J. Michael, J. Pfeiffer, A. Wortmann (Hrsg.): Software Engineering 2022 Workshops,
68 Digital Library, Gesellschaft für Informatik 2022

A Translation Semantics for Driving Simulation Languages

Jörn Schneider, Marvin Schneider[1]

Abstract: The development of advanced driver assistance systems and automated driving functions requires the usage of driving simulation as integral part of the software engineering process. Moreover, safety standards such as SOTIF (ISO 21448) and legal regulations give driving simulation a key role for the safety validation of automated driving functions by OEMs and Tier-1s as well as independent or governmental institutions. Even as new standards for driving simulation languages come into use, this gives rise to the need for translation tools between different driving simulator languages. Two major challenges in this context for translation tools are hitherto not well addressed: 1. Adaptability to new languages or versions thereof. 2. Correctness of translation. We elaborate on some of the central challenges in this regard, present a prototype of a retargetable translator for driving simulation languages, and a suiting translation semantics, as first cornerstones of a future approach to validate or verify translations.

Keywords: Driving Simulation; Formal Semantics; Automated Driving; SOTIF

1 Introduction

Driving simulation has become an integral part of automotive software engineering for driver assistance systems (DA) and automated driving functions (ADF) in the recent years. This is no surprise, as it enables the frontloading of activities such as testing and validation in the development process, supports a more agile style of software development, and allows for reproducible and safe virtual driving. Moreover, driving simulation can be utilized to train artificial intelligence algorithms, evaluate the user acceptance and behavior of drivers and passengers, as well as validate the behavior of DA and ADF systems.

The typical development process of automotive software involves different contributors and stakeholders from Tier-N-Suppliers to OEMs, and as agile as it may be, still takes a considerable amount of time. Consequently, it is not guaranteed that all involved driving simulations are either using identical language formats or are completely independent, i. e. having nothing in common regarding the considered road networks. Furthermore, the life cycle of a vehicle and thereby the relevance of related driving simulation results is much longer than the initial development phase, leading to (further) legacy issues. Moreover, driving simulations are considered as key to the validation of the safety of DA and ADF systems and their (future) type approval. This necessitates the involvement of independent (e. g. governmental) institutions, that need to replicate driving simulation results. Although

[1] {j.schneider, marv.schneider}@hochschule-trier.de, Trier University of Applied Sciences, Department of Computer Science, Schneidershof, Trier, Germany

it is certainly not sufficient to have a correct translation between different driving simulation languages (or language versions), to achieve reproducibility and comparability of different simulations, it can hardly be assumed that a correct translation is irrelevant or not necessary.

While there are many aspects of driving simulation with more or less relevance for different fields within and beyond automotive software engineering, the simulated road network is clearly among the most fundamental things that need to be described in one way or another. Consequently, that is the focus of this work. Since many vendors of driving simulation software used to develop their own proprietary description format, compatibility became an issue. The rising importance of driving simulations in automotive software engineering has led to efforts to harmonize this by a new family of ASAM e.V. standards. Regarding road networks *OpenDRIVE* is part of this family. This standard still evolves and the current released version is ASAM OpenDRIVE 1.7.0, the subsequent version 1.8.0 is projected to be released in November 2022, and work on version 2.0 is already in progress [As21]. Since it is unlikely that this will be the last change to the standard and because other description formats are still in use, tools that translate between different description language formats are required now and in future.

Two major requirements for translation tools in this context will be *adaptability* to new source and target languages or versions thereof, and *provable correctness of translation*.

The remainder of the paper starts with a brief description of languages considered in the context of our retargetable driving simulation language translator. Section 3 discusses the question how correctness of translation can be achieved and to this end considers the semantics of driving simulation languages in relation to the notion of formal semantics of programming languages. This is also the place where we discuss related work and possible solutions to these aspects. Thereafter the architecture and implementation of our translator is presented. In Section 5 its translation semantics is described. The last Section summarizes the presented results.

2 Driving Simulation Languages and Formats

We consider examples of languages with different levels of abstraction and different origin as well as application area. In all cases it is possible to describe the basic stationary aspects of streets, e. g. road geometries, lane markings, and traffic signs. The formats differ in the way they represent data and regarding additional aspects supported.

ASAM **O**pen **D**ynamic **R**oad **I**nformation for **V**ehicle **E**nvironment (ASAM OpenDRIVE) is an open standard (developed from a formerly proprietary language) for describing road and rail networks [As21]. It is XML based and supports complex connections between multiple roads and lanes. Traffic signs, traffic lights and specific road markings can also be defined.

SILAB is an example of a simulation software with a proprietary language for describing the simulation scenarios, and has its origin in the area of human-factors research [Wü]. It covers static description of roads, objects and environments, and notably supports dynamic road connections, i. e. the simulated road network can be changed during runtime, e. g. to allow the sequence of road segments to be picked by an algorithm or an operator. Moreover, events, which influence the state of the simulation, can be defined. Triggering of events is for instance possible via so-called hedgehogs (triggering points within lanes). Additionally, objects can be placed in the scenario with coordinates and a 3D-Model. An environment next to the street can also be defined with a surface texture and mathematical descriptions of terrain and tree distribution. More complex roads can be described via a software tool, which creates so-called Area2 language elements, that can be connected with other roads. The language is also used to integrate additional software to add further functionality to the simulation.

RoadXML is an open XML file format which describes the data in multiple layers for fast access for real time applications. In the four main layers it characterizes topology, logic, physical properties and visual representation [10].

OKSTRA® is a German standard for road design, documentation and traffic data acquisition [21]. It defines an XML file format and is used to design, construct and document road networks. OKSTRA® differs significantly from the above mentioned languages, as it was not developed to be used in simulation context.

3 Translation Correctness: Challenges and Solution Idea

3.1 Formal Semantics of Programming Languages

The problem to ensure that the translation from one driving simulation language to another is correct, has similarities with challenges in the domain of compiler design, where the translation of one high-level programming language to several low-level languages (e. g. assembler) should be true to the original, or when translating one high-level language into another. The existence of at least one commercially available optimizing compiler that is formally verified [Le16] demonstrates that this is a viable approach in practice.

In compiler design a formal semantics of a programming language exists (at least in the ideal case) that exactly describes how the state of a system shall be altered by each individual language construct. Leaving open for the moment what kind of formal mathematical model could serve this purpose for driving simulation, we can seek inspiration by looking at the case of translating from one programming language to another.

Given two languages L_1 and L_2 and two interpreters I_1 and I_2, and the goal of a semantics preserving translator T. Let E be the input alphabet and A the output alphabet for the

programs, and Σ the space of possible states of the interpreters. The interpreters and the translator can be considered as functions

$$I_1 : L_1 \times E^* \times \Sigma \to \Sigma \times A^*$$

$$I_2 : L_2 \times E^* \times \Sigma \to \Sigma \times A^*$$

$$T : L_1 \to L_2$$

Let's assume we have a formal semantics for L_1 and L_2, e. g. as denotational semantics, which could be defined as set of partial functions specifying the semantics of each language construct. For example, if **Com** is the syntactic set of commands of a language, the semantic function

$$C : \mathbf{Com} \to (\Sigma \to \Sigma)$$

provides for each valid command a function that describes the effects of the command on the state of the system. For a given language construct to assign a value to a variable the matching partial definition of C could be

$$C[\![x := a]\!] = \{(\sigma, \sigma[x/n]) | \sigma \in \Sigma \wedge n = \mathcal{A}[\![a]\!]\},$$

where $\mathcal{A} : \mathbf{AExp} \to (\Sigma \to \mathbf{N})$ is the semantic function for arithmetic expressions.[2] For our Translator T we could identify matching partial functions of the semantics of the input and the output languages for each construct of the input language.[3] Given these pairs, we would have a starting ground to prove the correctness of the translation by showing that the corresponding partial functions indeed deliver the same results.

However, to follow that approach would require to establish a formal semantics for each language, be it input or output, we consider for our retargetable translator.

3.2 (Formal) Semantics of Driving Simulator Languages

Considering the task to translate from one driving simulator language to another, one would like to have a formal semantics for each of the involved languages, for example a semantics such as the one sketched above, which maps language constructs to changes of the system state.

In compiler design an abstract machine is typically used to describe the effect of a certain language construct. For these abstract machines there usually is a more or less straightforward way to come up with a mapping to real processor architectures. Moreover, they can constitute a bridge between the programming language paradigm (e. g. of a functional programming language) and the strictly imperative execution style of real hardware.

[2] Note, that the brackets $[\![\ ]\!]$ are traditional in denotational semantics.

[3] In practice this is typically not quite as easy as it might seem. Especially not, if the two languages belong to different families, e. g. an imperative and a functional programming language.

Given an abstract machine, a formal semantics can be formulated in a generic way that is agnostic of the 'dirty' details of the real hardware and the specific flavors of different processor architectures. The relevant properties of the state space of an abstract machine in the programming language domain are for instance the values of its stores (e. g. registers, stack, heap). An interesting question is: What would be the relevant properties of the state space of an abstract driving simulator machine?

Let's consider the description of a road as example. What aspects of the state space of an abstract driving simulator machine would be altered by the road, or expressed otherwise, what is the semantics of the language construct(s) describing the road? Its shape? Its elevation? Its roughness? Even considering the complete geometry of the road as its semantics, ignores many (potentially) important aspects, e. g. the color of the road markings, or the reflection behavior for electromagnetic waves emitted from a radar system.

A more comprehensive approach would be to consider the (vector-based or discrete) geometry of all elements of the road plus their 'exact' relevant physical properties. The problem here is not only the amount of data, or the question how 'exact' is 'exact' enough, and what properties are relevant (this could begin with the geometry of the road network, and continue up to the physical properties regarding reflection of light, or radar signals of simulated elements and beyond). Of course, one could think of a 'flat' and simply adjustable model, where everything is just a number, be it the three-dimensional position and orientation, or an arbitrary physical property. A core problem is that this kind of semantics prevents almost any abstraction. To know how a certain part of the geometry reflects electromagnetic waves of a given spectrum can be very important, but the information whether it is a stop sign or a lane marking should not be dropped in favor of 'exactness'.

From the viewpoint of provability of correct translation, one would like to have a common, detailed mathematical model, e. g. the above sketched three-dimensional space with a list of exact properties for each point in that space. However, what is it worth to have this, unless driving simulator software *A* and driving simulator software *B* react identical and provide the same properties (e. g. color in case a video camera is used as sensor of the DA or ADF system) at its interfaces? From the perspective of developing or discussing a certain driving simulation scenario, one would like to have a semantics following the abstract concepts, such as road sign, lane marking, and so on.

3.3 Related Work

In fact, both lines of thought (abstract concepts and detailed properties) are considered in literature and practice, though to the best of our knowledge no existing work resulted in a universal formal semantics that could be used for the translation between driving simulation languages.

Ulbrich et al. define a *scene* in the context of automated driving as a snapshot of the environment and the self-representations of all actors and observers and the relationship between those [Ul15]. While this is too informal to immediately help regarding a formal semantics, the snapshot concept would suit to the idea of a state space that is changed at discrete events or times. And of course, if we realize a driving simulator based on discrete event simulation, each subsequent simulator state would contain a *scene* representation.[4]

Menzel et al. discuss a hierarchy of scenario representations, distinguishing functional, logical, and concrete scenarios, with the former ones being the most abstract [MBM18]. In their work abstraction is seen as a way to characterize multiple scenarios[5] in one description, for instance by considering ranges of parameters. They also formulate the requirement that a scenario must be reproducible.

Menzel et al. describe a format conversion from a parametrized representation of scenarios to OpenDRIVE, where the relevant parameters of the higher language have been derived from OpenDRIVE [Me19]. They report that a strict evaluation regarding the correctness of their translation was not possible. Instead, they manually inspected (examined) the generated files for potential errors and executed the generated scenarios in a driving simulation environment to visually check whether 3-D graphics were completely displayed and whether lanes had been correctly connected.

Schwab et al. describe an approach to map OpenDRIVE to the 3D city model standard CityGML [SBK20]. They mention that their chosen mapping between language elements is ad-hoc and should not be considered as final. A formal presentation of the translation is not given, nor a formal semantics of the used languages.

To the best of our knowledge, there is up to now no published related work on the formal correctness of the translation of driving simulation languages regarding the road network or other static aspects. While there are published results regarding formalizing and proving correctness regarding dynamic aspects of driving simulation, this is outside the scope of this work.

3.4 Solution Idea

What could help is a kind of universal semantics that orients itself on the abstract meanings of the elements of a road network and allows for arbitrary detailed additional properties. And in fact (and by no means surprising), the syntax of existing driving simulator languages also (at least somewhat) follows this path, by using syntactic elements that are connected to

[4] It might be worth noting that there is no immediate correlation between a language construct (e. g. describing the shape of a road) and an immediately visible state change. When considering static aspects of the simulated world, the related language constructs are similar to declarations and definitions in a programming language, as they have no immediate effect on the output, but are important when evaluated later on.

[5] If not stated otherwise we use the terms *scene* and *scenario* as defined in [Ul15]. Where a scenario is a certain sequence of concrete static scenes.

typical terms humans use to characterize road networks. However, what in one description format might be a semantic concept, is not necessarily found in another (let alone the syntactic differences, which are not of primary concern here). Unfortunately to the best of our knowledge, there is no readily usable formal semantics of driving simulator languages (although OpenDRIVE has some good concepts, that could be building blocks to be used for that purpose).

Instead of striving for a 'universal' and unifying formal semantics we propose a universal mathematical model that allows for the description of a translation semantics, i. e. the state space of the semantics comprises the language constructs. This has two major advantages. First, we can somewhat ignore the mismatches between semantic concepts of different description formats, which would jeopardize an elegant and consistent way to form a unifying semantics. Second, we can ignore the unsolved problem of coming up with a unified abstract machine of driving simulators. Conceptually we can achieve the most that can currently be expected, regarding provability of correctness of translation. This is similar to the problem of verifying correct translation from one high-level programming language to another such as C, without a strictly defined formal semantics of one or both languages.[6] An established approach here is translation validation [PSS98], that follows the concept of proving for each individual translation that the target code correctly implements the source program, and does that by showing that each refinement step from source language to target language has been performed as specified. Our translation semantics approach allows to realize a similar concept.

4 Architecture and Implementation

Our approach uses two main components for the translations between formats. The front-end parses the source file format to construct the intermediate representation, which then is translated via the back-end to the target file format. This well-known concept from the domain of compiler design offers the flexibility to easily adapt to additional or changed source or target languages. For example, adding a new source file format only requires to implement that specific format as front-end. All other components remain unaffected.

Figure 1 shows a diagram of the basic program structure. The source file formats, are shown on the left. Next to it is the command line interface, with which the user can interact. The compilation starts with the lexical and syntactical analyses on the given format. Thereafter the intermediate representation is generated on which a semantic analysis can be applied and subsequent optimizations are possible. The last step is to generate the target file format, which again is format specific. The file formats surrounded with dashed lines and optimizations are not implemented yet.

[6] For instance as reported in [Le16] the semantics of the C programming language standard permits different evaluation orders of expressions.

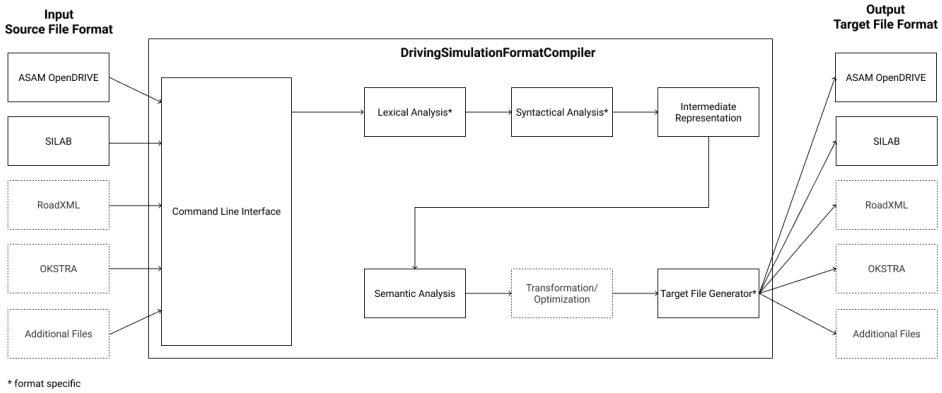

Fig. 1: Basic program structure

ASAM OpenDRIVE and SILAB are both partially implemented in the front- and back-end, which means that it is possible to translate from one to another. To add a file format to the front-end, there are three steps required. First to prepare the translation, second to read the given files and third to construct the intermediate representation with the acquired data. For ASAM OpenDRIVE the ad-xolib[7] XMLParser is utilized within the front-end. In case of SILAB we reengineered the grammar of the supported constructs and developed our own parser. Flex and Bison were used for the scanner and the parser, respectively.

The front-end's result is the intermediate representation (IR), which is an in-memory data structure. This approach is very flexible and can be extended if some required information is missing. Transformations or optimizations can be performed on the IR before it is converted to the target file format in the back-end. The main parts of the IR are *Street*, *StreetType*, *Environment*, *Module* and *Map*. *Street* stores the information specific to one road. *StreetType* can be used by one or many *Streets* and contains data about the lanes. The *Environment* describes the surrounding area next to the road. A *Module* stores the road network and a *Map* can be used to connect *Modules*. These components were chosen to cover different approaches of the individual formats.

In the context of file formats, transformations or optimizations could be a reduction of elements in the formats to reduce file size and complexity. Another possible optimization could be a restructuring of individual elements such that the target files could be parsed more efficiently in the simulation context. The optimization part is not implemented yet.

The final step is the target file generation, where the IR is translated into target language constructs. This is done by iterating over the different components of the IR and piecewise generation of target format elements.

[7] Source: https://github.com/javedulu/ad-xolib

5 Translation Semantics

As motivated above, we introduce a semantics of the translation between driving simulation languages. This encompasses the translation from source to intermediate language and from intermediate to target language. The central piece is the state space of the IR, which is defined as follows.

Let A be a set of *attributes*, where each attribute comprises a set of key-value pairs (k, v), with k constituting a property and v an associated value. Let further be T a set of predefined *tokens* (i. e. the language keywords of the main components of the IR described in Section 4), and S a set of *names*. Then $\Sigma = \mathcal{P}(S \times T \times A)$ constitutes the state space of the IR. Whenever the translator reads a new statement (e. g. a language construct describing an element of a road), its translation state is updated to reflect the effect of the statement.

5.1 Translating Source Format to Intermediate Representation

As it is beyond the scope and available space of this paper to provide a detailed semantics of the languages supported by out translator, we present the following very simple language VSL with no further utility, except for illustrating the concept of the translation semantics. The syntactic set for VSL road constructs is **Con**:

$$c ::= i(n, a) \mid i(n) \mid i(a) \mid c_0; c_1$$

$$i ::= \textbf{road} \mid \textbf{lane}$$

$$a ::= v \mid v, a$$

Where n are names (strings) to identify the individual constructs, and v are key-value pairs as we also use them in the IR (see above), which can provide details such as geometric properties. An example of an element of a specification in VSL for a single-lane street with element named MainRoad with a length of 1000 meters, and a width of 3 meters would look like this.

road (MainRoad, (length, 1000), (width, 3), (lanes, 1))

Consider the translation semantics function O for VSL with $O : \textbf{Con} \to (\Sigma \to \Sigma)$.

A new token is added or an existing token with the same name is modified, if the input is an identifier with name and attribute:

$$O[\![i(name, v_\alpha)]\!] = \begin{cases} \{(\sigma, \sigma') \mid \sigma' = \sigma[(name, t_i, a[\alpha/v_\alpha])]\} & \text{if } (name, t_i, a) \in \sigma \\ \{(\sigma, \sigma') \mid \sigma' = \sigma \cup \{(name, t_i, a^0_{t_i}[\alpha/v_\alpha])\} & \text{otherwise} \end{cases}$$

Where $\sigma[(name, t, a[\alpha/v_\alpha])]$ is short for $\sigma[(name, t, a)/(name, t, a[\alpha/v_\alpha])]$.

Note, that t_i is the token of the intermediate language that matches the identifier i of the source language (in this example we assume a static 1:1 mapping). The default attribute a_t^0 is needed to provide for (yet) missing values, e. g. when the input is a street with a name but incomplete data (either because of implicit rules of the source language or because of postponed definitions). For instance a given **road** identifier could come without a value for the length of the road, and the matching value for the IR could be set to zero in this case.

Applying $O[\![\,]\!]$ to our example MainRoad from above would yield $street$ as token in the IR and more or less identical key-value pairs as given in the input example.

If there is more than one attribute in the input, the suitable functions are applied in order[8]:

$$O[\![i(name, v_{\alpha_1}, ..., v_{\alpha_n})]\!] = O[\![i(name, v_{\alpha_2}, ..., v_{\alpha_n})]\!] \circ O[\![i(name, v_{\alpha_1})]\!]$$

If the input is a named identifier i without attribute (e. g. a new road element in the input description without details yet) the matching token is added with name and default attributes $a_{t_i}^0$:

$$O[\![i(name)]\!] = O[\![i(name, v_{a_{t_i}[1]}, ..., v_{a_{t_i}[n]})]\!]$$

Where $v_{a_{t_i}[1]}, ..., v_{a_{t_i}[n]}$ are the default values from $a_{t_i}^0$.

If the input is unnamed with an attribute, the matching token is added with a new name and the given attribute:

$$O[\![i(v_\alpha)]\!] = O[\![i(name', v_\alpha)]\!]$$

Where $name'$ is a new name.

5.2 Translating Intermediate Representation to Target Format

We briefly present the principle approach to describe the translation semantics function from our IR to SILAB and to ASAM OpenDRIVE, respectively.

5.2.1 SILAB

Let B be the set of all SILAB track components and P the set of all attributes of these components. The state space can be described with $\Theta = \mathcal{P}(S \times B \times P)$. To define the translation semantics function from the IR to SILAB, a helper function $sil : S \times T \times A \rightarrow S \times B \times P$ is used, so that sil provides the specific SILAB syntax for each tuple in a concrete state σ of the IR to be processed. The translation semantics function O_{sil} is similar to O in Subsection 5.1, with $O_{sil} : \Sigma \rightarrow (\Theta \rightarrow \Theta)$.[9]

[8] As composition function we use $(g \circ f)(x) = g(f(x))$

[9] We do not include the serialization of the target language constructs to an output file, as we consider this as separate subsequent step. However, this can be accounted for either as another semantics function or by adopting O_{sil} accordingly.

For any $\sigma \in \Sigma$ with $\sigma = \{(s_1, t_1, a_1), ..., (s_n, t_n, a_n)\} \wedge n \geq 1$ we can define O_{sil} as:

$$O_{sil}[\![\sigma]\!] = \begin{cases} O_{sil}[\![(s_2, t_2, a_2), ..., (s_n, t_n, a_n)]\!] \circ O_{sil}[\![(s_1, t_1, a_1)]\!] & \text{if } |\sigma| > 1 \\ O_{sil}[\![(s, t, a)]\!] = \{(\theta, \theta') \mid \theta' = \theta \cup \{sil(s, t, a)\}\} & \text{otherwise} \end{cases}$$

5.2.2 ASAM OpenDRIVE

Let E be the set of all ASAM OpenDRIVE XML-Elements and Q the set of all XML-Attributes. The state space can be described with $\Gamma = \mathcal{P}(S \times E \times Q)$. Similar to the case for SILAB we use a helper function $od : S \times T \times A \rightarrow \Gamma$, that provides the specific XML-Elements of OpenDRIVE for a given tuple from the IR state. $O_{od} : \Sigma \rightarrow (\Gamma \rightarrow \Gamma)$ is the translation semantics function for OpenDRIVE as target language.

For any $\sigma \in \Sigma$ with $\sigma = \{(s_1, t_1, a_1), ..., (s_n, t_n, a_n)\} \wedge n \geq 1$ we can define O_{od} as:

$$O_{od}[\![\sigma]\!] = \begin{cases} O_{od}[\![(s_2, t_2, a_2), ..., (s_n, t_n, a_n)]\!] \circ O_{od}[\![(s_1, t_1, a_1)]\!] & \text{if } |\sigma| > 1 \\ O_{od}[\![(s, t, a)]\!] = \{(\gamma, \gamma') \mid \gamma' = \gamma \cup \{od(s, t, a)\}\} & \text{otherwise} \end{cases}$$

6 Conclusion and Further Work

We stated that the field of driving simulation languages is of rising importance for software engineering in the automotive domain. Not only because driving simulation is an important tool for the software development itself, but also because future type approvals of automated driving functions without simulation results seem to be impossible, as a formal verification of all involved software, especially of artificial neuronal networks is (yet) out of reach. Nevertheless, there are things that can be verified, and should be formally proven as far as possible for the sake of safety and our profession. This holds for instance for tool chains used to develop safety-related functionality.

As a small contribution to this interesting field, we presented a semantics based retargetable driving simulation language translator, together with some thoughts about the relation between formal semantics of programming languages and driving simulation semantics. Additionally, we described the underlying translation semantics, which we developed to form an initial stepping stone to prove the correctness of translations from one driving simulation language to another.

Future work will be to develop a tool that allows to prove the correctness of the translations per instance in a manner similar to the approach taken in the area of programming languages following the concepts of [PSS98] and more recent work in that field.

References

[10] RoadXML Website, July 2010, URL: https://www.road-xml.org/index.php, visited on: 11/30/2021.

[21] Objektkatalog für das Straßen- und Verkehrswesen Website, Nov. 2021, URL: https://www.okstra.de/.

[As21] Association for Standardization of Automation and Measuring Systems: ASAM OpenDRIVE V1.7.0 User Guide, version 1.7.0, Aug. 2021, URL: https://www.asam.net/index.php?eID=dumpFile&t=f&f=4422&token=e590561f3c39aa2260e5442e29e93f6693d1cccd.

[Le16] Leroy, X.; Blazy, S.; Kästner, D.; Schommer, B.; Pister, M.; Ferdinand, C.: CompCert - A Formally Verified Optimizing Compiler. In: ERTS 2016: Embedded Real Time Software and Systems. Jan. 2016.

[MBM18] Menzel, T.; Bagschik, G.; Maurer, M.: Scenarios for Development, Test and Validation of Automated Vehicles. In: 2018 IEEE Intelligent Vehicles Symposium (IV). IEEE, June 2018.

[Me19] Menzel, T.; Bagschik, G.; Isensee, L.; Schomburg, A.; Maurer, M.: From Functional to Logical Scenarios: Detailing a Keyword-Based Scenario Description for Execution in a Simulation Environment. In: 2019 IEEE Intelligent Vehicles Symposium (IV). Pp. 2383–2390, June 2019.

[PSS98] Pnueli, A.; Siegel, M.; Singerman, E.: Translation Validation. In: Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, pp. 151–166, Mar. 1998, ISBN: 978-3-540-64356-2.

[SBK20] Schwab, B.; Beil, C.; Kolbe, T. H.: Spatio-Semantic Road Space Modeling for Vehicle–Pedestrian Simulation to Test Automated Driving Systems. Sustainability 12/9, 2020, ISSN: 2071-1050, URL: https://www.mdpi.com/2071-1050/12/9/3799.

[Ul15] Ulbrich, S.; Menzel, T.; Reschka, A.; Schuldt, F.; Maurer, M.: Defining and Substantiating the Terms Scene, Situation, and Scenario for Automated Driving. In: 2015 IEEE 18th International Conference on Intelligent Transportation Systems. IEEE, Sept. 2015.

[Wü] Würzburger Institut für Verkehrswissenschaften: SILAB Documentations, version SILAB 5, Würzburger Institut für Verkehrswissenschaften GmbH.