given for which virtual testing has to be executed. The test engineer converts it to an abstract scenario, representing the input as a TSC, a world model and a symbol dictionary. It is then further enriched by test requirements. In addition, a partialization method for the SuT trajectory can be determined (as explained in subsection 4.2). The combination of the TSC specification, test requirements and partialization method constitutes the abstract test case.

4.3.2. Test suite generation

We propose the following procedure for test suite generation based on the enumeration of concrete scenarios for a TSC. For simplicity, we assume as input a single abstract test case, containing a set of requirements $R = \{r_1, \dots, r_k\}$, a TSC T , which is a finite representation of a possibly infinite set of concrete scenarios $\{\tau_1, \tau_2, \dots\}$, and a partialization method $p(\tau)$ that returns, for a concrete scenario τ , the concrete scenario with a partial SuT trajectory. We assume that p does not remove the initial position of the SuT (to allow for proper initialization of the SuT during simulation). Extending this to abstract test suites is straightforward by just applying our procedure sequentially on all abstract test cases in the suite.

Since the solution space of a TSC is possibly infinite, we require an exit condition for the sampling process. This condition is also provided by the user, e.g. based on an adequate test suite quality function (as discussed in subsection 4.6) or a fixed number of solutions.

Our proposed procedure for generating a test suite S is shown in the activity diagram of Fig. 5. The returned set $S = \{s_1, \dots, s_j\}$ consists of j test cases containing a partialized trajectory $p(\tau)$ and a set of requirements R to evaluate later on. Details on how we translate a TSC to a problem to SMT (step 2) are given in subsection 4.4 and variation methods for solving the SMT problem (step 3) can be found in subsection 4.5. These variation methods can also ensure the constraint $(p(\tau), R) \notin S$. A discussion of useful exit conditions based on measuring test suite quality (step 5) is given in subsection 4.6.

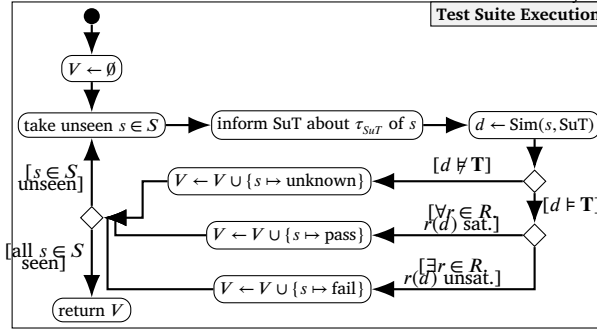


Fig. 6. Activity diagram for executing a test suite. Expected input is a test suite S (as generated in subsection 4.3.2) and output is a mapping from the test cases of S to one of the verdicts unknown, pass, or fail. The function $\text{Sim}(s, \text{SuT})$ simulates all non-SuT traffic participants according to s as well as the SuT itself.

4.3.3. Test suite execution

We now have a test suite S , i.e. a set of concrete scenarios where the SuT trajectory is partial. From here on, we call this partial trajectory τ_{SuT} . Note that the creation of a test suite is essentially independent of the given SuT, which reduces efforts during test definition. We will later see that when solving T , i.e. step 3 of the test suite generation, we assume only very basic properties of the SuT dynamics (such as restricting the acceleration to physically reasonable values). For executing a test suite when confronted with testing a given system, we now apply a simulator to execute the SuT and thus ‘fill in the gaps’ of τ_{SuT} . As input, we take the test suite $S = \{s_1, \dots, s_j\}$, where each element is a tuple of a concrete scenario s and a set of requirements R . The procedure is shown in the activity diagram of Fig. 6.

As the SuT has various freedoms in ‘filling the gaps’ during simulation, it may choose a trajectory that invalidates the given abstract test case [40]. To detect such non-instances of the abstract test case during simulation, we propose to use TSC runtime monitoring, which has been investigated in prior work and provides a verdict – as early as possible – on whether a concrete scenario satisfies or violates a TSC [41]. In detail, during a simulation run, relevant input data is analyzed with regard to spatial properties, which then provide information about the set of currently satisfied spatial views of a TSC specification. The temporal evolution of the satisfied spatial views up to the current point in time is then analyzed, leading to a verdict. If the monitor detects a non-instance, our testing framework rejects the run as inconclusive, i.e., the resulting trajectory does not provide a suitable foundation for assessing whether the SuT satisfies requirements. The satisfaction of requirements may only be evaluated within instances of the abstract test case. If the monitor evaluates the trajectory as a valid instance, it can consequently be checked whether the SuT satisfies its requirements.

4.3.4. Test verdict aggregation

We are now presented with a mapping from test cases to verdicts. However, we must assume to have an enormous number of test cases and thus require an aggregation of these verdicts. For example, we can return a simple ratio of passed test cases. More complexity can be added by grouping test verdicts by their features (e.g. high or low speeds), which allows further analysis by the test engineer (e.g. finding that most fail verdicts occur at high speeds).

4.4. Translation to satisfiability problems

It remains to describe the theoretical method for deriving concrete scenarios from TSCs (our abstract scenario formalism of choice). The first step is a translation from a TSC T to an SMT problem, which previously denoted by $\text{SMT}(T)$. We start with an introduction to the basic notation of SMT formulae.

In the following we denote by $\ell = \ell(c, x)$ linear constraints of the form $c^T x \sim d$ where, $c \in \mathbb{R}^k$ and d are real constants, $x = (x_1, \dots, x_k)$ is a vector of real variables, $\sim \in \{<, \leq, =\}$, and $k \in \mathbb{N}$.

Definition 7 (Models of SMT Formulae). A model \mathcal{M} of F , denoted by $\mathcal{M} \models F$, assigns a value to all free variables of F such that F is satisfied under this assignment. In particular, $\mathcal{M}(x) = (\mathcal{M}(x_1), \dots, \mathcal{M}(x_k))^T \in \mathbb{R}^k$ refers to assigned value for a vector $x = (x_1, \dots, x_k)^T$ of real variables while $\mathcal{M}(b) \in \{0, 1\}$ is the assigned value for a Boolean variable b . Likewise, given some atom (or Boolean combination of atoms) a of F , we write $\mathcal{M} \models a$ if a is satisfied under the assignment of \mathcal{M} , and $\mathcal{M} \not\models a$ otherwise.

TSCs specify constraints on traffic participants and scenes based on an abstract description of scenario phases to cover a wide set of possible situations. Since TSCs are semantically defined via first order logic, a (significant) subset can be encoded as SMT formulae with the object states encoded as free variables. Then, an SMT solver generates a model, also called *instance*, (if possible) and its assignments of values to free variables can be used to derive a concrete scenario. Prior work of the authors already describes this approach for obtaining a concrete scenario via SMT [13,42]. Thus, we only shortly re-iterate the essential steps of

1. encoding spatial views and vehicle dynamics as linear constraints,
2. translating temporal constraints into time slices,

3. unrolling both into a bounded model checking problem and
4. using Bézier interpolation to generate smooth trajectories within time slices.

First, the spatial views in any invariant node are translated into a corresponding linear constraint. These include variables for position, velocity, acceleration and size of cars, pedestrians as well as roads and lanes. For our running example of Fig. 2, an excerpt³ of the formula extracted from the first invariant is

$$\text{VehicleA.y} + \text{VehicleA.bb_y_min} > \text{WEELane.y} - 0.5 \cdot \text{WEELane.width} \quad (4)$$

encoding that `VehicleA`'s southernmost point must be above the southernmost of the lane `WEELane`, which is the lane on the western side of the junction with a driving direction to the east.

Secondly, the temporal constraints of the chart structure are encoded in the SMT formula using a bounded model checking approach that slices the time domain into equally sized intervals (steps). In our experiments, we use a step size of 1 s. In essence, this means defining valid configurations for step $i + 1$ based on the information present at some step i . For example, we assert that, in any step i , each invariant of the upmost sequence (called A) must have been completed before the start of the next invariant (called B):

$$\begin{aligned} \text{started}_B^{i+1} &\implies (\text{complete}_A^{i+1} \vee \text{started}_B^i) \\ \text{started}_B^i &\implies \text{started}_B^{i+1} \end{aligned}$$

Here, started_B^i denotes that B has been started and complete_A^i indicates the possibility to leave A . Because we assert only an implication, not an equivalence, the start point for B can be freely chosen among all steps where complete_A^i is true; in fact, there is no obligation to start B at all. However, the second implication ensures that B is started at most once, which is important for correct parallel composition of charts. Moreover, the temporal constraints must be connected to their encoded invariants. An invariant node can be left if, and only if, its invariant (denoted by inv_X^i for invariant node X) is satisfied since its start. For this, we introduce the variable ok_X^i tracking the satisfaction state of X 's invariant constraints. In our example, this leads to

$$\begin{aligned} \text{ok}_A^{i+1} &\iff (\text{started}_A^i \implies \text{ok}_A^i \wedge \text{inv}_A^i) \\ \text{complete}_A^{i+1} &\iff \text{started}_A^i \wedge \text{ok}_A^{i+1} \end{aligned}$$

for specifying that A is *ok* (i.e., not violated) in the next step if and only if it has not been started or its invariant (e.g. `VehicleA`'s position relative to the lower western lane) has constantly been satisfied since then.

Finally, the formula is unrolled up to a user-supplied depth n (i.e. iterating over $i \in \{0, \dots, n-1\}$) and the problem is handed to the SMT solver. For $i = 0$, further constraints are added that define valid initial assignments for the variables: Because invariant nodes need to be fulfilled for at least one step, we add $\neg \text{complete}_A^0$ for every invariant; also, $\text{started}_B^0 \implies \text{complete}_A^0$ is added for every sequence.⁴ The full set of constraints can be found in subsection Appendix B.1. Note that by fixing n , all sampled scenarios will have exactly the same length. As to generate models for varying durations, we can simply unroll for different values of n , e.g. $n \in \{10, 15, 20, 25\}$. The underlying assumption is that scenarios useful in testing are neither exceedingly short nor long and that the duration can be drawn from a small, manageable set of durations.

The final task is to produce fine-grained trajectories, as the duration of each time slice is rather long (e.g. one second). For this, the SMT formula encodes the position of dynamic objects as control points of Bézier splines. These points are then used to safely generate smooth trajectories within each time slice, which are piecewise connected for the overall trajectories of the dynamic objects. More precisely, using Bézier splines of degree m , the position of an actor at time t in the scenario is given by the Bézier interpolation

$$\mathbf{p}(t) = \sum_{k=0}^m \mathbf{p}_k^i \mathcal{B}_k^n \left(\frac{t - t_{i-1}}{t_i - t_{i-1}} \right)$$

with Bernstein basis $\mathcal{B}_k^n(s) = \binom{n}{k} s^{n-k} (1-s)^k$. Here, $[t_{i-1}, t_i] \ni t$ is the current time slice and \mathbf{p}_k^i the control points for the corresponding spline segment. Hereby, the convex hull property⁵ of Bézier splines ensures that the invariant nodes are satisfied at any point in time [42] (which allows large step sizes ≥ 1 s without loss of precision). Additional constraints incorporated in step 2 into the SMT formula ensure that the trajectories are physically possible by maintaining curvature and acceleration limits of the vehicles. The detailed construction is presented by Becker [42]. In particular, continuous speed is guaranteed, and the followed path (i.e., arc-length parametrization of the trajectory) is at least continuous differentiable. For the experiments in this work, quadratic Bézier splines ($m = 2$) are used. They guarantee the aforementioned properties, but may result in abrupt acceleration and steering. Using splines of an higher degree (e.g., $m = 3$ or $m = 4$) allows to make acceleration and steering continuous, but on the other hand increases the size of the SMT problems to be solved.

³ The complete formula is a conjunction of 72 linear constraints.

⁴ In conjunction with $\neg \text{complete}_A^0$, the latter may obviously be simplified to $\neg \text{started}_B^0$ for every invariant B that does not start a sequence. In order to keep the translation procedure simple, such cheap simplifications are left to the SMT solver.

⁵ A Bézier curve is always contained in the convex polygon spanned by its control points.

4.5. Variation methods for test suite generation

The process described in subsection 4.3.2 used the method $\text{Solve}(F)$ to generate concrete scenarios from an SMT formula F that can be transformed to the standardized formats for further execution and simulation. We now describe $\text{Solve}(F)$. In order to generate a diverse test suite, we require multiple different concrete scenarios sampled from the same TSC. In case of testing, we also have to generate them with a high variety (as to increase coverage of the executed tests). In this section, we describe three different variation methods for SMT solving that can find diverse models for the SMT problem as introduced in subsection 4.4, namely *Solver Seed Variation (SSV)*, *Recursive Blocking (RB)* and *Recursive Blocking of Invariants (RBI)*.

4.5.1. Solver seed variation (SSV)

To find a model \mathcal{M} for a formula F we can use any SMT solver, for example, Z3 [43]. Many SMT solvers offer the possibility to set seeds, which effectively influences nondeterministic choices in the solver and may create different scenario models. This simple variation method has been already proposed and compared to other methods by Kalwa [44]. Note that variation of Z3's solver seed is not expected to produce a lot (if any) variety among the solutions. Therefore, this method will serve as baseline to compare the other methods with.

4.5.2. Variation by recursive blocking (RB)

Although SSV may create different solutions, it is likely that most solutions are identical, as the solver seed influences only a small part of the solving process. In order to facilitate diverse solutions systematically, we ensure previous solutions to be blocked by adapting the SMT formula.

Definition 8. Let \mathcal{M} be a model of F . A *blocking clause* for \mathcal{M} is any formula $\neg G$ with $\mathcal{M} \models G$. Hence, any model \mathcal{M}' of $F \wedge \neg G$ is a model of F that is not a model of G and different to \mathcal{M} . We say that F is *blocked by* $\neg G$. A *blocking method* is any systematic to obtain a blocking clause $\neg G$ from a model \mathcal{M} .

In case that a blocking clause method is used to create n models, new terms are introduced to the overall formula to solve. If n models for a formula F with m variables shall be found via a blocking clause method, the blocking of each model uses m formulae to restrict all m variables. This results in a total of $n \cdot m$ additional atoms, vastly increasing the complexity of solving the formula with each model found.

To avoid the creation of new atoms, we introduce the atom blocking method:

Definition 9 (Atom Blocking). Based on the observation that $F(L, B)$ may change its truth value only if at least one of its atoms changes its truth value, the atom blocking clause $\neg G$ for $\mathcal{M} \models F$ is defined as

$$G := \bigwedge_{a \in L \cup B} a^{\mathcal{M}} \text{ with } a^{\mathcal{M}} := \begin{cases} a & \text{if } \mathcal{M} \models a, \\ \neg a & \text{if } \mathcal{M} \not\models a. \end{cases}$$

Note that F and $\neg G$ are formulas over the same set of atoms. So the complexity is of finding a model of the blocked formula $F \wedge \neg G$ is (mostly) shifted to the Boolean fragment of the theory.

Instead on defining a completely new region to block, the atom blocking method uses the existing set of atoms. More precisely, to block a model \mathcal{M} it searches for the truth value of all atoms to describe the region where \mathcal{M} is in. Atom blocking has the advantage that blocking of a complete region described by atoms ensures that a new solution has to differ in at least one atom. In case of TSCs, the atoms are linear constraints built on variables of traffic objects, like a restriction on vehicle's distance or relative positions, or they are boolean variables with a strong dependency to traffic object's variables. Hence, if two models differ in at least one atom, the solver is forced to find a solution that differs in a linear constraint. The basic idea is that blocking atoms leads to increases the variation among generated scenario instances.

Furthermore, by continuously blocking new regions of the formula F , we can, in theory, obtain a complete coverage by iterating until the entire solution set is blocked. The Z3 solver provides two methods that continuously introduce new blocking regions: *iterative blocking* (Algorithm 1) and *recursive blocking* (Algorithm 2) [43]. Each method returns a list of pairs $(\mathcal{M}_i, G_i), i = 1, \dots, N$, with $\mathcal{M}_i \models F$, $\mathcal{M}_i \models G_i$ and all G_i pairwise disjoint. Here, N is the number of blocking clauses found by the method, which is naturally bounded by $N \leq 2^{|L \cup B|}$, the number of different blocking clauses that can be constructed from the atoms $L \cup B$ in F . The disjunction of all G_i is a complete cover of F , i.e. $F \implies \bigvee_{i=1, \dots, N} G_i$.

The differences between both methods are as follows: Similar to the ε -blocking, the iterative blocking in Algorithm 1 increases the formula size monotonically, while in Algorithm 2 the formula size of F is bounded, as at most k literals for the initial terms a_1, \dots, a_k are added. Furthermore, Algorithm 1 can be used with any blocking method, Algorithm 2 is tailored towards constraints and atoms. This means that the blocking takes place in the subspace of the initial terms. The method yields a partition of F , i.e. $F \leftrightarrow \bigvee_{i=1, \dots, N} G_i$ with pairwise disjoint G_i in this subspace.

The blocking methods are based on the principle of changing the truth value of atoms to get new models and respective blocking regions. In theory, the atom blocking methods could block more and more regions until the overall solution space of a formula is discovered and all regions are found. For the generation of a test suite, it is advantageous to explore all possible regions. Then, test scenarios can be selected with a sufficiently large distribution and variation, for example by selecting scenarios equally distributed

Algorithm 1 Iterative Atom Blocking.

```

procedure BLOCKITERATIVE( $F$ )
  while  $F$  is satisfiable do
     $\mathcal{M} \leftarrow$  model of  $F$ 
     $\neg G \leftarrow$  blocking clause for  $\mathcal{M}$ 
    yield  $(\mathcal{M}, G)$ 
     $F \leftarrow F \wedge \neg G$ 
  end while
end procedure

```

Algorithm 2 Recursive Atom Blocking.

```

procedure BLOCKRECURSIVE( $F, \{a_1, \dots, a_k\}$ )  $\triangleright \{a_1, \dots, a_k\}$  is a set of atoms
  if  $F$  is satisfiable then
     $\mathcal{M} \leftarrow$  model of  $F$ 
    for  $i \leftarrow 1 \dots k$  do
       $\neg G_i \leftarrow \neg a_i^{\mathcal{M}} \wedge \bigwedge_{j=1, \dots, i-1} a_j^{\mathcal{M}}$   $\triangleright$  blocking clause for  $\mathcal{M}$ 
      yield from BLOCKRECURSIVE( $F \wedge \neg G_i, \{a_{i+1}, \dots, a_k\}$ )
    end for
  yield  $(\mathcal{M}, \bigwedge_{i=1, \dots, k} a_i^{\mathcal{M}})$   $\triangleright \bigwedge_{i=1, \dots, k} a_i^{\mathcal{M}} = \neg \left( \bigvee_{i=1, \dots, k} \neg G_i \right)$ 
  end if
end procedure

```

from all regions. However, the experimental results presented in subsection 6.1 indicate that this is practically infeasible (due to the large size of the solution space) and it is not advisable to discover the entire solution space for achieving a high variation. We have to deal with a nearly uninformed exploration of the solution space and try to create as much variation between the identified models as possible. Therefore, we focus on (a) methods to guide the search to a high variation, e.g. recursive blocking of invariants (RBI) (presented in subsection 4.5.3), and (b) measuring the already achieved variation (presented in subsection 4.6).

4.5.3. Variation by recursive blocking of invariants

The SMT formulae from subsection 4.4 include Boolean variables modeling the temporal satisfaction of each of the TSCs invariant nodes. More precisely, we introduced the variable inv_A^i for each invariant A and time step i , where inv_A^i is true if and only if the invariant A is valid at time step i .

When using RB, it can be advantageous to search for blocking regions within a (small) subset of all atoms. This is realized by the method *Recursive Blocking of Invariants* (RBI) that uses the Boolean variables inv_A^i as initial terms a_1, \dots, a_k . The idea is that blocking those variables will force the solver to perform a time-shift of the invariants' satisfaction. As many influential variables in traffic scenarios are time dependent, e.g. positions and velocities, this shift is hypothesized to result in vastly different scenarios. As an example, consider the invariant node in the middle of Fig. 2, specifying an occlusion scene. If this node is valid at different time steps, the bicyclist's occlusion will happen at different times in the scenario as well.

4.6. Measuring test suite quality

As we now have several methods to generate concrete scenarios, i.e. to generate instances (or models) from a TSC-derived SMT formula, it remains to assess the quality of such sets. This enables 1) a comparison of such variation methods and 2) the formulation of exit conditions for the sampling process at an appropriate point in time. Naturally, we neither want too few concrete scenarios that do not generalize well to the abstract test suite, nor do we want to generate too many unnecessary samples that do not add value, but increase costs in testing.

For this, we first introduce a formal measure of distance between two concrete scenarios in subsection 4.6.1. In order to properly generalize this notion of distance to test suites containing $2 \leq n \in \mathbb{N}$, we elicit desirable properties in subsection 4.6.2. In subsection 4.6.3, we present a suitable candidate for aggregating such distances based on an distance-based estimate of the differential entropy. Ideas on quality functions as exit conditions are sketched in subsection 4.6.4.

4.6.1. Measures for scenario distances

In its most basic form, scenario distance measures rely on actor positions over time (however, there are certainly other elements to focus on in a scenario), which corresponds to restricting the scenario distance to layer 4 of the 6-layer model [45]. Such approaches to measure distance and similarity between concrete scenarios have been studied in-depth by Braun et al. [46]. Among others, *Dynamic Time Warping* (DTW) is presented as one candidate when given trajectory data (as it is the case in this work).

DTW has been proposed in the 1970 s [47] and takes as input two time series, i.e. positions of two actors. Essentially, DTW measures the pairwise Euclidean distance between all points of the given two time series. It then uses a greedy algorithm to find a matching with minimal distance between the points of the two time series. The sum of the distances in this matching is the DTW result. For multidimensional time series the DTW can be estimated either in a dependent or independent way [48]. As the relative positions of the different actors is typically relevant for the comparison of two concrete scenarios, we further apply the dependent version of the DTW.

In the following, we assume that in the scenario belonging to the abstract test case there exist actors A_1, \dots, A_q with $q \geq 1$ and trajectories tr_1, \dots, tr_q where $tr_k^{d,t}$ is A_k 's position in the d -th dimension at time t . Formally, DTW of two models of an abstract scenario with two-dimensional trajectories and d and e time steps can then be defined as

$$\text{DTW}(\mathcal{M}, \tilde{\mathcal{M}}) = \min_{\pi \in M(\mathcal{M}, \tilde{\mathcal{M}})} \sqrt{\sum_{(i,j) \in \pi} \sum_{k=1}^q (tr_k^{1,i} - \tilde{tr}_k^{1,j})^2 + (tr_k^{2,i} - \tilde{tr}_k^{2,j})^2},$$

where $M(\mathcal{M}, \tilde{\mathcal{M}})$ is the set of all possible sequential matchings between the time steps of the models \mathcal{M} and $\tilde{\mathcal{M}}$, i.e. $M(\mathcal{M}, \tilde{\mathcal{M}}) = \{((1,1), (r_2, s_2), \dots, (r_{m-1}, s_{n-1}), (d, e)) \mid r_i \in \{1, \dots, d\} \text{ and } s_j \in \{1, \dots, e\} \text{ and } r_{i-1} \leq r_i \leq r_{i-1} + 1 \text{ and } s_{j-1} \leq s_j \leq s_{j-1} + 1 \text{ for all } i \in \{1, \dots, m\}, j \in \{1, \dots, n\}\}$ [49]. Note that the matchings in M do not need to be of equal length.

DTW has several features making it particularly applicable to the task at hand:

1. It considers temporal information s.t. patterns occurring in both models at different times can be identified as similar,
2. It does not require both models to have an equal number of time steps.
3. It is bounded from below by zero. Further, an upper bound is given by the Euclidean distance. Since every instance of an abstract scenario is of finite length, there are bounds for the trajectories of the different actors that hold for every concrete instance of an abstract scenario. Consequently, an upper bound exists for $\text{DTW}(\mathcal{M}, \tilde{\mathcal{M}})$ that holds for all instances $\mathcal{M}, \tilde{\mathcal{M}}$ of an abstract scenario.
4. Its output is readily interpretable by a test engineer as it is based on a matching of Euclidean distances.

However, the incorporation of temporal aspects is limited; in fact, DTW tries to stretch or shorten time within the trajectories to find the smallest sum of pairwise Euclidean distances. In some special cases, this leads to DTW not discerning two different models (e.g. if the trajectories are shifted in time and padded appropriately with the same points at start and end).

There exist several other suitable distance measures [46], such as the Fréchet distance. Since our experiments indicated comparable behavior to DTW, we chose to focus solely on DTW in what follows. For the Fréchet distance, such behavior is expected as it 'can be seen as a version of DTW that takes the maximum distance between aligned points along the path' [50].

4.6.2. Desirable properties of distance-based quality functions

We have now established a formal notion of distance between two concrete scenarios in a test suite. Recall that we are generally interested in not only determining the distance of two concrete scenarios but measuring the quality of a set thereof. In our case, quality is understood as a proxy for predicted efficiency during downstream testing. Intuitively, a test suite is efficient if the test cases are diverse in the sense that from each test case new information can be gained in testing. A quality function shall characterize different properties of the overall test suite and combine them into a scalar value, i.e. we are looking for a map $Q : S \rightarrow \mathbb{R}$ with certain desirable properties. This enables the comparison of test suites and, therefore, also the comparison of different variation methods. Before we discuss a potential candidate for Q , we define a set of properties, which we hold valuable to characterize the test suite quality.

- #1 **Avoid redundancy:** Adding an instance to the test suite that is equal to an already sampled instance leads to a decrease in quality.
- #2 **Existence of quality decreasing neighborhoods:** There exist neighborhoods around the instances in a test suite, so that adding an instance that is inside of such a neighborhood leads to a decrease in quality. This property extends the *Avoid redundancy* property.
- #3 **High distances increase quality:** If there exists an instance whose minimal distance to the instances in the test suite is higher than the maximal distance between those instances, adding this instance leads to an increase in quality.
- #4 **Preference of higher distance instances:** Adding an instance to the test suite that is further away to all already sampled instances than another possible instance leads to a stronger increase in quality.
- #5 **Preference of less sampled regions:** Having clusters of instances in the test suite, adding an instance to the test suite that is equally far away to a cluster than another possible instance to a denser cluster leads to a higher quality.
- #6 **Boundedness:** There exist lower and upper bounds for the test suite quality, enabling a better estimation of the actual test suite quality.
- #7 **Invariance under permutation:** The order in which instances were added to the test suite has no influence on the quality.

We refrain from a formal notion of these properties here, and leave this aspect open for future work.

4.6.3. Distance-based quality of test suites

The quality of a test suite should correlate with its efficiency in terms of information gain during testing. In the context of information theory, entropy has been established as tool for evaluating the diversity of information. Therefore, to evaluate the quality of a test suite containing instances $\mathcal{M}_1, \dots, \mathcal{M}_n$ we propose a function Q that leverages a distance-based estimate of the differential entropy [51]. Notably, our proposed notion is slightly modified to allow for the evaluation of redundant instances:

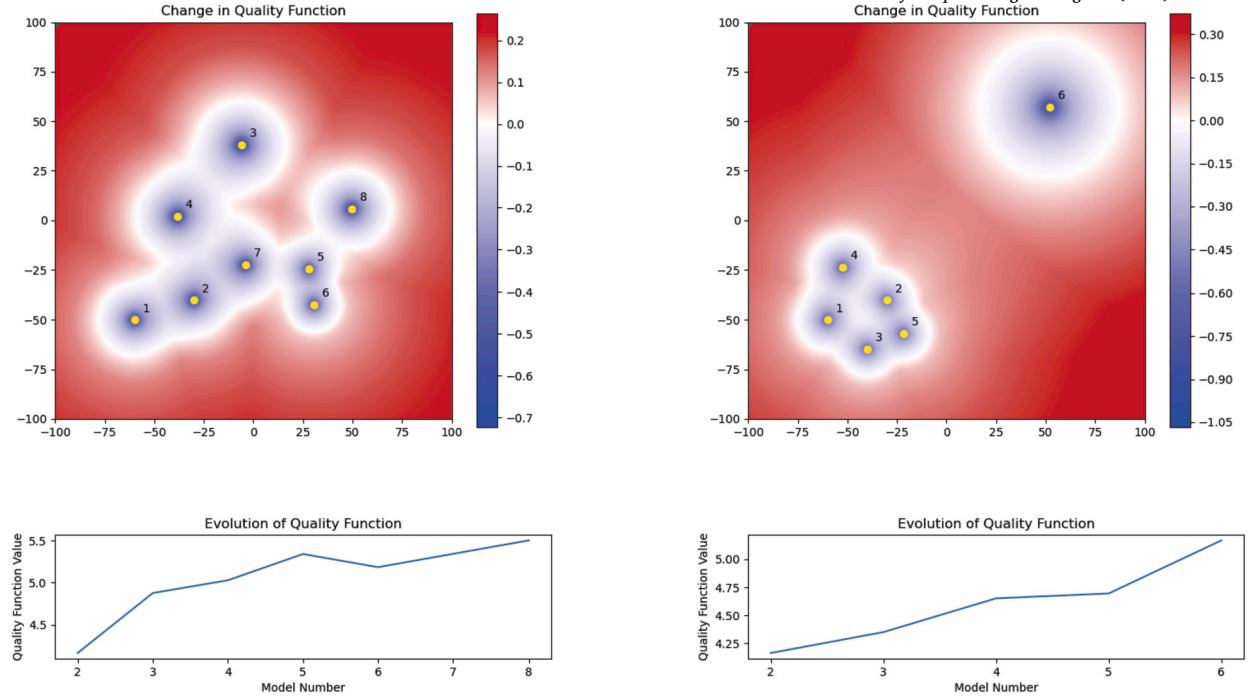


Fig. 7. Visualization of the behavior of the quality function Q in a two-dimensional Euclidean space. The upper plots illustrate how adding a new point would change the quality function. The lower plots depict the evolution of the quality function for a sequential addition of points from upper plot.

$$Q(\mathcal{M}_1, \dots, \mathcal{M}_n) = \frac{1}{n} \sum_{i=1}^n \ln(1 + n \cdot \min_{\substack{j=1, \dots, n \\ j \neq i}} DTW(\mathcal{M}_i, \mathcal{M}_j)) \quad (5)$$

The function fulfills several of the properties introduced in subsection 4.6.2:

- #1 **Avoid redundancy:** Based on heuristic experiments in the two-dimensional Euclidean space we conclude that adding a redundant instance to a test suite is penalized by a decrease in quality, cf. Fig. 7.
- #2 **Existence of quality decreasing neighborhoods:** This property is an extension of the *Avoid redundancy* property. Similar to #1, our heuristic experiments indicate its fulfillment for the Q of equation (5).
- #3 **High distances increase quality:** Adding an instance to the test suite, whose distance to each of the already sampled instances is higher than the maximal distance between those instances, does not affect the minimal distances of the initial test cases. Based on this it can be easily proven that this property holds.
- #4 **Preference of higher distance instances:** As the logarithm is a strictly increasing function, this property is fulfilled.
- #5 **Preference of less sampled regions:** This property is not fulfilled. A counterexample is illustrated on the upper right side of Fig. 7.
- #6 **Boundedness:** Let d_{max} describe the upper bound of the DTW in a given abstract test case, cf. subsection 4.6.1. Then, Q is bounded by

$$0 \leq Q(\mathcal{M}_1, \dots, \mathcal{M}_n) \leq \ln(1 + n \cdot d_{max}).$$

- #7 **Invariance under permutation:** The invariance under permutation is given by construction of Q .

We remark, that formal proofs for these properties as well as an extensive evaluation of other quality functions are left for future work.

4.6.4. Quality functions as exit conditions

Based on our quality function Q , we can now define our final component, namely an exit condition for the sampling process. Note that $Q \rightarrow \infty$ for $n \rightarrow \infty$. For fixed n , however, Q is bounded from above, as d_{max} is finite because the distances between actors are naturally bounded by the distance they can travel within the scenario. Moreover, d_{max} can be computed from the input TSC, e.g. using the Euclidean distance. Therefore, an exit condition of the form

$$\frac{Q(\mathcal{M}_1, \dots, \mathcal{M}_n)}{\ln(1 + n \cdot d_{max})} > \alpha \quad \text{with } \alpha \in (0, 1), \quad (6)$$

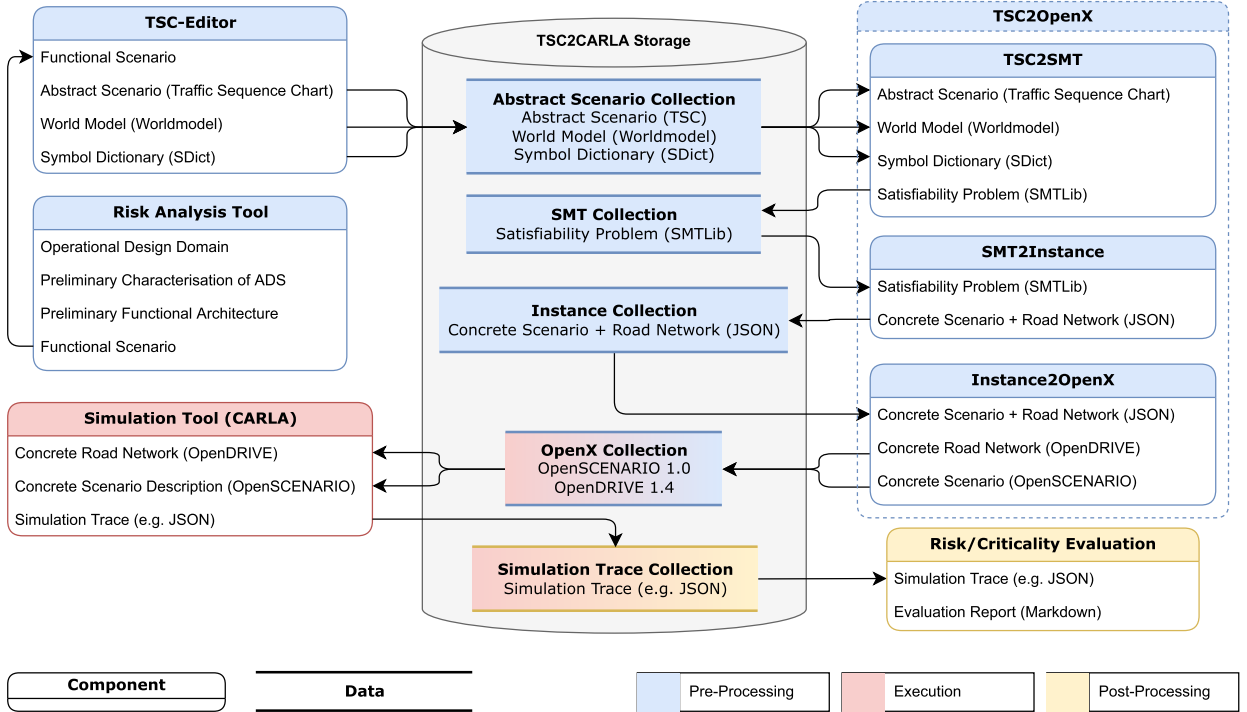


Fig. 8. Components and data flow within the TSC2CARLA toolchain.

which is evaluated for every $n \geq 2$, is feasible. As the upper bound for Q is not particularly sharp, experimental results indicate that values of $\alpha = 0.5$ already point to high quality test suites. However, this may lead to extremely small test suites, e.g. when the distances of the first few samples are large enough to fulfill such a criterion. Obviously, having only a couple of scenarios in a test suite can be undesirable. Thus, we propose to combine the condition of equation (6) with a fixed minimum number n_{\min} of test cases to generate. This means that equation (6) is only evaluated in case $n \geq n_{\min}$. This minimum number can be chosen based on the test modalities later on. For example, in case of SIL-simulations, this may be 10^7 , whereas for proving grounds, a realistic minimum may be 10^2 .

4.7. A note on performance

We conclude the theoretical underpinnings of TSC2CARLA with a discussion of its performance. Note that we have two computationally expensive parts in our method: Firstly, the SMT solving and secondly, the computation of the quality metric.

In general, the overall complexity is dominated by the computational properties of the employed SMT theory. In our case of linear real arithmetic, the often employed simplex algorithm runs in exponential worst-case time [52]. Note that the size of the initial SMT formula is linear in the number of basic charts p in the TSC, the number of actors q in the scenario, and the unrolling depth t . More precisely, it is in $\mathcal{O}(t \cdot p \cdot q)$. For recursive blocking, the size of the SMT formula increases by the size of the blocking clause. In the blocking clause, each atom can occur at most once, so it never grows larger than the initial SMT formula.

Despite the exponential worst-case complexity, optimizations made SMT solving tractable in practice. It is thus of interest to also examine the computational complexity of computing the exit condition, which is repeated for each scenario. Assuming we have q actors in a given TSC and t time steps in the SMT solutions, the worst case complexity of adding the $(n + 1)$ -th scenario is $\mathcal{O}(n \cdot q \cdot t^2)$, for computing the distances to the n former scenarios [49], i.e. evaluating n times the multi-dimensional, dependent DTW for two matrices of size $2 \cdot q \cdot t$.

Overall, for the generation of n scenarios, computing Q according to equation (5) adds a worst case complexity of $\mathcal{O}(n^2 \cdot q \cdot t^2)$.

5. TSC2CARLA: toolchain architecture

To instantiate the previously introduced theoretical framework, we now present the TSC2CARLA toolchain architecture and its prototypical implementation. The toolchain architecture is summarized by Fig. 8, concretizing the abstract artifacts of the underlying framework as presented in Fig. 4. We have three major components: First, the specification of abstract test cases is supported by the TSC-editor. Second, the TSC2OpenX component realizes the test suite generation. Third, for execution, we rely on a suitable simulation tool such as the CARLA Scenario Runner. The technical implementation of the different components of the TSC2CARLA toolchain is described below.

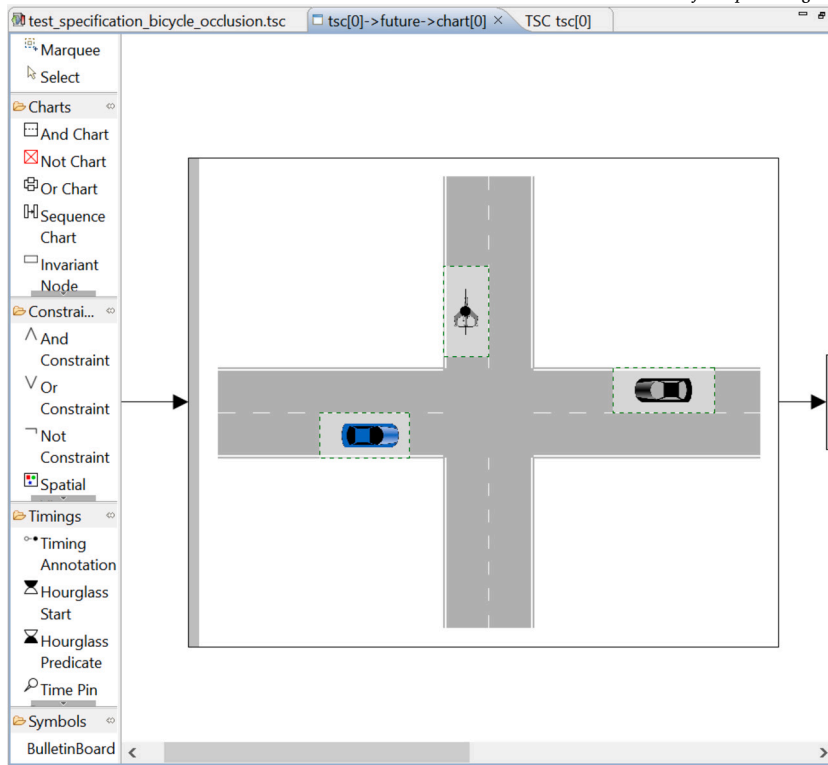


Fig. 9. Screenshot of an invariant node in the TSC-editor.

5.1. Toolchain user interface

At the current stage, the toolchain is accessible by a rather rudimentary user interface consisting of two configuration files stored in YAML format that are passed as command line arguments to the toolchain execution script. The usability improvement is mainly due to the fact that information relevant for multiple tools is stored in a central configuration file and distributed to each subcomponent upon execution of the toolchain. Further, the toolchain orchestrates the execution of the individual tools while ensuring their interchangeability and the consistent usage of configuration parameters. For the future, an improved support for external development and verification tools is planned, however, for the current state of a prototypical implementation these aspects have not been prioritized. Since the toolchain is in its early development stages a graphical user interface is not yet available, but planned for the future.

5.2. TSC-editor for the specification of abstract scenarios

The TSC-editor provides a model-centric workbench for the formal specification of TSCs. It has been designed with the spirit of TSCs in mind: Assisting users in writing down well-defined scenarios while not being bound to a pre-defined domain ontology. In the first place, the TSC-editor provides, as its name suggests, a graphical editor for TSCs, as shown in Fig. 9. Before creating a TSC, the user has to define a world model and a symbol dictionary. For this task, the workbench also provides editing tools. Here, the user defines object and symbol types, including the attributes and anchors to be referred to from the TSCs. The user-defined symbols are then added to the tool palettes of the TSC-editor. This ensures that the graphical syntax of TSCs is always correct and the semantics well-defined w.r.t. a symbol dictionary and a world model. Furthermore, the workbench provides analysis techniques like syntax checking of textual annotations in the TSC as well as a consistency analysis that helps to find specification errors. This architecture is visualized in Fig. 10.

The TSC-editor stores TSCs, world models and symbol dictionaries in an XML format. The editor has been implemented with eclipse technology such as EMF and GEF, which enabled rapid prototyping of the workbench as well as straightforward provision of a model-based API as a basis for the supporting tooling. This API is also used by the trajectory generation.

5.3. TSC2OpenX

The TSC2OpenX tool is an integral component of the TSC2CARLA toolchain and is responsible for the generation of the test suite. As can be seen from Fig. 8, TSC2OpenX consists of the three submodules TSC2SMT, SMT2Instance and Instance2OpenX. Based on a TSC specified with the TSC-editor, an SMT formula describing the constraint system is generated, then solved with the SMT solver Z3 [53] and the solution is mapped to a combination of an OpenDRIVE and an OpenSCENARIO file [13,44]. Initially, TSC2OpenX has

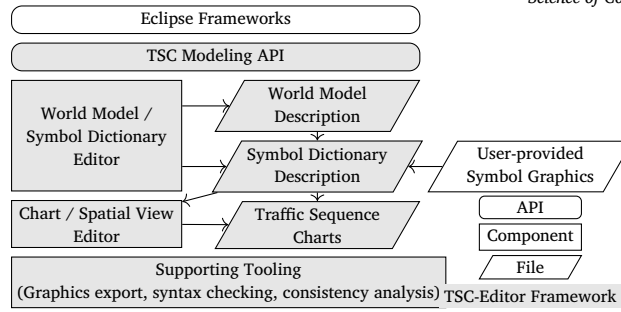


Fig. 10. Architecture of the TSC-editor.

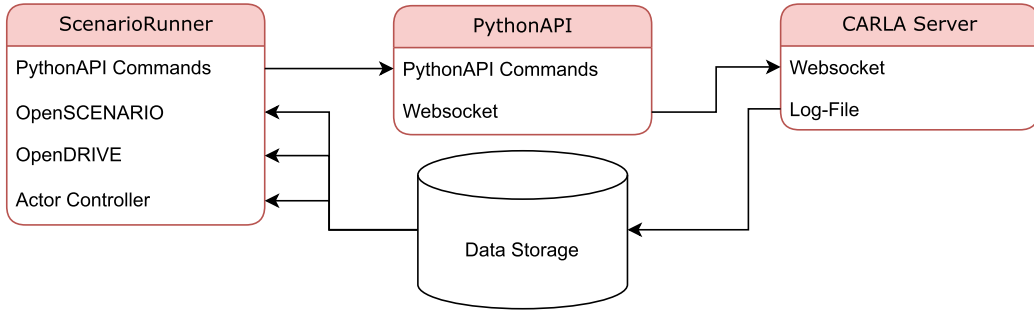


Fig. 11. Detailed view on the communication of ScenarioRunner and CARLA.

been developed for and evaluated on highway scenarios. In this work, our running example features an urban intersection scenario, cf. Fig. 2, thereby extending the toolchain to urban traffic scenarios.

5.3.1. TSC2SMT

The `TSC2SMT` component realizes the translation of a TSC into a satisfiability problem as introduced in subsection 4.4. Therefore, the component takes a TSC specification as input and gives a formula in the SMTlib format as output. The user has to either provide a configuration with a fixed number of time steps for SMT generation and a fixed time increment Δt , or set an upper and lower bound for the number of steps (then, the tool finds the minimum depth that has a solution). The component then generates an SMT formula according to the bounded model checking problem introduced in subsection 4.4. In a first step, a formula with the minimal number of time steps and step size Δt is created. Then the component incrementally adds a new time steps until a solution is found.

5.3.2. SMT2Instance

The `SMT2Instance` component implements the variation methods described in subsection 4.5. As input, the component can be configured with the number of models to search for as well as one of the searching methods SSV, RB and RBI. As output, the component produces a list of JSON files for each model instance that has been found. A model is an assignment of values to the variables in the SMT problem. In our case, there are static variables like the position and size of roads and lanes as well as dynamic variables like the evolution of invariant satisfaction or vehicles' position over time. The static variables are assigned directly, the dynamic variables are assigned as list with the values for each time step. Furthermore, `SMT2Instance` implements the pairwise scenario distances based on DTW as introduced in subsection 4.6.

5.3.3. Instance2OpenX

The component `Instance2OpenX` generates OpenSCENARIO and OpenDRIVE files (referred to as 'OpenX') based on the trajectories obtained from the previous `SMT2Instance` component. Furthermore, it takes the environment related variables, like road length and lane width, to create an OpenDRIVE file. For the generation, `Instance2OpenX` uses a configuration to map the traffic objects in the TSCs to corresponding types in the OpenX files.

5.4. ScenarioRunner and CARLA

We rely on the CARLA ScenarioRunner⁶ to run the OpenX files of the generated test suite in CARLA. There are multiple ways to specify a scenario in the OpenSCENARIO format and one of them employs the specification of trajectories using FollowTrajectoryActions, i.e. specifying points that shall be reached over time. Since the solutions of the SMT problem are encoded as trajectories

⁶ https://github.com/carla-simulator/scenario_runner.

and not as a sequence of maneuvers, these FollowTrajectoryActions were chosen for the encoding of trajectories in OpenSCENARIO. However, there are even several ways to specify these trajectories, for example expressing how each points shall be reached from the one before, e.g. using polylines or non-uniform rational B-splines. The CARLA ScenarioRunner for CARLA 0.9.13 did not support either of them, but using the help of a public pull request we were able to get the polyline variant running, which is sufficient due to the usage of small time step sizes and the splines that were used to generate the trajectories in the first place. Further, spawning a construction site in the simulation for one of the evaluation scenarios was not straightforward and we had to rely on a slight modification of the ScenarioRunner to achieve this. In the instantiation phase of the simulation, the model corresponding to the SuT is inserted as controller for the ego vehicle. Before starting the simulation, the ego trajectory in the OpenSCENARIO file is modified such that just the starting point remains. Additionally, the last point of the ego trajectory is injected into the controller as its target location.

The general data flow of the connection of ScenarioRunner and CARLA is depicted in Fig. 11. The scenario execution consists of iterative executions of the CARLA ScenarioRunner that converts instructions as defined in the OpenSCENARIO XML file into commands that may be transmitted to the CARLA server using the PythonAPI. Thereby, it also parses the controller section of the OpenSCENARIO file and employs specified controller definitions into the scenario execution. Even though OpenSCENARIO is considered a standard, the specification of controllers is left to the simulator itself. In the case of CARLA, this is realized with the possibility to reference Python files defining a `run_step()` method that is called in every simulation step. Two different controller implementations are utilized in the TSC2CARLA toolchain: an SuT controller employs the response of an implemented ADS model, which will be introduced in subsection 6.2. All other vehicles follow their predefined trajectories described in the OpenSCENARIO XML file. ScenarioRunner already provides this controller functionality with its `NpcVehicleControl`. As for the underlying static road network, the OpenDRIVE file is read by ScenarioRunner and sent to the CARLA server before the scenario is then executed. Once all available scenarios are simulated and the log files are written to the data storage, the requirement evaluation is started.

5.5. Requirement evaluation

During the execution of the CARLA simulation, data is collected for the subsequent evaluation of the SuT performance. For this, the user has to specify a set of requirements in form of criticality metrics with corresponding target values [39]. A test case counts as passed if there is no metric violating its target value and failed otherwise. The current implementation is based on the `MetricsLog` of ScenarioRunner (for logging and data storage) and implements custom criticality metrics on top. Currently, we support:

- Time To Collision
- Required Longitudinal Acceleration
- Brake Threat Number
- Post Encroachment Time
- (Scaled and Duration-dependent) Predictive Encroachment Time
- Predictive Conflict Index (with Duration-dependent Predictive Encroachment Time)

Except for the Post Encroachment Time, all metrics are based on a constant-velocity, constant-heading prediction model that uses a two-dimensional bounding box representation for the actors.

Once the sampling and simulation process is finished, the given metrics are evaluated for a set of actor-pairs specified by the user. The resulting two-dimensional (actor-pairs and time) outcomes are then aggregated for both dimensions (e.g. by using min, max or sum) and compared to user-supplied target values. A tabular report file is generated for the user, where overall and metric-specific pass/fail verdicts are reported for each test case. Finally, our toolchain reports ‘pass’ if and only if the verdicts for all metrics and test cases resulted in ‘pass’.

6. Evaluation

We now evaluate the presented theoretical approach of section 4 together with a prototypical implementation the toolchain described in section 5. Since we are not aware of any publicly available implementation for sampling from abstract scenarios based solely on constraints,⁷ effectiveness of our approach can only be evaluated relative to itself. We split our evaluation in two parts. First, we examine the feasibility of our SMT-based model generation approach and on how test suite quality behaves during sampling in four different settings. Second, we demonstrate overall practical applicability of our toolchain by testing a custom ADS model on two abstract test cases. Since we aim to investigate the behavior of our method over the number of generated scenarios, we did not use an exit condition for scenario generation here. To keep the evaluation extent manageable, we decided to generate $n = 2000$ scenarios. All generated SMT formulae, OpenDRIVE and OpenSCENARIO files, and resulting simulation data are available online [58].

6.1. Evaluation of test suite generation

We start by evaluating our toolchain up to the simulation component, i.e. we evaluate the test suite generation with a particular focus on variation methods and test suite quality. For this, we seek to answer the following evaluation questions (EQs):

⁷ Fremont et al. propose the only comparable and available approach [8], however, it requires parallel simulation.

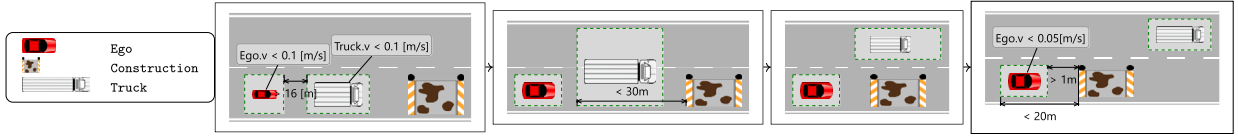


Fig. 12. Ego following a truck which cuts out of lane in front of a construction site, specified as a Traffic Sequence Chart.

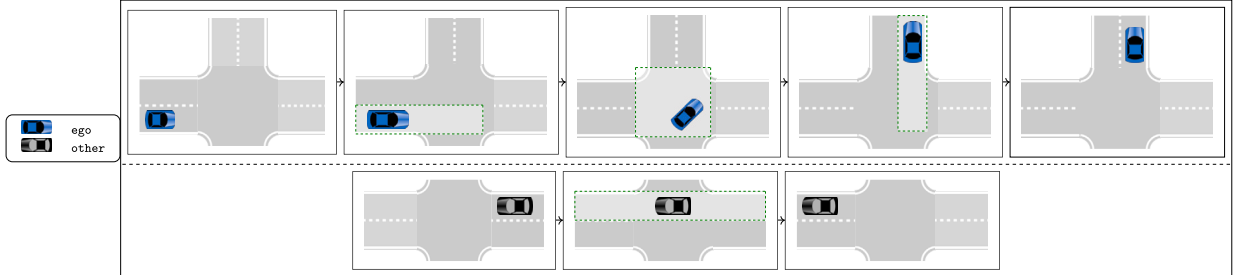


Fig. 13. Ego performing an unprotected left turn at an four-armed intersection with oncoming traffic, specified as a Traffic Sequence Chart using parallel composition.

- EQ1 How effective are the variation methods in generating diverse test suites?
- EQ2 What is the effect of different scenarios durations on test suite quality?
- EQ3 Can the SMT-based sampling process be performed under reasonable resources?

As to provide evidences, we examine three different abstract scenarios

1. our running example, the *bicycle occlusion* from Fig. 2,
2. a *cut-out* scenario where a truck changes the lane just in front of a construction site, cf. Fig. 12, and
3. a *left turn* scenario with oncoming traffic, specified as parallel composition, cf. Fig. 13.

The abstract scenarios were solved with $t = 13$ time steps of step length 1 s. Additionally, the bicycle occlusion scenario was solved with $t = 11$ time steps of length 1s.

All experiments were performed on WSL2 on a Windows 11 host machine with an Intel i9-13900 K CPU and 64 GB RAM.

EQ1 – effectiveness Our goal is to evaluate the effectiveness of the variation methods of subsection 4.5 in generating diverse instances that can be used as test cases. For this, we employ the quality function Q , as introduced in subsection 4.6. Using Q , we compare the output of RB and RBI against the baseline method SSV. In a first step we analyze the overall number of instances as well as the number of redundancies generated by the different variation methods. In a second step, we have a closer look on the evolution of the distances over time. Thirdly, we evaluate the different variation methods according to the Quality function introduced in subsection 4.6.

Recall that we use the multi-dimensional, dependent DTW [48] as a distance between scenarios which receives as input two matrices containing the (x, y) -positions for each actor over a finite set of time steps corresponding to two model instances $\mathcal{M}, \tilde{\mathcal{M}}$. Not all atoms of an SMT formula corresponding to a TSC – when blocked – influence the actors’ trajectories found by the solver. So, it may happen that $DTW(\mathcal{M}, \tilde{\mathcal{M}}) = 0$, even though $\mathcal{M} \neq \tilde{\mathcal{M}}$. Adding to this is the fact, that although DTW has some nice properties, it is not a metric in the mathematical sense [49].

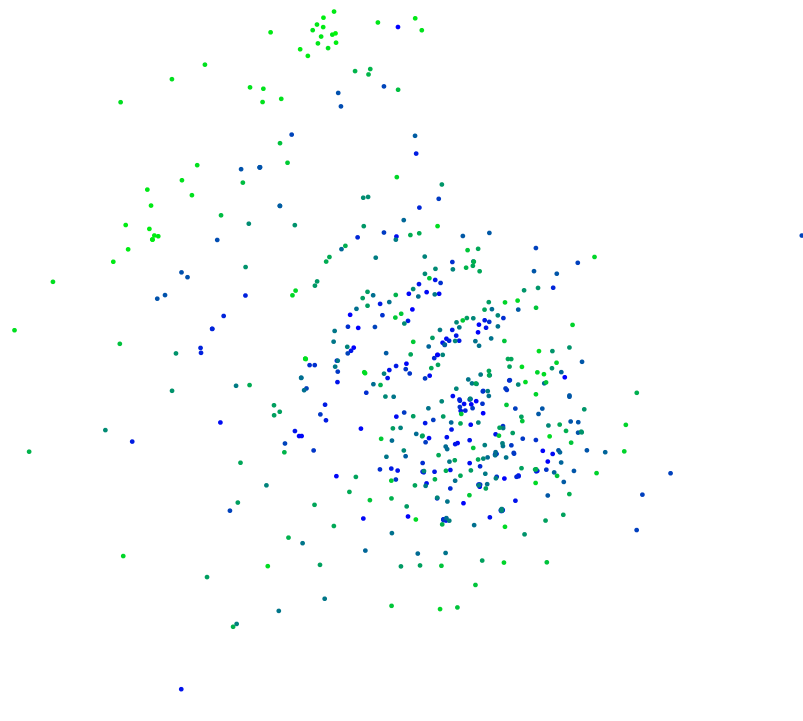
Therefore, the first two interesting questions regarding effectiveness are: i) Can our variation methods find the user-specified number of scenarios? and ii) Do the variation methods consistently produce model instances that have non-zero DTW to instances that were already found? For this, let us consult Table 2. We immediately see that SSV and RB have no problem in finding $n = 2000$ model instances in all four scenarios. However, they produce mostly redundant instances with zero distance. On average, our baseline method SSV produced only 13 (or, $\approx 0.65\%$ of) model instances with pairwise non-zero distances. This really shows that the solver seed in Z3 has almost no impact on solving TSC-derived SMT formulae.

The variation method RB manages to find ≈ 80 ($\approx 4\%$) non-zero distance instances by blocking out former solutions recursively according to Algorithm 2. This indicates that most of the atoms blocked in the procedure have no influence on the actors’ trajectories found by Z3. The variation method RBI – a special case of RB which limits the blocked atoms to Boolean variables – generates significantly less redundant instances. Due to the rather small amount of boolean variables combined with a rather short scenario length, RBI has not been able to produce the user-specified amount of $n = 2000$ scenarios in three out of four cases. However, relatively speaking, of the total number of instances found by RBI, $\approx 71\%$ had pairwise non-zero distances. This answers the previous questions: i) SSV and RB produce 2000 instances in each case, while RBI did so only in one out of four cases; ii) only RBI produced a satisfactory amount of non-zero distance instances.

As an intermediate conclusion, let us remark that the ordering of the atoms to be blocked in Algorithm 2 seems to have a profound impact on the variety of the resulting trajectories. By limiting blocking to Boolean variables (RBI), we already achieve a high degree of

Table 2
Ratio of non-zero distance models by total models found on input $n = 2000$.

Scenario	Variation Method	Non-Zero Distance Models
Bicycle Occlusion ($t = 11$)	SSV	5/2000
	RB	102/2000
	RBI	122/192
Bicycle Occlusion ($t = 13$)	SSV	15/2000
	RB	59/2000
	RBI	517/1024
Cut-Out	SSV	23/2000
	RB	85/2000
	RBI	379/400
Left Turn	SSV	9/2000
	RB	73/2000
	RBI	1536/2000



10

Fig. 14. Plot of all 1024 models sampled from the Traffic Sequence Chart of Fig. 2 with Recursive Blocking of Invariants (cf. lower left of Fig. 15) according to their distance (Dynamic Time Warping), with blue colors found early and green colors found late in the sampling process. As coordinates were chosen by multi-dimensional scaling based solely on distances, axes do not possess relevant information and have thus been omitted. Therefore, only a key with a distance of 10 is provided at the bottom left.

variation. However, this possibly reduces the total number of model instances below the user-specified amount. So, further refinement of the methods RB and RBI regarding which atoms of TSC-derived SMT formulae are suitable for blocking when aiming for effective sampling from abstract test cases is warranted.

Distance of models sampled by RBI over time In order to analyze the promising results of the RBI variation method closer, we examine its sampling behavior over time. Fig. 14 shows a two-dimensional plot of all 1024 instances generated by the RBI method from the bicycle occlusion TSC of Fig. 2 with $t = 13$ time steps, arranged according to their distances (using multi-dimensional scaling [54]). Note that the reduction from originally 66 to two dimensions may lead to information loss and thus, distances in Fig. 14 can be imprecise. Here, we find that RBI is able to identify new models late in the sampling process (upper-left green part) with high distances to the already identified cluster (bottom-right blue part). Moreover, the sampling process shows no directly obvious

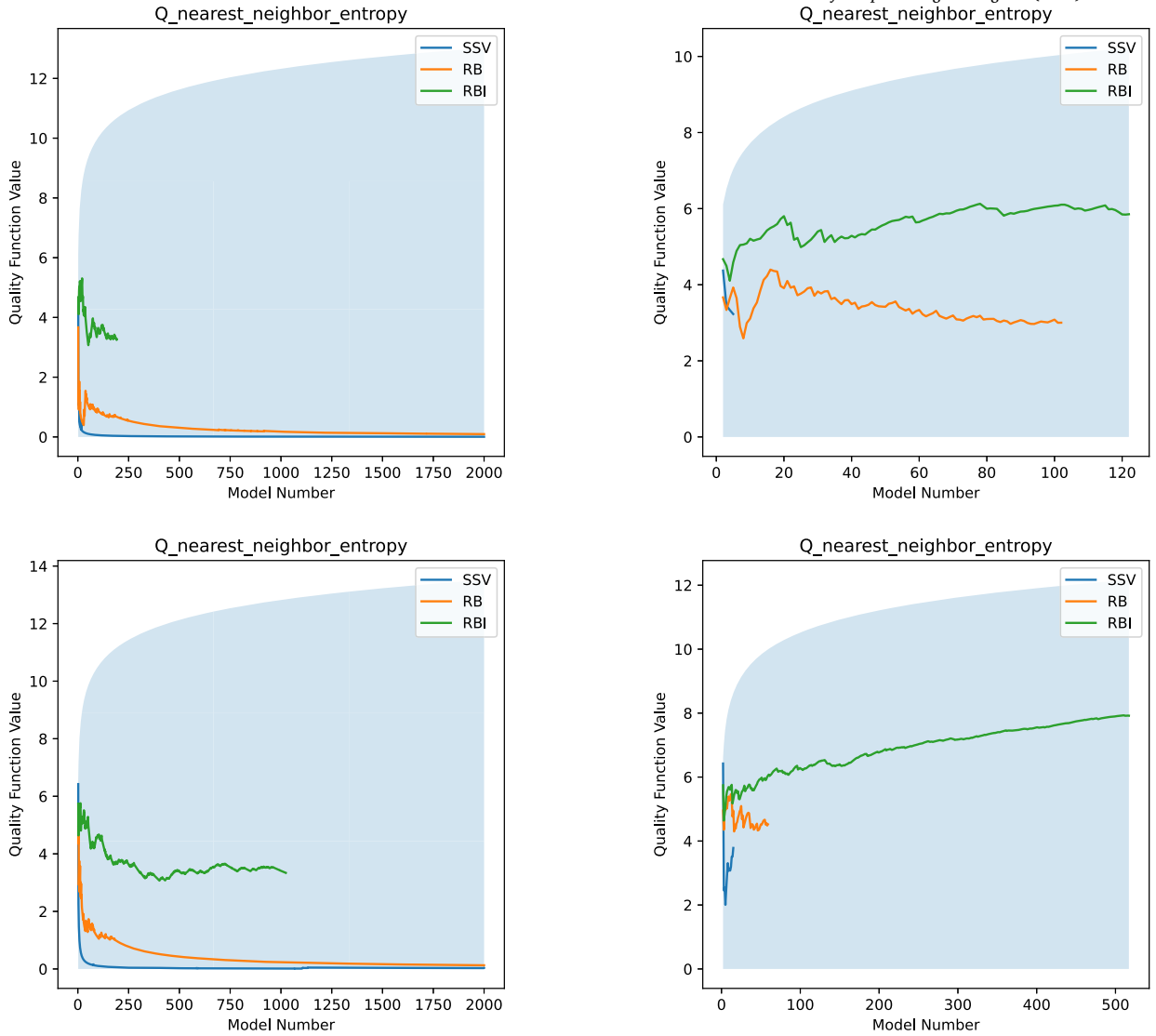


Fig. 15. Evolution of the quality function Q for the bicycle occlusion scenario of Fig. 2 with $t = 11$ time steps (upper row), $t = 13$ time steps (lower row), including zero distances (left column), restricted to non-zero distances (right column). The light blue background visualizes the upper- and lower bounds of Q .

tendency to generate isolated clusters of models in our example. Some outliers, however, exist (e.g. at the right and bottom left) exhibiting large distances to almost all models, making them suitable candidates for further investigation during testing.

Evolution of quality function value The final part in answering question EQ1, is to look at the evolution of the test suite quality function Q , based on the nearest neighbor entropy and introduced in subsection 4.6.

Fig. 15 shows the evolution of Q over time for the bicycle occlusion scenario solved with $t = 11$ steps (upper row) and $t = 13$ steps (lower row), with non-zero distance model instances included (left column) and removed (right column).

These graphs largely confirm our findings from Table 2: the variation methods SSV and RB produce only a small number of different scenarios, which is why their Q -value tends to zero in the left column. RBI on the other hand produces a Q -value of approximately 4, although the method terminates at 192 ($t = 11$) resp. 1024 ($t = 13$) models as discussed above. The situation changes when we remove all model instances exhibiting zero-distance to at least one other instance before evaluating Q . As expected, for all three variation methods, test suite quality increases vastly when restricted to non-zero distance instances. However, RBI outperforms SSV and RB even more drastically, reaching values of ≈ 6 and ≈ 8 , respectively, which correspond to $\approx 3/5$ and $\approx 2/3$ of the upper bound. Not only does RBI find more non-zero distance instances, but they are also farther away from each other.

Fig. 16 shows the quality evolution for the cut-out scenario (upper row) and the left-turn scenario (lower row) – again with (left column) and without (right column) zero-distance instances. Albeit the evolution of Q -values differs slightly from Fig. 15, the

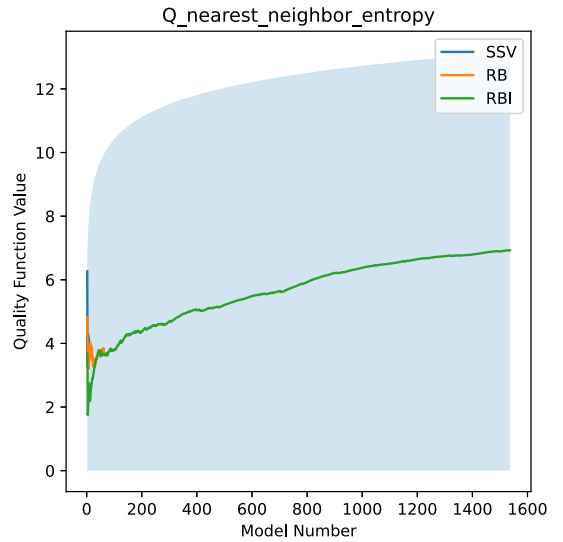
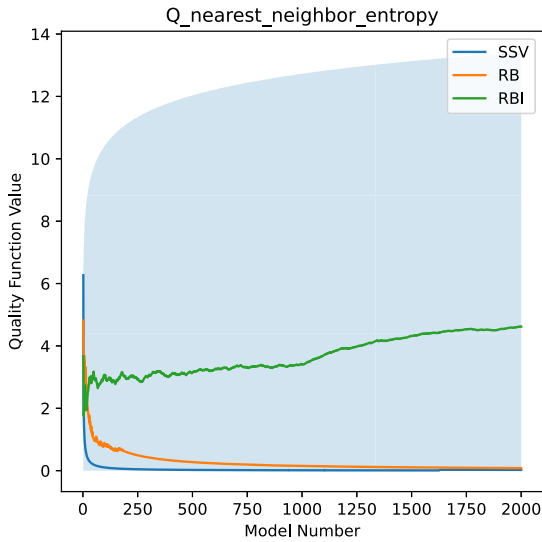
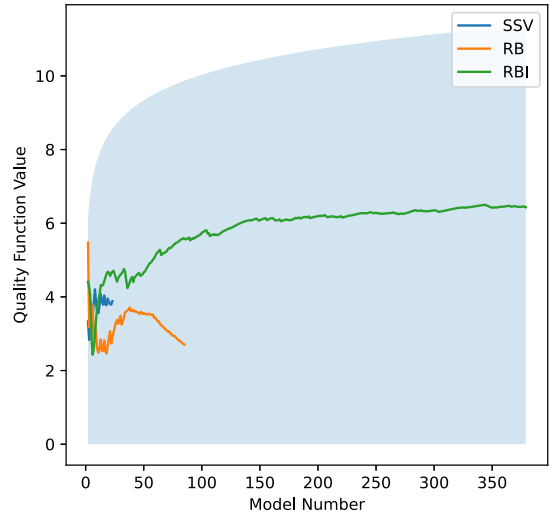
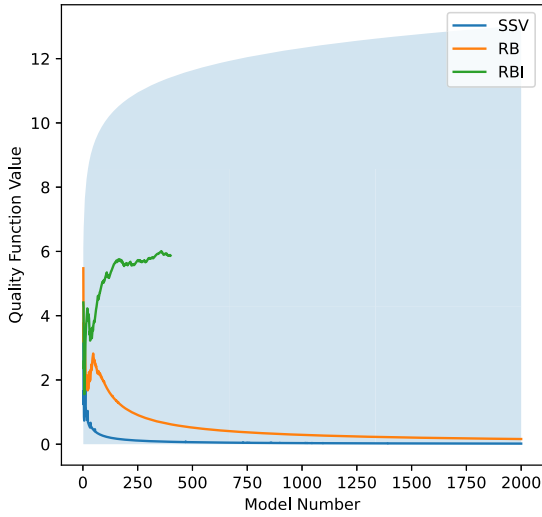


Fig. 16. Evolution of the quality function Q for the cut-out scenario of Fig. 12 (upper row) and the left turn scenario of Fig. 13 (lower row), including zero distances (left column), restricted to non-zero distances (right column). The light blue background visualizes the upper- and lower bounds of Q .

characteristics are identical. Therefore, these additional scenarios further corroborate the supremacy of the RBI variation method when compared to RB and SSV, independently of whether zero distance instances are removed from the test suite or not.

A note on the quality function in practice Let us summarize this first application of our entropy-based quality function Q in practice. Its usage enabled a comparison between test suites generated by the different variation methods. Moreover, the results are consistent with the authors' expectations: SSV as baseline, RB as a good idea, and RBI as a practical improvement thereof. The bounds of Q even make it possible to estimate how far a generated test suite from the best respectively worst possible test suite (although the upper bound is far from being sharp).

Finny, we note that the shift in the value of Q when removing zero-distance instances in Figs. 15 and 16 acts as empirical evidence for desired property #1 (Avoid redundancy), cf. subsection 4.6.2.

A note on coverage Naïvely, one could try to measure the coverage of the blocking methods – as there are only finitely many solutions to block. However, it turns out that estimating effectiveness in such a way is practically infeasible: Consider a TSC with the single invariant node and just two cars driving on a road. With 2 time steps the derived SMT formula has 80 atoms and 44 free variables. So we have a 44-dimensional solution space cut by 80 hyperplanes into $\sum_{i=0}^{44} \binom{80}{i} \approx 10^{24}$ regions in theory [55]. In 9 days we were able to find $\approx 17 \cdot 10^6$ models, respecting the blocked regions. During the SMT solving process, we counted the number of excluded Boolean combinations of atoms. If the recursive procedure in Algorithm 2 is called with k atoms and no solution has been found, we know

Table 3

Timings for model searching, pairwise distance calculation, and total time for the variation methods solver seed variation (SSV), recursive blocking (RB), and recursive blocking of invariants (RBI) in four different scenarios and $n = 2000$.

Scenario	Variation Method	Mean Model Searching Time [s]	Mean Pairwise Distances Calculation [s/n]	Total Time [m]
Bicycle Occlusion ($t = 11$)	SSV	0.11 ± 0.04	0.97 ± 0.05	37
	RB	47.1 ± 62.8	5.51 ± 0.40	1761
	RBI	0.29 ± 0.29	4.29 ± 0.17	3
Bicycle Occlusion ($t = 13$)	SSV	0.13 ± 0.05	1.30 ± 0.04	49
	RB	3.70 ± 23.2	7.66 ± 0.60	388
	RBI	0.34 ± 0.50	7.87 ± 0.09	77
Cut-Out	SSV	0.17 ± 0.07	1.35 ± 0.11	51
	RB	4.14 ± 11.2	7.67 ± 0.62	403
	RBI	0.55 ± 0.40	6.15 ± 3.66	11
Left Turn	SSV	0.09 ± 0.25	1.32 ± 0.02	48
	RB	70.1 ± 91.4	7.75 ± 0.40	2598
	RBI	0.83 ± 1.17	7.38 ± 0.79	287

that all 2^k Boolean combinations of those atoms are not part of the solution space and can be excluded. In our 9-day experiment, we were able to exclude $\approx 9 \cdot 10^{21}$ combinations, which are $\approx 0.7\%$ of a theoretical maximum of $2^{80} \approx 10^{24}$ Boolean combinations of atoms. Because we decided to cancel the experiment before termination of the algorithm, this is only a *theoretical* maximum. For the unexplored part of the search space, we cannot say how many combinations are actually contradictions and therefore outside of the solution space. Even if we found a large amount of models and blocking regions, we can neither estimate the exact size of the solution space nor a coverage percentage.

EQ2 – duration To gain insight into the effects of scenario duration on test suite quality, we again consider Table 2 and Fig. 15. The former indicates that SSV finds more instances with non-zero distance (+0.5%) and RB finds less (−2.15%) for $t = 13$ compared to $t = 11$. However, these relative changes seem rather insignificant. One could argue that the larger SMT formula for $t = 13$ leads to an even more unfortunate ordering of atoms for RB.

The performance of the RBI method reveals more interesting insights. While the relative ratio of non-zero instances to total instances found by RBI decreases from 122/192 to 517/1024 when transitioning from $t = 11$ to $t = 13$, this shift actually enables RBI to discover 395 additional scenarios. We remark that an increase in scenario length offers more possibilities for variation of the TSC’s time structure (which is encoded using Boolean variables). Therefore, more time steps extend the variation possibilities, specifically for the RBI method.

Regarding the impact of duration on the evolution of the quality function, we compare the graphs of Fig. 15: upper left with lower left and upper right with lower right. In both cases we observe that the duration has no influence on which variation method produces the best results, i.e. $RBI > RB > SSV$. In the left column, we note that, for $t = 11$ RBI reaches a maximum of ≈ 6 is quite early on, whereas for $t = 13$, the quality function keeps increasing until the algorithm terminates with 2000 models. Both RB and SSV approach zero rapidly, independent of the scenario duration. In the right column, we can see RBI plateauing at ≈ 6 for $t = 11$, but also increasing until the end for $t = 13$. For RB and SSV the evolution of the quality is inconclusive, as too few non-zero distance instances were found. However, the main difference in the graphs (upper row vs. lower row) is the length of RBI’s graph – which we already explained.

Another possibility is to manipulate the temporal length of the generated scenarios. Recall that the step size for all experiments in this work was set to 1 s. Instead of changing the number of time steps, varying the step size may also lead to increases in quality. To understand this, recall the encoding scheme sketched in subsection 4.4. First, transitions between invariant nodes only occur at step boundaries, so step sizes (without a common divisor) can lead to completely different concrete scenarios. Secondly, vehicle trajectories are encoded as Bézier splines, where each spline segment corresponds to one step (so, the average speed on a segment is the segment length divided by step size). Given the step size, TSC_{2SMT} restricts the segment length in relation to heading change. This ensures that curve radius and lateral acceleration are within the vehicle’s driving capabilities. Therefore, the time step size has a direct impact on the possible velocity range on curved trajectory segments. However, the experimental evaluation of step size evaluation is left for future work.

EQ3 – performance To evaluate the performance and, therefore EQ3, let us first recall prior work – where an initial feasibility assessment of the SMT-based sampling approach using Z3 was provided [13, Table 1]. There, we observed that SMT solving of a non-trivial TSC with a scalable number of invariants – up to 77 in the experiment – was possible in at most two minutes on a standard notebook. This indicates the general feasibility of the SMT-based sampling approach, even for complex TSCs.

We now assess the timings for $SMT_{2Instance}$ on our four handcrafted examples and for each of the three variation methods. Specifically, Table 3 displays the mean model searching time, mean pairwise distances calculation, and total time for the 12 resulting combinations. Recall from Table 2 that for ‘Bicycle Occlusion’ ($t = 11$ and $t = 13$) as well as ‘Cut-Out’, the variation method RBI does not manage to produce the required 2000 instances. So, these timings for RBI are not really comparable.

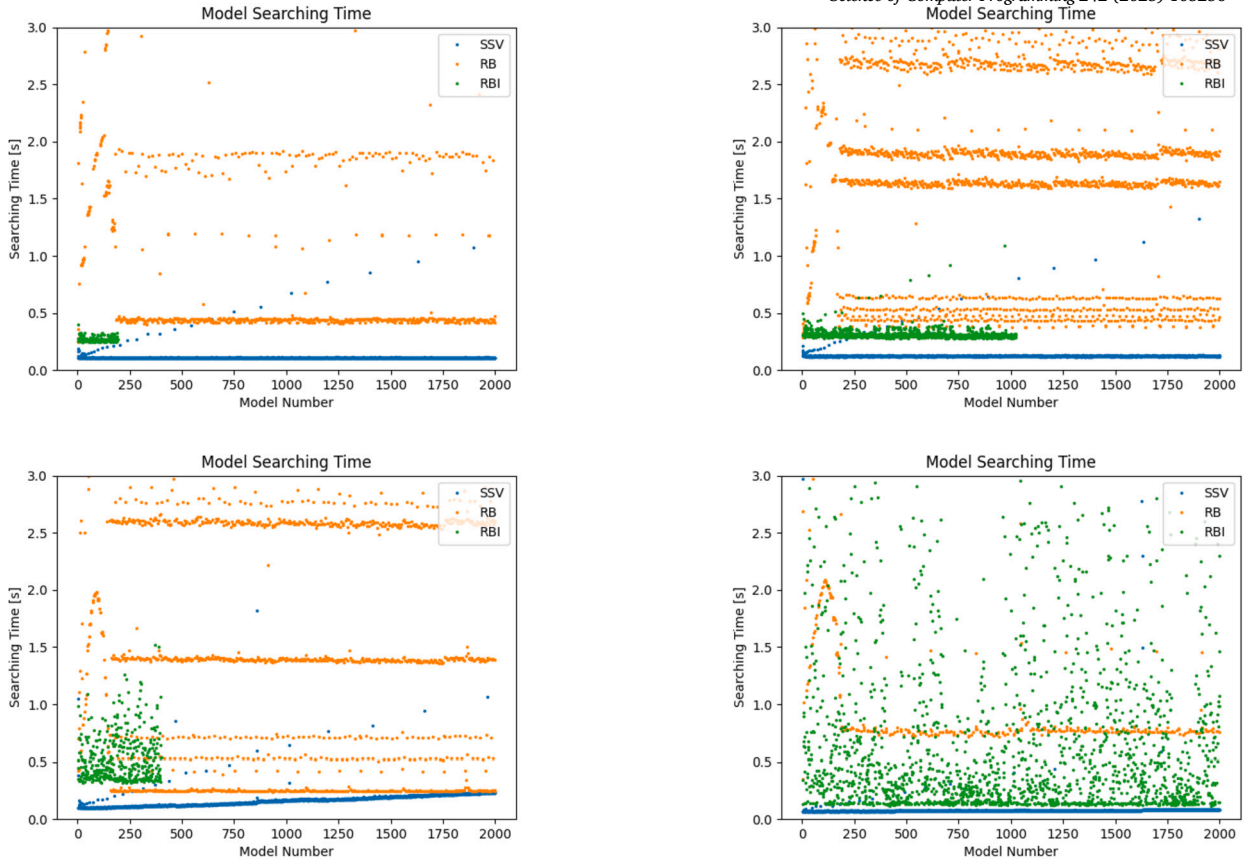


Fig. 17. Model searching times for the variation methods solver seed variation (SSV), recursive blocking (RB), and recursive blocking of invariants (RBI) in the four scenarios ‘Bicycle Occlusion’ ($t = 11$, upper left), ($t = 13$, upper right), ‘Cut-Out’ (lower left), and ‘Left Turn’ (lower right) with $n = 2000$ each. The y-axis is capped at 3 s.

Considering this, we can deduce from Table 3 that the baseline method SSV is the fastest in terms of model searching and distance calculation. The former is no surprise, as there is no extra computational cost in solver seed variation and the length of the SMT formula remains constant. The latter is likely caused by the fact that most of the distances evaluate to zero. The timings for the pairwise distances calculation between RB and RBI are roughly even and offer no insights.

The key take away is that RB is extremely slow when searching for new models. On the one hand, this is to be expected from Algorithm 2, as the formula grows with each blocking clause. However, when comparing the model searching time of RB to RBI in the ‘Left Turn’ scenario – for which RBI did manage to produce 2 000 instances as well – we observe a drastically increased mean model searching time by a factor of ≈ 85 . Again, this indicates that i) the choice of atoms to create blocking clauses from and ii) the ordering of these atoms likely plays a pivotal role in the capabilities of Algorithm 2 regarding EQ1 and EQ3.

As a final illustration regarding timings, we present Fig. 17, which shows the evolution of the searching time for new models during run-time. Note that the y-axis is capped at 3 s here, so slower models do not appear in the charts. Fig. 17 nicely explains the mean model searching times of Table 3: SSV remains essentially constant, RBI takes slightly more time than SSV but terminates when no more blocking clauses can be formed, and RB is by far the slowest method here. However, it is noteworthy that for RB there seems to be groupings of SMT formulae that are easy to solve (near RBI), further away, or very far away. Identifying which atoms lead to which blocking clauses may lead to valuable insights regarding the improvement of SMT-based sampling strategies.

6.2. Evaluation of test suite execution

We now highlight how the overall toolchain can be applied in a real-world setting using two previously generated test suites and a simple, custom ADS model in the CARLA simulator.

ADS model We have implemented a rudimentary ADS model using CARLA’s PythonAPI for evaluating our toolchain. For its perception input, it relies on an occlusion-sensitive sensor. The planner component essentially extends CARLA’s behavior agent by supplying custom (negative or positive) acceleration output without changing the steering behavior. For selecting an appropriate acceleration response at each simulation step, the model implements three modes: following, intersection and free driving. The overall acceleration response is calculated as a weighted sum over the individual responses of these three modes. These weights represent the applicability

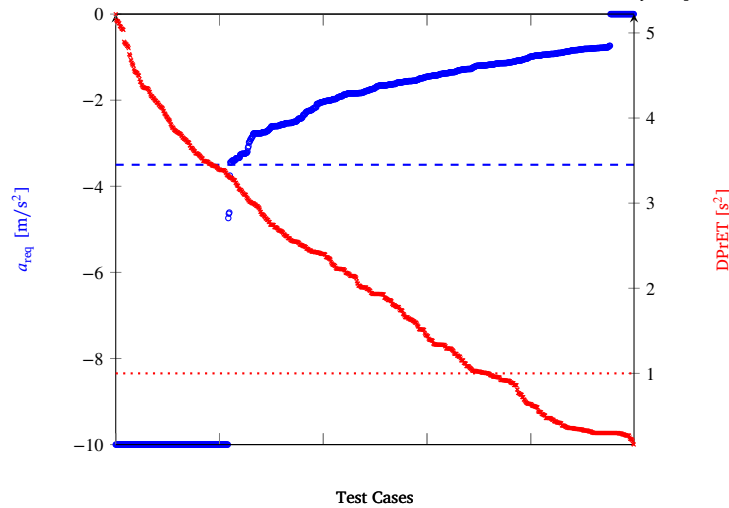


Fig. 18. Values of the criticality metrics Required Acceleration (a_{req}) and Duration-dependent Predictive Encroachment Time (DPrET) computed on the generated 517 test cases for the bicycle occlusion scenario, independently sorted for each metric. The a_{req} values are cut off at -10 m/s^2 . Target values are indicated by dashed (-3.5 m/s^2 for a_{req}) and dotted (1 s^2 for DPrET) lines.

of the mode to the current situation and are calculated based on the sensor input. The individual modes' acceleration computations are:

Following: Implements the rule of thumb of 'half the speed in km/h interpreted as m' with an additional adjustment based on the relative speed to the lead vehicle.

Intersection: Decelerates in case of a predicted intersection if it will be slowest to reach the intersection point and accelerates otherwise. For this, it uses the ego's planned trajectory and a constant-velocity, constant heading bounding-box prediction model for the intersecting vehicle.

Free driving: Gradually increases the speed until the ego approaches the target speed of 50 km/h .⁸

The ADS model is provided in the supplementary code base. We now investigate whether unsafe behavior is present in the SuT, e.g. due to faults in the mode arbitration and acceleration response calculation routines.

Results We evaluated the performance of the ADS model on the two test suites generated for the bicycle occlusion scenario ($t = 13$), cf. Fig. 3, and the cut-out scenario, cf. Fig. 12. The test suites consist of 517 (bicycle occlusion) and 379 (cut-out) concrete scenarios respectively, as generated by the RBI variation method. Note that the two test suites were restricted to non-zero distance instances, cf. Table 2. For both of them, TSC2OpenX generated a set of corresponding OpenDRIVE and OpenSCENARIO files. The CARLA simulator was able to parse and execute all files, and a subsequent manual inspection of the simulation results showed valid simulation behavior. In order to gain insight on the behavior of the ADS model, we employed the TSC2CARLA toolchain.

Bicycle occlusion scenario Recall the requirements r_1 from equation (1) (a_{req}) and r_2 from equation (2) (DPrET) for this abstract test case from subsection 4.2. After toolchain execution, we evaluated r_1 and r_2 based on the simulation logs. In total, the ADS model passed on 174 (34%) the cases and failed on the remaining 343 cases (28%), i.e. one of the two target values was violated. Fig. 18 reports the computed criticality metrics on the test cases. It can be seen that the metrics are consistent in the sense that all test runs that violate the a_{req} target also violate the DPrET target.

In 35 (10%) of the failing cases, the simulator recognizes a collision between the SuT and another vehicle. The remaining failing cases show both near collisions and uncritical scenarios. A manual investigation of the collisions showed that the SuT slowed down before the collision in all cases. In total, the bicycle collided 19 times (54%) with the side of the Ego vehicle, and 16 times with the front. In the majority of the front collisions (10 cases), the SuT initiated a (too late) braking maneuver. The SuT did not brake before side collisions. In 4 cases, the SuT even tries to avoid a collision by accelerating shortly before the side impact.

Because all passing test cases are collision-free, we can conclude that the test criterion was adequate to judge safe behavior of the SuT. The diversity of the test suite generated by TSC2CARLA allowed to uncover the inability of the SuT to handle certain crossing situations in the occlusion scenario. Suitable adaptations could now be implemented, e.g. disallowing collision avoidance through acceleration when being close to the other traffic participant.

⁸ For the cut-out scenario we reduced the ADS model's target speed to 18 km/h .

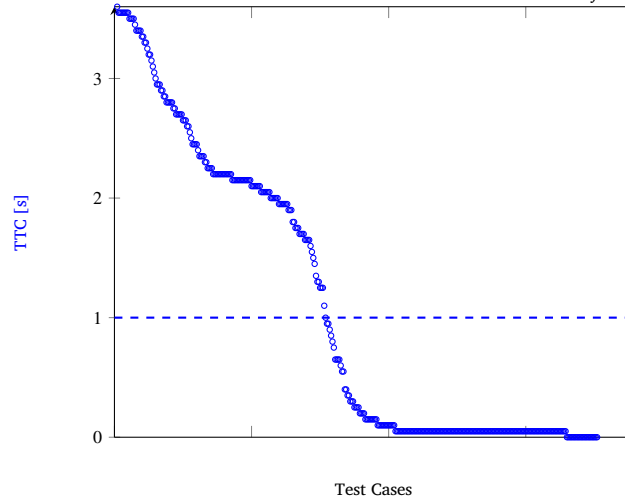


Fig. 19. Values of the criticality metric Time-To-Collision (TTC) computed on the generated 379 test cases for the cut-out scenario, sorted from uncritical (left) to critical (right). The target value of 1 s is indicated by a dashed line. Test cases with $TTC > 10$ s have been dropped for readability.

Cut-out scenario In the cut-out scenario of Fig. 12, the SuT reacted successfully to the truck in front of the ego vehicle, as the simulator did not report collisions between the Ego vehicle and the truck. Here, we use the requirement r_3 from equation (3) (TTC) to evaluate the test cases. The test results plotted in Fig. 19 show that 155 test cases (41%) passed with TTC above the target value of 1 s. In 173 test runs (77% of the 224 failing test runs), a collision between a vehicle and a static object is reported. This means that the SuT does not manage to brake in front of the construction site barrier. All the collisions result in a reported TTC of 0.05 s or less, which is consistent with the frame rate of the simulator.⁹

In summary, subjecting our simple ADS model to two test suites generated from TSC-specified abstract test cases, we uncovered unsafe behavior present in the ADS model. Although further experiments are necessary, this indicates the applicability of TSC2CARLA toolchain for scenario-based verification in a virtual environment.

7. Limitations and future work

While the evaluation shows promising results of our approach, we acknowledge two major limitations.

First, the formal basis behind TSCs assumes a declarative open-world semantics, meaning that anything not constrained might occur. Since we base the sampling process on SMT solving, only solutions are found that correspond to variables introduced in the SMT formula. Therefore, additional traffic participants or road infrastructure not specified in the TSC will not be generated during sampling, despite such solutions potentially satisfying the given TSC. A possible solution can be found by leveraging the compositionality of TSCs and use additional parallel compositions specifying the existence of further objects in the scenario. These compositions can be drawn from a catalog and added to the abstract test case in a combinatorial manner during scenario generation.

The second limitation concerns the partialization of the SuT trajectory, which can lead to unintentionally leaving the original abstract test case due to behavioral freedom of the SuT. While such behavior is generally not preventable in scenario-based testing, it can be minimized. For example, Scenic uses an iterative sampling-scene-generation approach and thus, only valid scenes from the abstract test case are presented to the SuT as possible inputs (which may still lead to invalid behavior of the SuT). Contrary to this, our approach of statically removing complete parts of the trajectory may increase chances of leaving the abstract test case. However, it has the advantage of disentangling scenario generation and test execution – allowing for test suite generation without access to the SuT. Although we proposed TSC monitoring for catching such instances during test execution, it is better to prevent them entirely. For this, we imagine injecting information about the original test case into the planner of the SuT. In this way, it can restrict its search space for possible trajectories to those satisfying the abstract test case.

Beyond addressing these limitations, there are several directions for future work that the authors deem interesting. (i) Applying the TSC2CARLA toolchain to a larger set of abstract test cases. Translation of an entire scenario catalogue for a given driving automation, e.g. for an automated lane keeping system (ALKS) in the sense of UN. Regulation No. 157 [56], into abstract scenarios. Then, a simple application of our toolchain could easily generate, execute, and evaluate the entire abstract test suite. Another option is to use highly critical scenarios from accident databases as a source for abstract test cases [57]. (ii) Investigating more refined variation methods for SMT-based sampling. As we have seen using RBI, the quality of test suites was immensely improved by restricting the blocking regions to atoms containing Boolean variables. However, RBI was not always able to generate the requested number of instances. Uncovering which atoms of the SMT formula – when blocked – generate the most variety among instances is a promising idea. Another potential

⁹ The minimal TTC is reached in the last simulation frame before the collision.

variation method is to steer the sampling process towards highly critical scenarios by adding (linear) constraints on criticality metrics to the SMT formula. (iii) Exploring scenario distance measures other than DTW. (iv) Formalizing properties on quality measures for test suites and considering further candidate functions. While the quality function Q fulfills many desirable properties, it also exhibits some weaknesses. Finding new quality measures and formally proving their properties will be included in the authors' future work. (v) More expressive requirement specification languages can be incorporated, as our implementation currently supports only criticality metrics for requirement evaluation. Finally, (vi) the `TSC2CARLA` toolchain can be adapted for applications within the development process other than testing. For example, for ADSs, a criticality analysis [5,37] or a hazard analysis and risk assessment [36] will benefit from highly automated generation and execution of abstract scenarios instances.

8. Conclusion

This work presented `TSC2CARLA`, a framework and prototypical toolchain for scenario-based verification, addressing the research questions from section 1. In particular, `TSC2CARLA` facilitates highly automated generation and execution of test suites for virtual testing of ADSs based on abstract scenarios, cf. sections 4 and 5, therefore addressing **RQ1**. Using abstract scenarios (instead of logical scenarios) presents a novelty compared to other contemporary approaches. In essence, we propose to use TSCs – a formal scenario language founded in first-order logic – for which we use SMT-solving to generate concrete scenarios. When the SuT trajectory is partially removed, these concrete scenarios become test cases such that an SuT model can be tested in a simulation environment, e.g. CARLA. Requirements for test cases can be specified using criticality metrics, which are subsequently evaluated on the simulation log data. Based on a formal notion of scenario distance provided by Dynamic Time Warping, we discussed desirable properties of quality functions for test suites and introduced the nearest neighbor entropy as a first viable candidate – addressing **RQ2** in subsection 4.6. Regarding **RQ3**, in subsection 4.5, we introduced three variation methods for SMT-based sampling of concrete scenarios: solver seed variation, recursive blocking, and recursive blocking of invariants. Preliminary experiments on three different abstract scenarios specified as TSCs, described in subsection 6.1, showed that RBI consistently generated the best results.

Finally, we showed the feasibility of our toolchain for virtual scenario-based verification for two test suites generated from TSC-specified abstract scenario using a non-trivial ADS model in CARLA. In both cases, the evaluation of requirements based on a combination of criticality metrics and thresholds uncovered critical behavior of the ADS model.

CRedit authorship contribution statement

Philipp Borchers: Writing – review & editing, Writing – original draft, Visualization, Software, Investigation, Formal analysis. **Tjark Koopmann:** Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Conceptualization. **Lukas Westhofen:** Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Conceptualization. **Jan Steffen Becker:** Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Conceptualization. **Lina Putze:** Writing – review & editing, Writing – original draft, Visualization, Methodology. **Dominik Grundt:** Writing – original draft. **Thies de Graaff:** Investigation, Software, Validation, Visualization, Writing – review & editing. **Vincent Kalwa:** Writing – original draft, Software. **Christian Neurohr:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. TSC formal semantics

In the following, we present the formal semantics of the subset of the TSC language that is used during this paper. For the full language definition, the reader is directed to the work by Damm et al. In the context of this paper, a *concrete scenario* defines a *scene* for every time point of the scenario. A scene defines a value for every attribute of every object that is present in the scenario.¹⁰ A spatial view The following definition holds for spatial views that contain only the following elements:

- Symbols from the bulletin board¹¹
- Somewhere-boxes (green dashed boxes)
- horizontal and vertical distance lines
- Attached predicates (boxed text connected to a symbol)

Each spatial view and each somewhere box span some so-called *frame*. The spatial relations inside a frame are translated to a mathematical constraint based on the *anchors* of the symbols and somewhere boxes. Throughout this paper we assume that every symbol

¹⁰ According to Menzel et al. [4], a scene also defines relations. We do not consider communication in this work and restrict ourselves to spatial relations which can be inferred from the object attribute values.

¹¹ For a clearer presentation of the TSCs in this paper, the roads have been omitted from the bulletin board.

Table A.5
Semantics of basic charts; $s(t) \models \phi_f$ denotes satisfaction of the spatial view according to Definition 10.

Name	$s, [b, e) \models BC$ if:	Syntax
Invariant node	$b < e \wedge \forall t \in [b, e) : s(t) \models \phi_f$	
Condition node	$s(b) \models \phi_f$	
Sequence of A and B	$\exists t \in [b, e) : s, [b, t) \models A \wedge s, [t, e) \models B$	
Parallel composition of A and B	$s, [b, e) \models A \wedge s, [b, e) \models B$	
Choice of A and B	$s, [b, e) \models A \vee s, [b, e) \models B$	

and every somewhere box has four anchors in its corners that translate to the coordinates of the represented object's (respectively somewhere box) axis aligned bounding box corners: $(o.min_x, o.min_y)$, $(o.max_x, o.min_y)$, $(o.min_x, o.max_y)$, $(o.max_x, o.max_y)$ where o stands for the represented object, respectively denotes the bounding box.

Definition 10 (Spatial View Semantics). For some frame f , let

- \mathcal{A} be the set of all anchors of symbols and somewhere boxes in f , and the anchors of the surrounding somewhere box, if any. Each anchor $a \in \mathcal{A}$ stands for a position (x_a, y_a)
- \mathcal{B} be the set of somewhere boxes in f , where each $b \in \mathcal{B}$ is translated to a constraint ϕ_b (applying this definition inductively)

Then, f is translated to a constraint

$$\phi_f := \exists (b.min_x, b.max_x, b.min_y, b.max_y)_{b \in \mathcal{B}} \in \mathbb{R}^{4|\mathcal{B}|} : \bigwedge_{\psi \in \Psi} \psi \wedge \bigwedge_{b \in \mathcal{B}} \phi_b$$

where the set Ψ consists of the following constraints:

- For each pair $(a_1, a_2) \in \mathcal{A} \times \mathcal{A}$:
 - $x_{a_1} \sim x_{a_2}$ with $\sim \in \{<, =, >\}$ depending if the x -position of a_1 in the spatial view is before ($<$), equal ($=$) or after ($>$) the x -position of a_2
 - likewise $y_{a_1} \sim y_{a_2}$ for the y -position
- $|x_{a_1} - x_{a_2}| \sim X$ for each horizontal distance line between anchors a_1 and a_2 labeled with some constraint $\sim X$
- likewise $|y_{a_1} - y_{a_2}| \sim X$ for each vertical distance line
- $P(o)$ for each attached predicate labeled with some expression P and connected to the symbol for some object o .

Note, that the set \mathcal{A} does not contain symbols contained within some nested somewhere box. Thereby, no implicit relations between objects in different somewhere boxes are created, only between the somewhere boxes itself.

With the help of Definition 10 we can define the semantics of existential TSCs. As explained above, an existential TSC consists of a basic chart, which is a combination of other basic charts.

Definition 11 (Semantics of Existential TSCs). A basic chart BC is satisfied on a left-closed, right-open interval $[b, e)$ on a concrete scenario s , if $s, [b, e) \models BC$ holds according to Table A.5.

A concrete scenario s satisfies an existential TSC with basic chart BC if there exists some interval $[b, e)$ such that $s, [b, e) \models BC$.

Appendix B. Full formulas

B.1. Temporal encoding

This appendix shows the encoding of the temporal structure of the TSC shown in Fig. 2.

Initial condition The initial condition of the BMC problem is a conjunction of the following constraints:

$$\begin{array}{lll} & \neg complete_A^0 & ok_A^0 \\ started_B^0 \implies complete_A^0 & \neg complete_B^0 & ok_B^0 \\ started_C^0 \implies complete_B^0 & \neg complete_C^0 & ok_C^0 \end{array}$$

$$\begin{array}{l}
\begin{array}{l}
\text{started}_E^0 \implies \text{complete}_D^0 \\
\text{extend}_E^0 \implies \text{complete}_E^0 \\
\text{started}_F^0 \implies \text{extend}_E^0
\end{array}
\quad
\begin{array}{l}
\neg \text{complete}_D^0 \quad \text{ok}_D^0 \\
\neg \text{complete}_E^0 \quad \text{ok}_E^0 \\
\neg \text{complete}_F^0 \quad \text{ok}_F^0 \\
\neg \text{complete}_G^0 \quad \text{ok}_G^0
\end{array} \\
\text{complete}^0 \iff \text{complete}_C^0 \wedge \text{extend}_E^0 \wedge \text{complete}_G^0
\end{array}$$

Step constraints The following constraints are added for every $i = 0, 1, \dots, N - 1$ (with unrolling depth N):

$$\begin{array}{l}
\text{complete}_A^{i+1} \iff (\text{true} \wedge \text{ok}_A^{i+1}) \quad \text{ok}_A^{i+1} \iff (\text{true} \implies \text{ok}_A^i \wedge \text{inv}_A^i) \\
\text{started}_B^i \implies \text{started}_B^{i+1} \quad \text{started}_B^{i+1} \implies \text{started}_B^i \vee \text{complete}_B^{i+1} \\
\text{complete}_B^{i+1} \iff \text{started}_B^i \wedge \text{ok}_B^{i+1} \quad \text{ok}_B^{i+1} \iff (\text{started}_B^i \implies \text{ok}_B^i \wedge \text{inv}_B^i) \\
\text{started}_C^i \implies \text{started}_C^{i+1} \quad \text{started}_C^{i+1} \implies \text{started}_C^i \vee \text{complete}_B^{i+1} \\
\text{complete}_C^{i+1} \iff \text{started}_C^i \wedge \text{ok}_C^{i+1} \quad \text{ok}_C^{i+1} \iff (\text{started}_C^i \implies \text{ok}_C^i \wedge \text{inv}_C^i) \\
\text{complete}_D^{i+1} \iff \text{true} \wedge \text{ok}_D^{i+1} \quad \text{ok}_D^{i+1} \iff (\text{true} \implies \text{ok}_D^i \wedge \text{inv}_D^i) \\
\text{started}_E^i \implies \text{started}_E^{i+1} \quad \text{started}_E^{i+1} \implies \text{started}_E^i \vee \text{complete}_D^{i+1} \\
\text{complete}_E^{i+1} \iff \text{started}_E^i \wedge \text{ok}_E^{i+1} \quad \text{ok}_E^{i+1} \iff (\text{started}_E^i \implies \text{ok}_E^i \wedge \text{inv}_E^i) \\
\text{extend}_E^i \implies \text{extend}_E^{i+1} \quad \text{extend}_E^{i+1} \implies \text{extend}_E^i \vee \text{complete}_E^{i+1} \\
\text{started}_F^i \implies \text{started}_F^{i+1} \quad \text{started}_F^{i+1} \implies \text{started}_F^i \vee \text{extend}_E^{i+1} \\
\text{complete}_F^{i+1} \iff \text{started}_F^i \wedge \text{ok}_F^{i+1} \quad \text{ok}_F^{i+1} \iff (\text{started}_F^i \implies \text{ok}_F^i \wedge \text{true}) \\
\text{complete}_G^{i+1} \iff \text{true} \wedge \text{ok}_G^{i+1} \quad \text{ok}_G^{i+1} \iff (\text{true} \implies \text{ok}_G^i \wedge \text{inv}_G^i) \\
\text{complete}^{i+1} \iff \text{complete}_C^{i+1} \wedge \text{extend}_E^{i+1} \wedge \text{complete}_G^{i+1}
\end{array}$$

Target constraint In the last unrolling step, complete^N shall hold.

References

- [1] M. Maurer, J.C. Gerdes, B. Lenz, H. Winner, *Autonomous Driving: Technical, Legal and Social Aspects*, Springer Nature, 2016.
- [2] S.A.E. International, J3016: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles, 2021.
- [3] S. Ulbrich, T. Menzel, A. Reschka, F. Scholdt, M. Maurer, Defining and substantiating the terms scene, situation, and scenario for automated driving, in: 2015 IEEE 18th International Conference on Intelligent Transportation Systems, 2015, pp. 982–988.
- [4] T. Menzel, G. Bagschik, M. Maurer, Scenarios for development, test and validation of automated vehicles, in: 2018 IEEE Intelligent Vehicles Symposium (IV), 2018, pp. 1821–1827.
- [5] C. Neurohr, L. Westhofen, M. Butz, M.H. Bollmann, U. Eberle, R. Galbas, Criticality analysis for the verification and validation of automated vehicles, IEEE Access 9 (2021) 18016–18041, <https://doi.org/10.1109/ACCESS.2021.3053159>.
- [6] C. Neurohr, L. Westhofen, T. Henning, T. de Graaff, E. Möhlmann, E. Böde, Fundamental considerations around scenario-based testing for automated driving, in: 2020 IEEE Intelligent Vehicles Symposium (IV), 2020, pp. 121–127.
- [7] A. Eggers, M. Stasch, T. Teige, T. Bienmüller, U. Brockmeyer, Constraint systems from traffic scenarios for the validation of autonomous driving, SC-Square@FLOC, p. 89 <https://ceur-ws.org/Vol-2189/paper1.pdf>, 2018.
- [8] D.J. Fremont, E. Kim, T. Drossi, S. Ghosh, X. Yue, A.L. Sangiovanni-Vincentelli, S.A. Seshia, Scenic: a language for scenario specification and data generation, Mach. Learn. 112 (10) (2023) 3805–3849, <https://doi.org/10.1007/s10994-021-06120-5>.
- [9] W. Damm, S. Kemper, E. Möhlmann, T. Peikenkamp, A. Rakow, Traffic Sequence Charts - from Visualization to Semantics, Reports of SFB/TR 14 AVACS 117, SFB/TR 14 AVACS, vol. 10, 2017.
- [10] W. Damm, S. Kemper, E. Möhlmann, T. Peikenkamp, A. Rakow, Traffic sequence charts - a visual language for capturing traffic scenarios, in: ERTS 2018, 9th European Congress on Embedded Real Time Software and Systems, Toulouse, France, 2018.
- [11] W. Damm, E. Möhlmann, T. Peikenkamp, A. Rakow, A Formal Semantics for Traffic Sequence Charts, Springer International Publishing, Cham, 2018, pp. 182–205.
- [12] J.S. Becker, Partial consistency for requirement engineering with traffic sequence charts, in: Software Engineering Workshops 2020, CEUR Workshop Proceedings, 2020, <https://ceur-ws.org/Vol-2581/ase2020paper1.pdf>.
- [13] J.S. Becker, T. Koopmann, B. Neurohr, C. Neurohr, L. Westhofen, B. Wirtz, E. Böde, W. Damm, Simulation of abstract scenarios: towards automated tooling in criticality analysis, Zenodo (2022) 42–51, <https://doi.org/10.5281/zenodo.5907154>.
- [14] D. Baumann, R. Pfeffer, 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), 2021, pp. 1–5.
- [15] M. O’Kelly, A. Sinha, H. Namkoong, R. Tedrake, J.C. Duchi, Scalable end-to-end autonomous vehicle testing via rare-event simulation, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS’18, Curran Associates Inc., Red Hook, NY, USA, 2018, pp. 9849–9860, https://proceedings.neurips.cc/paper_files/paper/2018/file/653c579e3f9ba5c03f2f2f8cf4512b39-Paper.pdf.
- [16] A. Fehnker, Application of Evolutionary Algorithms to Analyze Criticality in Urban Traffic Scenarios, Master thesis, Carl Von Ossietzky Universität Oldenburg, 2022.

- [17] L. Sorokin, T. Munaro, D. Safin, B.H.-C. Liao, A. Molin, OpenSBT: a modular framework for search-based testing of automated driving systems, in: *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings, ICSE-Companion '24*, Association for Computing Machinery, New York, NY, USA, 2024, pp. 94–98.
- [18] F. Bock, A. Heinz, J. Lorenz, Efficient usage of abstract scenarios for the development of highly-automated driving functions, in: M. Bargende, H.-C. Reuss, A. Wagner (Eds.), 20. Internationales Stuttgarter Symposium, Springer Fachmedien, Wiesbaden, Wiesbaden, 2020, pp. 373–387.
- [19] Q. Song, R. Anderberg, H. Olsson, P. Runeson, Generating executable test scenarios from autonomous vehicle disengagements using natural language processing, in: *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2024, pp. 98–104.
- [20] M. Butz, C. Heinzemann, M. Herrmann, J. Oehlerking, M. Rittel, N. Schalm, D. Ziegenbein, SOCA: domain analysis for highly automated driving systems, in: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, 2020, pp. 1–6.
- [21] Association for Standardization of Automation, Measuring Systems, ASAM OpenSCENARIO dsl, <https://www.asam.net/standards/detail/openscenario-dsl>, 2022.
- [22] D.J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A.L. Sangiovanni-Vincentelli, S.A. Seshia, Scenic: a language for scenario specification and scene generation, in: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019*, Association for Computing Machinery, New York, NY, USA, 2019, pp. 63–78.
- [23] F. Schuldt, A. Reschka, M. Maurer, A Method for an Efficient, Systematic Test Case Generation for Advanced Driver Assistance Systems in Virtual Environments, Springer International Publishing, Cham, 2018, pp. 147–175.
- [24] F. Bock, C. Sippl, A. Heinz, C. Lauer, R. German, Advantageous usage of textual domain-specific languages for scenario-driven development of automated driving functions, in: *2019 IEEE International Systems Conference (SysCon)*, 2019, pp. 1–8.
- [25] A. d.Matos Pedro, T. Silva, T. Sequeira, J. Lourenço, J. Costa Seco, C. Ferreira, Monitoring of spatio-temporal properties with nonlinear SAT solvers, in: *International Conference on Formal Methods for Industrial Critical Systems*, Springer, 2022, pp. 155–171.
- [26] K. Scheibler, A. Eggers, T. Teige, M. Walz, T. Bienmüller, U. Brockmeyer, Solving constraint systems from traffic scenarios for the validation of autonomous driving, in: *Proceedings of the 4th SC-Square Workshop*, 2019, pp. 2–12, <https://ceur-ws.org/Vol-2460/paper2.pdf>.
- [27] M. Klischat, M. Althoff, Synthesizing traffic scenarios from formal specifications for testing automated vehicles, in: *2020 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2020, pp. 2065–2072.
- [28] F. Finkeldei, M. Althoff, Synthesizing traffic scenarios from formal specifications using reachability analysis, in: *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2023, pp. 1285–1291.
- [29] International Organization for Standardization, I.S.O. 26262, Road vehicles – Functional safety, 2018.
- [30] N. Kalra, S.M. Paddock, Driving to safety: how many miles of driving would it take to demonstrate autonomous vehicle reliability?, *Transp. Res., Part A, Policy Pract.* 94 (2016) 182–193, <https://doi.org/10.1016/j.tra.2016.09.010>.
- [31] S. Riedmaier, T. Ponn, D. Ludwig, B. Schick, F. Diermeyer, Survey on scenario-based safety assessment of automated vehicles, *IEEE Access* 8 (2020) 87456–87477, <https://doi.org/10.1109/ACCESS.2020.2993730>.
- [32] D. Nalic, T. Mihalj, M. Bäuml, M. Lehmann, A. Eichberger, S. Bernsteiner, Scenario-based testing of automated driving systems: a literature survey, in: *FISITA Web Congress*, vol. 10, 2020.
- [33] J. Sun, H. Zhang, H. Zhou, R. Yu, Y. Tian, Scenario-based test automation for highly automated vehicles: a review and paving the way for systematic safety assurance, *IEEE Trans. Intell. Transp. Syst.* 23 (9) (2022) 14088–14103, <https://doi.org/10.1109/TITS.2021.3136353>.
- [34] W. Damm, S. Kemper, E. Möhlmann, T. Peikenkamp, A. Rakow, Using traffic sequence charts for the development of HAVs, in: *ERTS 2018, 9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, Toulouse, France, 2018, <https://hal.science/hal-01714060>.
- [35] W. Damm, E. Möhlmann, A. Rakow, A Scenario Discovery Process Based on Traffic Sequence Charts, Springer International Publishing, Cham, 2020, pp. 61–73.
- [36] B. Kramer, C. Neurohr, M. Büker, E. Böde, M. Fränzle, W. Damm, Identification and quantification of hazardous scenarios for automated driving, in: M. Zeller, K. Höfig (Eds.), *Model-Based Safety and Assessment*, Springer International Publishing, Cham, 2020, pp. 163–178.
- [37] C. Neurohr, L. Westhofen, M. Butz, M.H. Bollmann, L. Putze, T. Koopmann, R. Gansch, M. Knoop, A. Rasch, B. Cojocar, J. Daube, Advances on the Criticality Analysis for Automated Driving Systems, *Tech. Rep.*, Mar. 2024, <https://doi.org/10.5281/zenodo.10815308>.
- [38] A. Pretschner, M. Leucker, 20 Model-Based Testing – A Glossary, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 607–609.
- [39] L. Westhofen, C. Neurohr, T. Koopmann, M. Butz, B. Schütt, F. Utesch, B. Neurohr, C. Gutenkunst, E. Böde, Criticality metrics for automated driving: a review and suitability analysis of the state of the art, *Arch. Comput. Methods Eng.* 30 (1) (2022) 1–35, <https://doi.org/10.1007/s11831-022-09788-7>.
- [40] T. Brade, B. Kramer, C. Neurohr, Paradigms in scenario-based testing for automated driving, in: *2021 International Symposium on Electrical, Electronics and Information Engineering, ISEEIE 2021*, Association for Computing Machinery, 2021, pp. 108–114.
- [41] D. Grundt, A. Köhne, I. Saxena, R. Stemmer, B. Westphal, E. Möhlmann, Towards runtime monitoring of complex system requirements for autonomous driving functions, in: *Proceedings Fourth International Workshop on Formal Methods for Autonomous Systems (FMAS), EPTCS*, 2022, pp. 53–61.
- [42] J.S. Becker, Safe linear encoding of vehicle dynamics for the instantiation of abstract scenarios, in: *FMICS 2024, Int. Conf. on Formal Methods for Industrial Critical Systems*, Milan, Italy, 2024, accepted for publication.
- [43] N. Björner, L. de Moura, L. Nachmanson, C.M. Wintersteiger, *Programming Z3*, Springer International Publishing, Cham, 2019, pp. 148–201.
- [44] V. Kalwa, TSC2OpenX – Realisierung einer Werkzeugkette zur Simulation abstrakter Verkehrsszenarien, Bachelor's thesis, Carl Von Ossietzky Universität Oldenburg, 2021.
- [45] M. Scholtes, L. Westhofen, L.R. Turner, K. Lotto, M. Schuldes, H. Weber, N. Wagener, C. Neurohr, M.H. Bollmann, F. Körtke, J. Hiller, M. Hoss, J. Bock, L. Eckstein, 6-layer model for a structured description and categorization of urban traffic and environment, *IEEE Access* 9 (2021) 59131–59147, <https://doi.org/10.1109/ACCESS.2021.3072739>.
- [46] T. Braun, J. Fuchs, F. Reisgys, L. Ries, J. Plaum, B. Schütt, E. Sax, A review of scenario similarity measures for validation of highly automated driving, in: *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*, 2023, pp. 689–696.
- [47] H. Sakoe, S. Chiba, Dynamic programming algorithm optimization for spoken word recognition, *IEEE Trans. Acoust. Speech Signal Process.* 26 (1) (1978) 43–49, <https://doi.org/10.1109/TASSP.1978.1163055>.
- [48] M. Shokoohi-Yekta, B. Hu, H. Jin, J. Wang, E.J. Keogh, Generalizing dynamic time warping to the multi-dimensional case requires an adaptive approach, *Data Min. Knowl. Discov.* 31 (2017) 1–31, <https://doi.org/10.1007/s10618-016-0455-0>.
- [49] R. Tavenard, An introduction to dynamic time warping, <https://rtavenard.github.io/blog/dtw.html>, 2021.
- [50] Y. Tao, A. Both, R.I. Silveira, K. Buchin, S. Sijben, R. Purves, P. Laube, D. Peng, K. Toohey, M. Duckham, A comparative analysis of trajectory similarity measures, *GISci. Remote Sens.* 58 (5) (2021) 643–669, <https://doi.org/10.1080/15481603.2021.1908927>.
- [51] J. Beirlant, E.J. Dudewicz, L. Györfi, E.C. Van der Meulen, et al., Nonparametric entropy estimation: an overview, *Int. J. Math. Stat. Sci.* 6 (1) (1997) 17–39, <http://jimbeck.caltech.edu/summerlectures/references/Entropy%20estimation.pdf>.
- [52] D. Goldfarb, On the complexity of the simplex method, in: *Advances in Optimization and Numerical Analysis*, Springer, 1994, pp. 25–38.
- [53] L. de Moura, N. Björner, Z3: an efficient SMT solver, in: C.R. Ramakrishnan, J. Rehof (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 337–340.
- [54] J. Ellson, E. Gansner, L. Koutsofios, S.C. North, G. Woodhull, Graphviz — open source graph drawing tools, in: P. Mutzel, M. Jünger, S. Leipert (Eds.), *Graph Drawing*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 483–484.
- [55] K. Fukuda, Polyhedral Computation, Department of Mathematics - Institute of Theoretical Computer Science ETH, Zurich, 2020.
- [56] United Nations, Economic Commission for Europe (UNECE), UN Regulation No. 157: Uniform provisions concerning the approval of vehicles with regard to Automated Lane Keeping Systems, 2022.

- [57] S. Babisch, C. Neurohr, L. Westhofen, S. Schoenawa, H. Liers, Leveraging the GIDAS database for the criticality analysis of automated driving systems, *J. Adv. Transp.* (2023), <https://doi.org/10.1155/2023/1349269>, publisher: Hindawi (May 2023).
- [58] P. Borchers, T. Koopmann, L. Westhofen, J.S. Becker, L. Putze, D. Grundt, T. de Graaff, V. Kalwa, C. Neurohr, TSC2CARLA: An Abstract Scenario-based Verification Toolchain for Automated Driving Systems (experimental data), <https://doi.org/10.5281/zenodo.14502996>.