# Formal Scenario-driven Logical Spaces for Randomized Synthetic Data Generation

Osama Maqbool and Jürgen Roßmann
*Institute for Man-Machine-Interaction, RWTH Aachen University, Germany*

Keywords: Logical Scenarios, Randomization, Synthetic Data.

Abstract: Simulations and synthetic data are a necessary supplement to real-world experiments in order to alleviate its effort, cost and risks. As demand of data for development and validation increases, simulations too must correspondingly be scaled. Variation of simulation parameters affords simulation designers control over the scope of *how* a simulation is scaled— they can chose a balance between target distribution of simulation variants and the degree of randomness— thereby achieving both the volume and diversity of synthetic data. This paper proposes logical scenarios as basis for simulation variation. Scenarios are formal human-readable scripts of simulations and test drives used within the automotive industry. They are defined at different abstraction levels, one of which is the logical scenario as a parameterized simulation model with description for parameters instead of concrete values. This contribution proposes methodologies to model the parameter descriptions in a modular fashion with parameter ranges, probability distributions and inter-relations. A randomization engine is introduced based on Markov chain Monte-Carlo methods to efficiently sample the modeled space. The result is a variety of simulation-independent concrete scenarios that follow the formal scenario specification.

## 1 INTRODUCTION

The growth of intelligent systems in various operational domains is marked by a significant increase in the complexity of systems, which has created demands for new and innovative strategies throughout the system life-cycle: from development through validation to maintenance. Based often on Artificial Intelligence, complex systems depend on diverse and ample training data, and the high complexity also necessitates a validation strategy in a large number of operational situations. Case in point is the Automobile Industry, where Wachenfeld and Winner (Wachenfeld and Winner, 2016) estimate 6.62 billion test kilometers for the validation of autonomous driving functions.

A natural alternative to costly real-world experiments is generating synthetic data from simulations. This is achieved in a number of examples in different ways. A single or set of parameters may be varied per a given distribution to study a particular aspect of the whole system (Wagner et al., 2018), the operational domain of a system may be randomized (Khirodkar et al., 2019), or a systematic variation approach may be built up allowing the designer full control over data distribution (Ben Abdessalem et al., 2016). This contribution considers a formal scenario-based approach (Menzel et al., 2018) coupled with digital twins for generating synthetic data in a systematic manner. Scenarios are used within the automotive industry as standardized descriptions of a *scene flow* and are used mainly for validation of complex Automated Driving Systems (ADS). Scenarios can be parameterized to create logical spaces, wherein parameter values may be varied to generate a variety of concrete scenarios. This method of parameter variation offers various advantages: a) it works with simulation-independent and standardized formats for describing situations, b) it generates semantically correct data i.e. a valid scenario, and c) it has potential applicability throughout the development life-cycle due to the inherent integrate-ability of scenarios throughout the cycle (Sippl et al., 2019).

This paper proposes a layered approach for modeling the logical space of scenarios — specified via a test specification — a proposed addition to formal scenario specifications. Parameters may be modeled using reusable building blocks in a piece-wise manner, with corresponding probability distributions, and may be constrained together via inter-parameter constraints. A randomization engine is introduced to sample the multi-dimensional parameter space (or



Maqbool, O. and Roßmann, J.
Formal Scenario-driven Logical Spaces for Randomized Synthetic Data Generation.
DOI: 10.5220/0010816400003119
In *Proceedings of the 10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2022)*, pages 203-210
ISBN: 978-989-758-550-0; ISSN: 2184-4348

logical space) via variants of Markov chain Monte-Carlo Sampling. The resulting concrete scenarios are brought to life in a multi-domain 3D simulation engine, within which the complete framework is integrated as a dynamic library.

The rest of the paper is organized as follows. Section 2 presents the related work, Section 3 gives an overview of different types of scenarios. Section 4 explains in detail the modeling approach for logical spaces and corresponding sampling methods. Section 5 presents two exemplary cases from the automotive domain, and Section 6 concludes the work.

## 2 RELATED WORK

Variation or randomization of simulation parameters has different purposes. One of the most common applications is the sensitivity analysis (Hartjen et al., 2019) of parameters. Within machine learning applications, domain randomization (Tobin et al., 2017) is used to increase the diversity of data and is especially beneficial in transfer learning applications. Generative Adversarial Neural Networks are neural networks (Goodfellow et al., 2014) that may be trained to generate synthetic data targeted towards challenging the system under training.

Systematic variation approaches target solely on the variation aspect and attempt to build models or grammar which inherently cater to the semantic correctness of a scenario. (Jiang et al., 2018) developed a stochastic grammar, which combined with a photo-realistic simulator yields valid 2D and 3D scenes. (Bagschik et al., 2018) developed an ontology for knowledge representation of automotive scenarios from which varied traffic scenes can be derived. (Feng et al., 2021) identifies decision parameters from a scenario description, and uses criticality metrics based on the decision variables to generate a testing scenario library.

Domain Specific Languages (DSL) provide users with a programming framework where the scenario execution can be described with value and probability distribution constraints. ASAM OpenScenario 2.0 standard (ASAM, 2021c) follows this approach. A comparable approach is the Scenic language (Fremont et al., 2019), a probabilistic programming language (PPL) designed for automotive scenarios. PPLs allow construction of probability distributions via expressive conditional constraints between variables. (Fremont et al., 2020) uses the formal probabilistic description in Scenic for test-case generation of autonomous vehicle safety scenarios.

DSLs with PPL-like attributes provide both expressiveness and a detailed control over design of variation space. However, the mentioned approaches are lacking on mathematical constraints between variables, which are crucial for 3D simulation, e.g. to place bounds on variation of trajectories and object geometries. This contribution focuses on the mathematical relations and inference techniques to samples with such constraints.

## 3 BACKGROUND ON SCENARIOS

Formal scenarios considered within this contribution were introduced in the PEGASUS project (Winner et al., 2019) as a validation strategy for complex automotive systems. A scenario is a simulation script or story, formally defined as “the temporal development between several scenes in sequence of scenes”, enriched by entities, actions and events, goals and values (Ulbrich et al., 2015).

### 3.1 Abstraction Levels

(Menzel et al., 2018) proposes three levels of scenario abstraction: functional, logical and concrete scenarios. *Functional* scenarios are at the highest level of abstraction, composed of semantic information in natural language, providing a common platform for cross-domain experts. On the other end, a *concrete* scenario contains state space variables and concrete parameters describing the environment, entities and their relations to each other. *Logical* scenarios are in the middle; they contain the state space variables describing the environment, entities and relations, but describe parameter ranges instead of concrete values.

### 3.2 Randomizing Scenarios

The logical space offers a systematic playground to specify variations of a scenario. Figure 1 illustrates a modeling approach for an automotive scenario. The static and dynamic aspects are typically specified via the ASAM OpenDRIVE (ASAM, 2021a) and OpenSCENARIO (ASAM, 2021b) standards respectively. The *test specification* file is a proposed addition to the formal scenario specification, an XML-file which refers to parameters within existing specification files, along with their ranges, probability distributions, parameter- and inter-parameter constraints, thereby providing a systematic modeling approach for the logical space. Once modeled, a *randomization engine* is proposed as the variation methodology to generate concrete scenario variants based on the logical

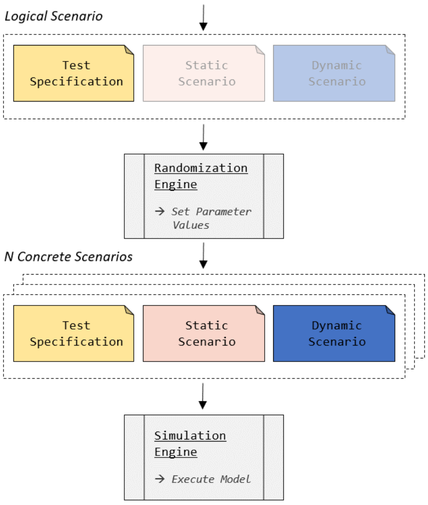



Figure 1: Test specification based description of logical scenarios.

space specification. The randomization engine uses Monte-Carlo based sampling of the modeled logical space. It works independently of the scenarios themselves, and is capable of taking mathematical and conditional constraints into account to sample from the specified distributions.

# 4 MODELING AND SAMPLING LOGICAL SPACES

A concrete scenario consists of a sequence of input data $\bar{u}(t)$, e.g. sensor data, initial states $\bar{s}_0$ and parameters $\bar{p}$. Based on the data, a simulation engine uses a digital twin $M$ of a system to generate state trajectories.

$$\bar{s}(t) = M(\bar{s}_0, \bar{u}(t), \bar{p}, t) \quad \bar{s}, \bar{s}_0 \in \mathbb{S},\ \bar{u} \in \mathbb{U},\ \bar{p} \in \mathbb{L} \quad (1)$$

$\mathbb{S}$, $\mathbb{U}$, $\mathbb{L}$ are the domains of their respective variables. In order to create variants of concrete scenarios, one can thus vary the static elements $\bar{p}$ by taking samples from $\mathbb{L}$.The initial state values $\bar{s}_0$ are also static and can be varied in a similar fashion. Therefrom follows the formal definition for logical space modeling:

> Logical space modeling is the modeling of $\mathbb{L}$, i.e. constituent values, constraining conditions and probability distribution functions that dictate whether and how often a value may be drawn.

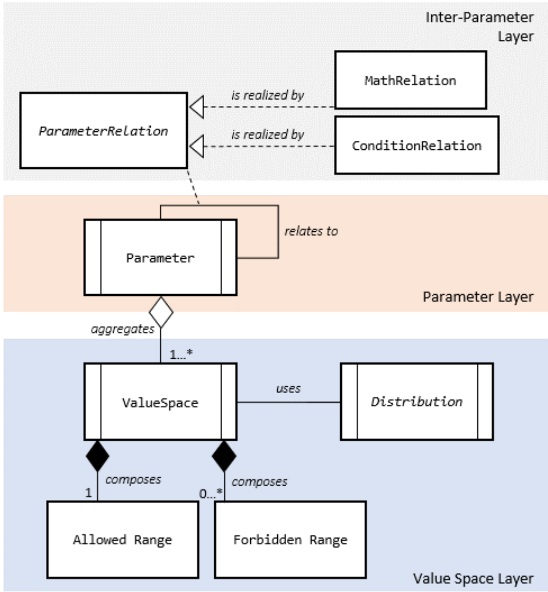



Figure 2: Layered approach for logical space modeling. Relations are read from left to right or top to bottom.

## 4.1 Modeling Methodology

The approach for modeling logical space $\mathbb{L}$ is illustrated in Figure 2. Starting from bottom to top, each layer builds some part of the logical space, and subsequent layers use the pre-built spaces to construct more complex spaces. Each layer is self-contained, and can be used independently to generate samples. The figure serves as a modeling guideline as well as the software architecture for the test specification reader, which must parse the XML elements and create corresponding entities (e.g. C++ objects). These are subsequently used by the randomization engine to generate samples (see Section 4.2).

### 4.1.1 Value Space Layer

This layer is the basic building block of the logical space. A value space is a continuous range or a discrete set from which a single parameter may draw values. The space is defined by allowed and forbidden ranges (or sets for discrete parameters). A probability distribution must be specified at this level to dictate how samples shall be drawn from the value space. Figure 3 illustrates a value space specified in the test specification for the speed of a vehicle on a highway, defined with an allowed and forbidden range, and a gaussian probability distribution.

```xml
<ValueSpace type="vehicle_speed_highway" basetype="double">
    <Range>[80:120]</Range>
    <ForbiddenRange>[110:115]</ForbiddenRange>
    <Dist type="Gaussian">
        <Mean>100</Mean>
        <StandardDeviation>10</StandardDeviation>
    </Dist>
</ValueSpace>
```

Figure 3: Value space in the test specification.

#### 4.1.2 Probability Distribution

The probability distribution can be specified individually for each parameter, or a multi-variate distribution may be specified for multiple parameters. The distribution can be derived from user knowledge as analytical models or from data as data-driven functions, and it must be implemented within the test specification reader inheriting from *Distribution* (see Figure 2). It can then be referred to in the test specification XML. Example is the gaussian distribution specified in Figure 3.

#### 4.1.3 Parameter Layer

Whereas value spaces have no direct meaning for the system per se, parameters refer to the properties of a concrete digital twin a system (and thereby also the real system). The same value space may be used by multiple parameters and multiple value spaces may be used by the same parameter, each with its occurrence likelihood. The multiplicity of value spaces allow the creation of discontinuous parameter ranges, diversified probability distributions and isolation of different operational domains of a parameter. Figure 4 illustrates an example of the parameter vehicle speed of a given ego vehicle, which may sample from two value spaces with an equal likelihood.

```xml
<Parameter ref="target_speed_ego" basetype="double">
    <ValueSpaces>
        <ValueSpace ref="vehicle_speed_highway">
            <Occurence>0.5</Occurence>
        </ValueSpace>
        <ValueSpace ref="vehicle_speed_city">
            <Occurence>0.5</Occurence>
        </ValueSpace>
    </ValueSpaces>
</Parameter>
```

Figure 4: Parameter specification in the test specification.

#### 4.1.4 Inter-Parameter Layer

The *Inter-Parameter Layer* allows the binding of various parameters through inter-parameter relations. Two types of relations are possible.

**Mathematical Constraints:** The parameters $p_i$ can be bounded with each other via mathematical constraints

$$C_e(\bar{p}) = \begin{cases} g(\bar{p}) = 0 \\ e(\bar{p}) \geq 0, \end{cases} \tag{2}$$

where $g$ and $h$ specify the mathematical equalities and inequalities respectively. An example of mathematical constraints is illustrated in Figure 5 within the XML-Tag *MathRelation*: speed of vehicle 1 must be greater than that of vehicle 2 by at least 5 km/h during an overtake scenario:

```xml
<ParameterConstraintRelations>
    <MathRelation>
        $vc_1_speed - $vc_2_speed &gt; 5
    </MathRelation>
    <CondRelation>
        <IF>$st_signal == "RED"</IF>
        <THEN>$vc_1_speed = 0</THEN>
        <THEN>$vc_2_speed = 0</THEN>
    </CondRelation>
</ParameterConstraintRelations>
```

Figure 5: Specification of inter-parameter constraints.

**Conditional Constraints:** *Conditional* here refers to the technical term in computer science, realized e.g. as If-Then-Else type constructs in programming languages.

$$\begin{aligned} & \text{IF}\,(p_0 = a \wedge p_1 \in [b,c]) \\ C_i(\bar{p}): \quad & \text{THEN}\,(p_m \in [x,y]) \\ & \vdots \end{aligned} \tag{3}$$

The conditional constraints must be formulated such that for a given set of concrete parameter values, all constraints $C_i$ can be evaluated to either true or false. The XML-Tag *CondRelation* in Figure 5 illustrates an example of a conditional relation: regardless of the specified distributions of vehicle speeds in a scenario, they must be zero if the parameter "state of traffic signal" is equal to "RED".

### 4.2 Generating Samples

Generating samples from $\mathbb{L}$ is achieved via a randomization engine, a task equivalent to making inferences about probability distributions. Harmonic with the layered modeling approach, samples can be drawn for each parameter individually or from the complete multi-dimensional constrained space.

#### 4.2.1 Sampling Value Spaces

Rejection sampling is used for drawing samples from value spaces. Samples are drawn from the specified distribution, and accepted only if the fall within the specified ranges.

#### 4.2.2 Sampling Parameters

Based on the occurrence frequency, an underlying value space is chosen, and its local sampling function is used to generate a sample.

#### 4.2.3 Sampling Constrained Logical Space

The constrained logical space $\mathbb{L}$ consists of all the parameters as well as their inter-relations. The sampling problem can be formulated in terms of a *target distribution* $f_T$ over the parameter vector $\bar{p}$:

$$f_T(\bar{p}) = \begin{cases} f_S(\bar{p}) & \boldsymbol{C_e}(\bar{p}) = 1 \wedge \boldsymbol{C_i}(\bar{p}) = 1 \\ 0 & otherwise, \end{cases} \quad (4)$$

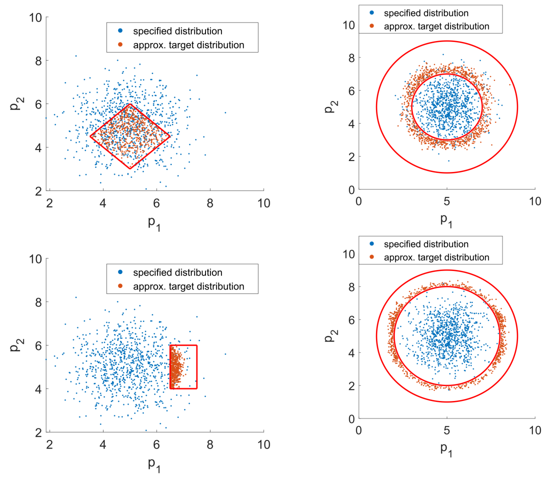
$\boldsymbol{C_e}(\bar{p})$ and $\boldsymbol{C_i}(\bar{p})$ are the mathematical and conditional relations respectively. $f_S$ is the distribution specified by the user, e.g. the gaussian distribution specified in Figure 3. The target distribution differs from the specified distribution in that it also takes parameter constraints into account. It is equal to $f_S$ when constraints hold and drops to zero otherwise. Samples therefore have to be drawn from the target distribution $f_T$, it is however not directly sample-able.

Rejection sampling, i.e. directly sampling from $f_S$ and rejecting samples violating constraints, is an obvious but inefficient candidate as the constraints may exclude high frequency areas of the specified distribution, leading to a very high rejection rate. To efficiently handle such conflicts, two variants of the Markov Chain Monte-Carlo (MCMC) method are used which draw from a distribution that approximates $f_T$. Essentially, they prescribe random sampling (the Monte-Carlo part) using a proposal distribution. The samples are drawn sequentially such that the acceptance of any sample is dependent on its probability with respect to the target distribution, relative to the previous sample, thus forming a Markov Chain.

#### 4.2.4 Resolving Mathematical Constraints with Metropolis Algorithm

The Metropolis algorithm (Roberts, 1996) is used by randomization engine to resolve linear mathematical constraints $\boldsymbol{C_e}$ and approximate the target distribution $f_T$. The approach is based on (Van den Meersche et al., 2009), and has the following steps:

1. **Choose a Mathematical Sub-space:** A single value space from each parameter is chosen for every iteration. The linear inter-parameter constraints, as well as the allowed and forbidden ranges of each value space are expressed as matrices $\boldsymbol{G}$ and $\boldsymbol{E}$.




(a) Metropolis algorithm with different linear constraints.

(b) Gibbs algorithm with different non-linear constraints.

$$\boldsymbol{C_e}(\bar{p}) = \begin{cases} \boldsymbol{G}\bar{p} = \mathbf{h} \\ \boldsymbol{E}\bar{p} \geq \mathbf{f}. \end{cases} \quad (5)$$

This step is the prepping of the problem for Metropolis algorithm.

2. **Draw Random Points:** A set of random points is generated such that they satisfies $\boldsymbol{C_e}$ by sampling from a proposal distribution. This random sampling is done via *xsample()*, an R function for solving linear inverse problems (Van den Meersche et al., 2009). The algorithm firstly eliminates the equality constraints $\boldsymbol{G}\bar{p} = \boldsymbol{h}$ by transforming the parameter vector $\bar{p}$ into a vector $\bar{p}^*$ with linearly independent elements.

$$\bar{p} = \bar{p}_0 + \boldsymbol{Q}\bar{p}^* \quad (6)$$

Where $\bar{p}_0$ is a solution of $\boldsymbol{G}\bar{p} = \mathbf{h}$ and $\boldsymbol{Q}$ serves as a basis for the null space of $\boldsymbol{G}$, such that $\boldsymbol{Q}^\top\boldsymbol{Q} = \boldsymbol{I}$ and $\boldsymbol{GQ} = 0$. Thus the boundaries are reduced to a set of inequalities

$$\boldsymbol{EQ}\bar{p}^* \geq \mathbf{f} - \boldsymbol{E}\bar{p}_0. \quad (7)$$

These inequalities are in-fact hyper-planes bounding the area of interest. Given an initial point that is within the hyper-planes, subsequent points are sampled sequentially, each dependent on last and each random. To generate a random point $z_2$ from a given point $z_1$, candidate $z_2^*$ is chosen by an arbitrary line from $z_1$ with length sampled from a normal distribution.

$$z_2^* = z_1 + \eta \quad (8)$$

If $z_2^*$ lies within the inequalities in Equation 7, it is accepted as a valid sample $z_2$. Otherwise, the point is "mirrored" across the first hyper-plane that intersects this line. If the new point lies within the bounds, it is accepted, otherwise the process is repeated.

The random walk using the mirror algorithm ensures that the points are generated from a symmetric distribution as required by MCMC, along with a high acceptance rate.

3. **Map Random Points to Target Distribution:** With a set of random points that all satisfy $\boldsymbol{C_e}$, a subset of the points is mapped to $f_T$. This is done sequentially with the MCMC criterion. Given a point $z_2$ and a previously accepted point $z_1$, if

$$\frac{f_T(z_2)}{f_T(z_1)} >= r, \tag{9}$$

then the point is accepted, otherwise rejected, $r$ being between 0 and 1. The process is then repeated.

Samples within an initial *burn-in* phase are typically ignored until the Markov chain converges to the target distribution. The choice of $r$ is also important to set the exploration-exploitation balance of the algorithm — whether it converges quickly to a local peak or chooses to explore low-density areas as well.

The algorithm results are illustrated in Figure 6a. Red lines denote the bounding constraints. Working with relative frequencies allow the algorithm to reduce rejection ratio yet still converge as best as possible to the specified distribution.

**Non-linear Constraints:** For non-linear, especially non-convex bounds, step 2 becomes a challenge. For this, another algorithm is introduced in the next section which bypasses this step altogether.

#### 4.2.5 Resolving Conditional Constraints with Metropolis within Gibbs Algorithm

The Gibbs algorithm (Smith and Roberts, 1993) is also a variant of the MCMC methods targeted more towards multivariate distributions. The algorithm uses the conditional probabilities rather than the joint probability and is capable of handling non-linear constraints, conditional constraints and both at the same time. The randomization engine applies a more robust variant, the "Metropolis within Gibbs" algorithm , in the following steps:

1. For the target distribution in Equation 4, choose a valid initial sample for $\bar{p}$. This is done via rejection sampling.

$$\bar{p}^0 = (p_0^0, p_1^0, ..., p_n^0) \tag{10}$$

2. Draw a sample for the first variable based on the previous values of other variable samples:

$$f_T(p_0|p_1^0, ..., p_n^0). \tag{11}$$

Since $f_T$ cannot be sampled directly, the sample is drawn with the Metropolis algorithm, albeit differently than before:

   (a) Choose a random value for $p_0$ *without* taking into account inter-parameter constraints $\boldsymbol{C_e}$ and $\boldsymbol{C_i}$.

   (b) For a concrete sample $p_0^*$, accept the sample if

$$\frac{f_T(p_0^*|p_1^0, ..., p_n^0)}{f_T(p_0^0|p_1^0, ..., p_n^0)} > r, \tag{12}$$

   otherwise reject it. This is the step where the constraints are taken into account. Since constraints $\boldsymbol{C_e}$ are mathematical equalities and inequalities, and constraints $\boldsymbol{C_i}$ are conditional statements, evaluation of both must return either true ($f_T$ is equal to $f_S$) or false ($f_T$ is zero).

3. Draw a sample for the rest of the variables similarly, using the new value of the previous variables in the conditional distribution in Equation 12. This completes one iteration, i.e. one complete sample. The complete process is then repeated for further samples.

Figure 6b illustrates the results of Metropolis within Gibbs, where the sampler converges to a normal distribution while being bounded within a 2-D torus.

The Metropolis within Gibbs method offers more flexibility than the Metropolis method, as a random sample is drawn first and *then* checked for constraints, making constraint handling simpler. Unlike Metropolis, this order is feasible here as variation takes place one dimension at a time, decreasing the likelihood of invalid samples. However, the algorithm is computationally expensive, and puts additional requirement on the implementation of *Distribution*, namely the calculation of conditional distributions.

## 5 EXAMPLES

As examples, scenario-based logical space sampling is applied to two automotive use-cases. The resulting concrete scenarios contain the initial states and parameters for digital twins which are simulated within the 3D simulation framework VEROSIM (Rossmann et al., 2013). The simulation framework is equipped to generate ground-truth sensor- and trajectory data that can be used to bridge the data requirements for training and validation purposes.

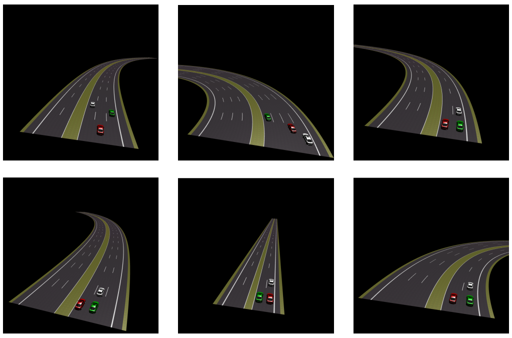
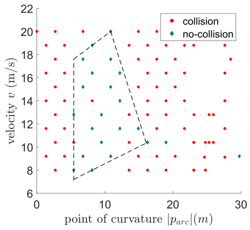
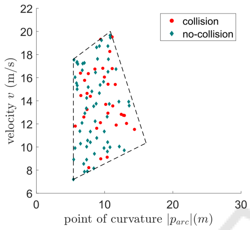


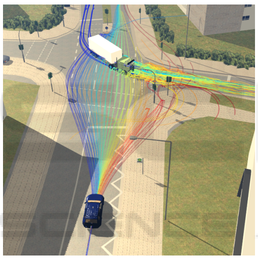
Figure 7: Randomizing static aspects of a scenario.



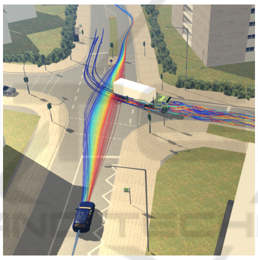

(a) Complete parameter space exploration.




(b) Targeted parameter space exploration.



(c) Simulation of the complete parameter space.



(d) Simulation of the targeted parameter space.

Figure 8: Investigating safe evasive maneuvers by randomizing the dynamic scenario.

#### 5.0.1 Randomizing Static Scenarios

Figure 7 illustrates an example of road layout randomization, specified by the OpenDRIVE and OpenSCENARIO formats. In this simple example, discrete value sets with random distributions are assigned to each of the following parameters of the scenario description: the road curvature, number of lanes, width of lanes and the initial coordinates of traffic participants. No inter-parameter relations are specified. The resulting variety of concrete scenarios can then serve as basis for simulation within different environments.

#### 5.0.2 Randomizing Dynamic Scenarios

Randomization of a dynamic scenario is illustrated in Figure 8 (based on a scenario from (Atorf and Roßmann, 2018)). An autonomous car approaches a road intersection while an oncoming truck takes a turn, cutting the path of the car. Given a constant behavior of the truck, the car must perform an evasive maneuver to avoid a collision.

The car drives with a velocity $v$ based on a set of target path points, out which the point of curvature $p_{arc}$ can be varied to achieve an evasive maneuver. The logical space modeling is done in two steps:

1. Firstly, the logical space is specified for the parameters $v$ and $p_{arc}$ by specifying feasible ranges and assigning a uniform distribution to each parameter. The simulations of the resulting concrete scenarios gives an overview of the collision and no-collision parameter regions, illustrated in Figure 8a. Figure 8c shows the simulated vehicle trajectories.
2. With an overview of the overall parameter space characteristics, regions more likely to result in the desired outcome (lower collisions) are isolated. This is done by adding mathematical constraints between $v$ and $p_{arc}$. These constraints are illustrated in Figure 8b as lines bounding the region of interest. The results are a higher ratio of no-collision points as well as the cleaner trajectory paths in the simulation in Figure 8d. Thus a detailed insight into the effects of parameter variation are achieved via iterative logical space modeling.

## 6 CONCLUSIONS

This contribution introduced a systematic framework for modeling logical scenarios and using them as a basis for parameter variation and simulation independent generation of diverse synthetic data. The XML-based test specification of logical scenarios allows a direct integration with formal scenario-based simulations, but is not restricted to these. A promising area of further research is the incorporation of the proposed framework within existing domain specific- and probabilistic programming languages in order to provide the designer not just low level constraints introduced here, but also higher-level modeling rules for description of the complete scenario.

## ACKNOWLEDGEMENTS

This work is part of the project “KImaDiZ”, supported by the German Aerospace Center (DLR) with funds of the German Federal Ministry of Economics and Technology (BMWi), support code 50 RA 1934.

# REFERENCES


ASAM (2021a). Asam opendrive. https://www.asam.net/standards/detail/opendrive/. Accessed: 2021-08-10.

ASAM (2021b). Asam openscenario. https://www.asam.net/standards/detail/openscenario/. Accessed: 2021-08-10.

ASAM (2021c). Asam openscenario 2.0. https://www.asam.net/project-detail/asam-openscenario-v20-1/. Accessed: 2021-09-08.

Atorf, L. and Roßmann, J. (2018). Interactive analysis and visualization of digital twins in high-dimensional state spaces. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 241–246. IEEE.

Bagschik, G., Menzel, T., and Maurer, M. (2018). Ontology based scene creation for the development of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1813–1820.

Ben Abdessalem, R., Nejati, S., Briand, L. C., and Stifter, T. (2016). Testing advanced driver assistance systems using multi-objective search and neural networks. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 63–74.

Feng, S., Feng, Y., Yu, C., Zhang, Y., and Liu, H. X. (2021). Testing scenario library generation for connected and automated vehicles, part i: Methodology. *IEEE Transactions on Intelligent Transportation Systems*, 22(3):1573–1582.

Fremont, D. J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A. L., and Seshia, S. A. (2019). 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, page 63–78, New York, NY, USA. Association for Computing Machinery.

Fremont, D. J., Kim, E., Pant, Y. V., Seshia, S. A., Acharya, A., Bruso, X., Wells, P., Lemke, S., Lu, Q., and Mehta, S. (2020). Formal scenario-based testing of autonomous vehicles: From simulation to the real world. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27.

Hartjen, L., Philipp, R., Schuldt, F., Friedrich, B., and Howar, F. (2019). Classification of driving maneuvers in urban traffic for parametrization of test scenarios. In *9. Tagung Automatisiertes Fahren*.

Jiang, C., Qi, S., Zhu, Y., Huang, S., Lin, J., Yu, L.-F., Terzopoulos, D., and Zhu, S.-C. (2018). Configurable 3d scene synthesis and 2d image rendering with per-pixel ground truth using stochastic grammars. *International Journal of Computer Vision*, 126(9):920–941.

Khirodkar, R., Yoo, D., and Kitani, K. (2019). Domain randomization for scene-specific car detection and pose estimation. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1932–1940.

Menzel, T., Bagschik, G., and Maurer, M. (2018). Scenarios for development, test and validation of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1821–1827. IEEE.

Roberts, G. O. (1996). Markov chain concepts related to sampling algorithms. *Markov chain Monte Carlo in practice*, 57:45–58.

Rossmann, J., Schluse, M., Schlette, C., Waspe, R., Van Impe, J., and Logist, F. (2013). A new approach to 3d simulation technology as enabling technology for erobotics. In *1st International Simulation Tools Conference & EXPO*, pages 39–46.

Sippl, C., Bock, F., Lauer, C., Heinz, A., Neumayer, T., and German, R. (2019). Scenario-based systems engineering: An approach towards automated driving function development. In *2019 IEEE International Systems Conference (SysCon)*, pages 1–8.

Smith, A. F. and Roberts, G. O. (1993). Bayesian computation via the gibbs sampler and related markov chain monte carlo methods. *Journal of the Royal Statistical Society: Series B (Methodological)*, 55(1):3–23.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30.

Ulbrich, S., Menzel, T., Reschka, A., Schuldt, F., and Maurer, M. (2015). Defining and substantiating the terms scene, situation, and scenario for automated driving. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 982–988. IEEE.

Van den Meersche, K., Soetaert, K., and Van Oevelen, D. (2009). xsample (): an r function for sampling linear inverse problems. *Journal of Statistical Software*, 30(Code Snippet 1).

Wachenfeld, W. and Winner, H. (2016). The release of autonomous vehicles. In *Autonomous driving*, pages 425–449. Springer.

Wagner, S., Groh, K., Kuhbeck, T., Dorfel, M., and Knoll, A. (2018). Using time-to-react based on naturalistic traffic object behavior for scenario-based risk assessment of automated driving. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1521–1528. IEEE.

Winner, H., Lemmer, K., Form, T., and Mazzega, J. (2019). Pegasus—first steps for the safe introduction of automated driving. In *Road Vehicle Automation 5*, pages 185–195. Springer.