Map Verification and Repairing Using Formalized Map Specifications

Sebastian Maierhofer*, Yannick Ballnath*, and Matthias Althoff

Abstract— **Autonomous vehicles benefit from correct maps to participate in traffic safely, but often maps are not verified before their usage. We address this problem by providing an approach to verify and repair maps automatically based on a formalization of map specifications in higher-order logic. Unlike previous work, we provide a collection of map specifications. We can verify and repair all possible map parts, from geometric to semantic elements, e.g., adjacency relationships, lane types, road boundaries, traffic signs, and intersections. Due to the modular design of our approach, one can integrate additional logics. We compare ontologies, answer set programming, and satisfiability modulo theories with our higher-order logic verification algorithm. Our evaluation shows that our approach can efficiently verify and repair maps from several data sources and of different map sizes. We provide our tool as part of the CommonRoad Scenario Designer toolbox available at commonroad.in.tum.de.**

I. Introduction

Maps used for autonomous driving are usually created from sensor data (e.g., vehicle cameras or satellite images), which can be noisy, imprecise, or incomplete. These measurement errors are inevitably propagated in various processing steps and create errors in the generated maps if no verification and repairing methods are applied. Possible errors are intersecting lane boundaries, incompatible lane types, or wrong adjacency relationships. Even if the created maps are error-free, processing the maps can introduce errors, e.g., if the map is converted into another format or a part of a large map is extracted. It is essential that the overall map toolchain complies with quality standards and produces the expected results. Since maps are usually provided by external data sources, verifying maps based on formal methods is often necessary. If a map verification finds errors, it is desirable that these are repaired automatically.

Many autonomous vehicles rely on map data, e.g., to obtain correct results from motion planning algorithms. For example, if there is a small gap in the road network, a planned motion could be valid, although it leaves the road. Another example requiring correct maps is the evaluation of traffic rule compliance [1], [2].

A. Related Work

Several map formats are used for autonomous driving in research and industry. There exist map formats more suited for simulations like OpenDRIVE[1] and for the application in an automated vehicle like lanelet-based formats [3]–[5] or NDS[2]. An overview of high-definition map elements, formats, and applications for automated driving is provided in [6]. Maps can be generated by vehicle sensors [7]–[9] or aerial images [10], [11]. Open-source tools to create maps manually and on a large scale are introduced in [12] and [13], where the focus of the tools is the generation of test scenarios for motion planning algorithms.

In map validation, the map obtained through sensors is validated against a given map [14]–[16]. This is important to detect whether an autonomous vehicle drives within its operational design domain. In [17], the change of a map over several years is evaluated, which shows the necessity of map validation. The authors of [18] introduce terminology and metrics for map deviations. In map verification, the map is verified for consistency, i.e., whether the map fulfills a specification. Note that map verification can also be applied within a vehicle, e.g., if the map is provided during operation by an external supplier and cannot be verified beforehand.

Subsequently, we focus on map verification. The authors of [19] use answer set programming to model simple topological road networks. Their approach can also repair invalid road networks, e.g., wrongly placed speed limit signs. OpenDRIVE maps are verified in [20], but the approach only evaluates the existence of gaps between succeeding lanes and successor/predecessor relationships. Ontologies are also often used to model road networks [21]–[23]. Whereas in [21] and [22], the focus is on reasoning about complete traffic scenes, including other traffic participants, the focus in [23] is on the verification and repairing of maps using ontologies. The approach in [23] consists of semantic enrichment, violation detection, and violation handling but only covers non-critical violations, e.g., attribute and topological errors. The authors of [24] use description logic, which is often combined with ontologies, to perform reasoning about road networks.

In summary, there exist only a few approaches to verify maps using logic, and those cover only a small set of available map elements. The used logics are mainly answer set programming and ontologies requiring additional reasoning tools.

B. Contributions

This paper presents an approach for verifying and repairing maps based on formalized map specifications. Compared to existing work, we provide the following contributions:

*The first two authors have contributed equally to this work.

All authors are with the School of Computation, Information and Technology, Technical University of Munich, 85748 Garching, Germany.
{sebastian.maierhofer, yannick.ballnath, althoff}@tum.de

[1]https://www.asam.net/standards/detail/opendrive/
[2]https://nds-association.org

1) map specifications formalized in higher-order logic covering all map elements;
2) a modular map verification and repairing approach;
3) evaluation of different logics for the verification of maps;
4) we verify and repair more than 600 publicly-available maps; and
5) our framework is provided as an open-source tool compatible with different map formats.

The remainder of this paper is structured as follows: In Sec. II, we introduce the preliminaries and our overall approach. Afterward, we formalize the road network in Sec. III. Map verification and repairing are introduced in Sec. IV and Sec. V. We evaluate our approach in Sec. VI and conclude the paper in Sec. VII.

II. Preliminaries and Overall Approach

Subsequently, we introduce terminology, elements used throughout the paper, and our overall approach.

A. Notation and Preliminaries

Let $\mathcal{M}$, $\mathcal{S}$, $\mathcal{F}$, and $\mathcal{E}$ be the set of map elements, map specifications, repairing functions, and map errors, respectively. We combine specifications into groups $g \in \mathcal{G}$, where $\mathcal{G}$ is a priority queue of all groups. Each group g is described by a tuple $g := \langle \mathcal{S}_\mathrm{g}, \mathcal{F}_\mathrm{g} \rangle$, where $\mathcal{S}_\mathrm{g} \subseteq \mathcal{S}$ and $\mathcal{F}_\mathrm{g} \subseteq \mathcal{F}$ are the specifications and repairing functions of group g. The priority queue $\mathcal{G}$ must be designed so that repairing does not influence map elements of higher priority groups, e.g., repairing an invalid polyline could influence adjacency relationships. We describe an error $e \in \mathcal{E}$ by a tuple $e := \langle s, \mathcal{M}_\mathrm{e} \rangle$, where $s \in \mathcal{S}$ is the violated specification and $\mathcal{M}_\mathrm{e} \subseteq \mathcal{M}$ are the invalid map elements.

B. Overall Approach

To create a specification-compliant map from an invalid map, we propose an iterative process consisting of map verification and repairing cycles. Our approach is summarized in Alg. 1. The algorithm receives a map m and the specification groups $\mathcal{G}$ as inputs. As long as the priority queue $\mathcal{G}$ is not empty, the group with the highest priority is extracted and the provided map is verified for the specifications associated with the group (cf. lines 2 - 4). For each error, the repairing procedure is executed with the relevant repairing functions belonging to the specification group (cf. line 6). The repaired map and the errors are provided to the user in the end so that the user can solve the root cause of an error if required (cf. line 10). We also support partitioning very large maps into smaller ones for parallel verification and repairing of partial maps. For simplicity, we neglect this in the presented algorithm.

Algorithm 1 Verification and Repairing Process

```
Input: map $m$, specification groups $\mathcal{G}$
Output: repaired map $m$, list of errors $\mathcal{E}$
 1: $\mathcal{E} \leftarrow \emptyset$
 2: while $|\mathcal{G}| > 0$ do
 3:     $\langle \mathcal{S}_\mathrm{g}, \mathcal{F}_\mathrm{g} \rangle \leftarrow \mathcal{G}.\textsc{pop}$
 4:     $\mathcal{E}' \leftarrow \textsc{verify}(m, \mathcal{S}_\mathrm{g})$                ▷ Sec. IV
 5:     for each $e \in \mathcal{E}'$ do
 6:         $m \leftarrow \textsc{repair}(e, m, \mathcal{F}_\mathrm{g})$            ▷ Sec. V
 7:     end for
 8:     $\mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{E}'$
 9: end while
10: return $m, \mathcal{E}$
```

III. Road Network

We base our road network specification on the CommonRoad map format. However, our approach and formalization can be adapted to other map formats and several other formats can be converted to our representation [13]. Subsequently, we introduce relevant elements of our road network. Fig. 1 shows an intersection with different road network elements. The road network elements are also summarized in Tab. I. We use the notation $\widehat{\square}$ for functions accessing the variable $\square$ of a road network element, e.g., the id of an element l can be accessed via $\widehat{\mathrm{id}}(l)$.

A. Geometric Elements

The fundamental elements within our map representation are vertices. A vertex is a vector $v = [x, y]^T \in \mathbb{R}^2$ representing a point in the Cartesian coordinate system. We restrict our formalization to the two-dimensional space since that is mainly used for motion planning of autonomous vehicles. However, our approach can easily be extended to the three-dimensional space. We use vertices to model polylines, where a polyline is a finite sequence of vertices $(v_0, ..., v_i, ...)$, $i \in \mathbb{N}_{\geq 0}, v_i \in \mathbb{R}^2$. The set of polylines is denoted as $\mathcal{P}$.

B. Lanelet and Area

The sets $\mathcal{L}$, $\mathcal{AR}$, $\mathcal{Y}$, and $\mathcal{LM}$ denote the set of lanelets, areas, lanelet types (e.g., $urban, interstate, rural, access\_ramp$), and line markings (e.g., $dashed, solid$), where $\mathcal{L}, \mathcal{AR} \subset \mathcal{M}$. A lanelet is a small drivable road segment defined by two polylines [3] and is specified by the elements introduced in Tab. I. If a element is not present, it is set to $\emptyset$.

To model drivable spaces which cannot be represented by lanelets, we use areas [5, Sec. IV.A], whose elements are introduced in Tab. I. Fig. 1 shows an area that is specified by four boundaries and fills the gap between several lanelets.

C. Traffic Sign, Traffic Light, and Stop Line

The sets $\mathcal{TS}$, $\mathcal{TSE}$, $\mathcal{TST}$, $\mathcal{TL}$, $\mathcal{TLD}$, $\mathcal{TLC}$, $\mathcal{U}$, and $\mathcal{SL}$ denote the set of traffic signs, traffic sign elements, traffic sign types (e.g., MAX_SPEED or U_TURN), traffic lights, traffic light directions, traffic light colors, Unicode characters, and stop lines, where $\mathcal{TS}, \mathcal{TL}, \mathcal{SL} \subset \mathcal{M}$. The elements of traffic signs, traffic lights, and stop lines are listed in Tab. I. A single traffic sign can be composed of several elements and their associated values, e.g., a speed

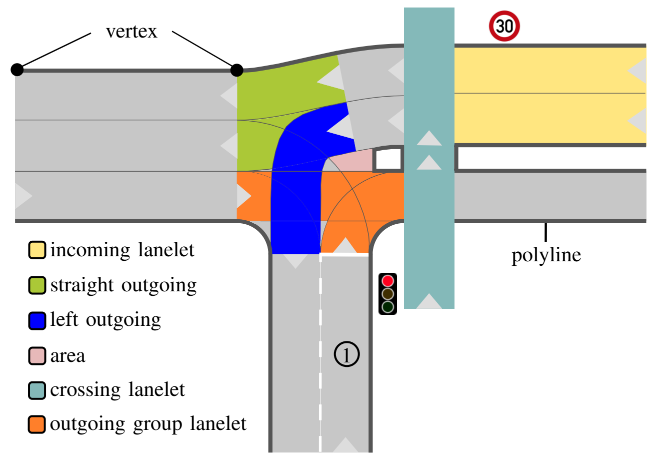


Fig. 1: A three-way intersection showing different CommonRoad elements. Only a single incoming/outgoing group is highlighted. For simplicity, the other two incoming/outgoing groups are not visualized. The crossing lanelets form a crossing group referencing the visualized incoming/outgoing group. Lanelet ① has a dashed thin line marking, references a traffic light, has a stop line, and its left adjacent lanelet has the opposite driving direction. The yellow lanelets reference a traffic sign.

limit and a stop sign. The traffic light cycle specification, i.e., the switching times, is not part of a traffic light since it is runtime-specific.

D. Intersection

An intersection consists of incoming, outgoing, and crossing groups, where the different elements are defined in Tab. I. The set of intersections, incoming groups, outgoing groups, and crossing groups are denoted as $\mathcal{I}$, $\mathcal{IG}$, $\mathcal{OG}$, and $\mathcal{CG}$, where $\mathcal{I}, \mathcal{IG}, \mathcal{OG}, \mathcal{CG} \subset \mathcal{M}$. The different intersection elements are also shown in Fig. 1.

TABLE I: Description of map elements.

Elements	Description
Lanelet	
$id \in \mathbb{N}_{\geq 0}$	ID of lanelet
$lb, rb \in \mathcal{P}$	polylines for left and right boundaries of lanelet
$adj_{\mathrm{l}}, adj_{\mathrm{r}} \in \mathbb{N}_{\geq 0}$	ID of left and right adjacent lanelet
$dir_{\mathrm{l}}, dir_{\mathrm{r}} \in \mathbb{B}$	Boolean value indicating whether left and right adjacent lanelets have same driving direction
$lm_{\mathrm{l}}, lm_{\mathrm{r}} \in \mathcal{LM}$	line marking type of left and right boundary
$suc, pre \subset \mathbb{N}_{\geq 0}$	IDs of successor and predecessor lanelets
$ts, tl \subset \mathbb{N}_{\geq 0}$	IDs of referenced traffic signs and lights
$ty \subset \mathcal{Y}$	lanelet types
$sl \in \mathbb{N}_{\geq 0}$	ID of stop line
$ar \subset \mathbb{N}_{\geq 0}$	IDs of adjacent areas
Stop Line	
$id \in \mathbb{N}_{\geq 0}$	ID of stop line
$p_1 \in \mathbb{R}^2$	start point of stop line
$p_2 \in \mathbb{R}^2$	end point of stop line
$lm \in \mathcal{LM}$	line marking type of stop line
Area	
$id \in \mathbb{N}_{\geq 0}$	ID of area
$b \subset \mathcal{P}$	polylines representing boundary
$adj \subset \mathbb{N}_{\geq 0}$	IDs of adjacent lanelets and areas
$lm \subset \mathcal{LM}$	line marking types
$ty \subset \mathcal{Y}$	types of area
Traffic Sign	
$id \in \mathbb{N}_{\geq 0}$	ID of traffic sign
$pos \in \mathbb{R}^2$	position of traffic sign
$el \subset \mathcal{TSE}$	traffic sign elements belonging to traffic sign
Traffic Sign Element	
$st \in \mathcal{TST}$	traffic sign type
$val \in (\mathbb{Z} \setminus 0) \times \mathcal{U}$	value of traffic sign, e.g., concrete speed limit
Traffic Light	
$id \in \mathbb{N}_{\geq 0}$	ID of traffic light
$pos \in \mathbb{R}^2$	position of traffic light
$lm \in \mathcal{LM}$	line marking type
$dir \in \mathcal{TLD}$	traffic light direction
$col \subset \mathcal{TLC}$	traffic light colors
Intersection	
$id \in \mathbb{N}_{\geq 0}$	ID of intersection
$incg \subset \mathcal{IG}$	incoming groups of intersection
$outg \subset \mathcal{OG}$	outgoing groups of intersection
$cros \subset \mathcal{CG}$	crossings of intersection
Incoming Group	
$id \in \mathbb{N}_{\geq 0}$	ID of incoming group
$incl \subset \mathbb{N}_{\geq 0}$	IDs of incoming lanelets
$rg \subset \mathbb{N}_{\geq 0}$	IDs of right outgoing lanelets
$lg \subset \mathbb{N}_{\geq 0}$	IDs of left outgoing lanelets
$sg \subset \mathbb{N}_{\geq 0}$	IDs of straight outgoing lanelets
$outg \in \mathbb{N}_{\geq 0}$	ID of outgoing group belonging to incoming group
Outgoing Group	
$id \in \mathbb{N}_{\geq 0}$	ID of outgoing group
$outgs \subset \mathbb{N}_{\geq 0}$	IDs of lanelets belonging to outgoing group
Crossing Group	
$id \in \mathbb{N}_{\geq 0}$	ID of crossing
$cros \subset \mathbb{N}_{\geq 0}$	IDs of lanelets belonging to crossing
$incg \in \mathbb{N}_{\geq 0}$	ID of crossed incoming group
$outg \in \mathbb{N}_{\geq 0}$	ID of crossed outgoing group

E. Map Specifications

The map specifications are formalized using higher-order logic (HOL). We use the logical operators $\vee$, $\neg$, as well as the quantifiers $\forall$ and $\exists$. The implication $a \implies b$ and the and-operator $a \wedge b$ are syntactic sugar and can be rewritten as $\neg a \vee b$ and $\neg(\neg a \vee \neg b)$, where a and b are Boolean propositions or predicates. The cardinality of a set is denoted as $|\cdot|$. Due to space limitations, we only present a subset of our formalized specifications. A complete list can be found in our CommonRoad format definition under commonroad.in.tum.de. For more fine-grained repairing feedback, to repair the correct elements, we intentionally separate specifications that could be combined (cf. Sec. III-E.2 and Sec. III-E.3).

1) Unique IDs (UI): All road elements must have a unique ID:

$$\neg\exists k_1, k_2 \in \mathcal{M} : k_1 \neq k_2 \wedge \widehat{\mathrm{id}}(k_1) = \widehat{\mathrm{id}}(k_2). \tag{1}$$

2) Successor Reference (SR): If a lanelet references a successor lanelet, the ID of the lanelet must correspond to another existing lanelet:

$$\begin{aligned} &\forall l_1 \in \mathcal{L} : \forall l_2^{\mathrm{id}} \in \widehat{\mathrm{suc}}(l_1) : \\ &\quad \exists l_2 \in \mathcal{L} : l_1 \neq l_2 \wedge \widehat{\mathrm{id}}(l_2) = l_2^{\mathrm{id}}. \end{aligned} \tag{2}$$

3) Successor Connection (SC): If two lanelets have a successor/predecessor relationship, their initial and final vertices

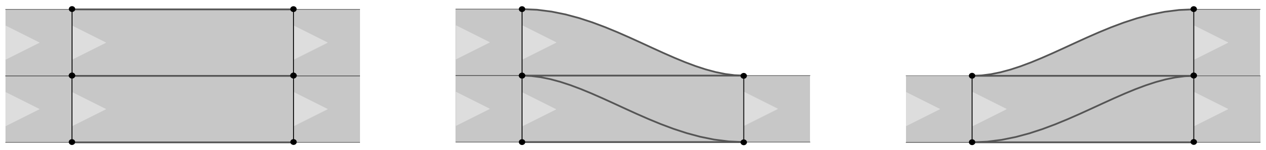

(a) Parallel adjacency. (b) Merging adjacency. (c) Forking adjacency.

Fig. 2: Different adjacencies considered for the map specification and repairing.

must be equal:

$$\forall l_1, l_2 \in \mathcal{L} : \widehat{\mathrm{id}}(l_2) \in \widehat{\mathrm{suc}}(l_1) \implies \mathrm{l}_{\mathrm{fin}}(l_1) = \mathrm{l}_{\mathrm{ini}}(l_2), \tag{3}$$

where the functions $\mathrm{l}_{\mathrm{ini}}(l)$ and $\mathrm{l}_{\mathrm{fin}}(l)$ return the initial and final vertices of a lanelet $l \in \mathcal{L}$. The equality of vertices is determined by an element-wise comparison.

4) Potential Successor (PS): If the final vertices of a lanelet and the initial vertices of another lanelet are equal, they must have a successor relationship:

$$\forall l_1, l_2 \in \mathcal{L} : \mathrm{l}_{\mathrm{fin}}(l_1) = \mathrm{l}_{\mathrm{ini}}(l_2) \implies \widehat{\mathrm{id}}(l_2) \in \widehat{\mathrm{suc}}(l_1). \tag{4}$$

Specifications SC and PS are separated to provide detailed errors for repairing, e.g., if specification SC is violated, vertices must be repaired, whereas if specification PS is violated, lanelet references must be repaired.

5) Adjacent Left Reference (ALR): If a lanelet has a left adjacent reference, the ID of the lanelet must correspond to another existing lanelet:

$$\begin{aligned} \forall l_1 \in \mathcal{L} : \widehat{\mathrm{adj}}_{\mathrm{l}}(l_1) \neq \emptyset \\ \implies \exists l_2 \in \mathcal{L} : l_1 \neq l_2 \wedge \widehat{\mathrm{id}}(l_2) = \widehat{\mathrm{adj}}_{\mathrm{l}}(l_1). \end{aligned} \tag{5}$$

6) Adjacency to Left Parallel Lanelet in Same Driving Direction (ALPS): Adjacent lanelets must share the adjacency over their complete length. For more precise repairing, we distinguish between parallel, merging, and forking adjacency (cf. Fig. 2). Left parallel adjacent lanelets must share the same boundary:

$$\begin{aligned} \forall l_1, l_2 \in \mathcal{L} : \big(l_1 \neq l_2 \wedge \widehat{\mathrm{adj}}_{\mathrm{l}}(l_1) = \widehat{\mathrm{id}}(l_2) \wedge \widehat{\mathrm{dir}}_{\mathrm{l}}(l_1) \\ \wedge\, \mathrm{adj\_type}(l_1, l_2, parallel)\big) \implies \widehat{\mathrm{lb}}(l_1) = \widehat{\mathrm{rb}}(l_2), \end{aligned} \tag{6}$$

where $\mathrm{adj\_type}(l_1, l_2, type)$ evaluates whether the two lanelets l_1 and l_2 fulfill the requirements for the adjacency defined by $type \in \{parallel, merge, fork\}$. The equality of two polylines is determined by the element-wise comparison of the vertices.

7) Left Merging Adjacency (LMA): If a lanelet references a left merging adjacent lanelet, a) the final vertices of both lanelets must be equal, b) the left initial vertex of the lanelet and the right initial vertex of the merging lanelet must be equal, and c) the right boundary of the merging lanelet must be contained by the lanelet:

$$\begin{aligned} \forall l_1, l_2 \in \mathcal{L} : \big(\widehat{\mathrm{adj}}_{\mathrm{l}}(l_1) = \widehat{\mathrm{id}}(l_2) \\ \wedge\, \mathrm{adj\_type}(l_1, l_2, merge)\big) \implies \big(\mathrm{l}_{\mathrm{fin}}(l_1) = \mathrm{l}_{\mathrm{fin}}(l_2) \\ \wedge\, \mathrm{l}^{\mathrm{l}}_{\mathrm{ini}}(l_1) = \mathrm{l}^{\mathrm{r}}_{\mathrm{ini}}(l_2) \wedge \mathrm{contains}(l_1, \widehat{\mathrm{rb}}(l_2))\big), \end{aligned} \tag{7}$$

where $\mathrm{contains}(l, p)$ checks whether the polyline $p \in \mathcal{P}$ is completely contained by the lanelet $l \in \mathcal{L}$, and $\mathrm{l}^{\mathrm{l}}_{\mathrm{ini}}(l)$ and $\mathrm{l}^{\mathrm{r}}_{\mathrm{ini}}(l)$ return the left and right initial vertex of lanelet l. Forking adjacency can be defined analogously.

8) Area Adjacency (AA): An area must have at least one adjacent lanelet or area:

$$\begin{aligned} \forall a \in \mathcal{AR} : |\widehat{\mathrm{adj}}(a)| \geq 1 \\ \wedge\, \forall k \in \widehat{\mathrm{adj}}(a) : \exists e \in (\mathcal{L} \cup \mathcal{AR}) : \widehat{\mathrm{id}}(e) = k. \end{aligned} \tag{8}$$

9) Polylines Intersection (PI): The left and right boundary of lanelets must not intersect:

$$\forall l \in \mathcal{L} : \neg \mathrm{polyline\_intersection}(\widehat{\mathrm{lb}}(l), \widehat{\mathrm{rb}}(l)), \tag{9}$$

where $\mathrm{polyline\_intersection}(lb, rb)$ evaluates whether the two polylines $lb, rb \in \mathcal{P}$ intersect.

10) Exclusive Lanelet Types (ELT): Certain lanelet types are exclusive, e.g., a lanelet must not be of type $urban$ and $rural$:

$$\forall l \in \mathcal{L} : \neg(urban \in \widehat{\mathrm{ty}}(l) \wedge rural \in \widehat{\mathrm{ty}}(l)). \tag{10}$$

11) Distance between Lanelet and Traffic Sign (DLTS): A traffic sign must be referenced by at least one lanelet. The distance between the lanelet and the traffic sign must be smaller than d_{ts}, where $d_{\mathrm{ts}} \in \mathbb{R}_{\geq 0}$ is a threshold specifying the maximum distance:

$$\begin{aligned} \forall s \in \mathcal{TS} : \exists l \in \mathcal{L} : \widehat{\mathrm{id}}(s) \in \widehat{\mathrm{ts}}(l) \\ \wedge\, \exists p \in (\widehat{\mathrm{lb}}(l) \cup \widehat{\mathrm{rb}}(l)) : \|\widehat{\mathrm{pos}}(s) - p\|_2 < d_{\mathrm{ts}}. \end{aligned} \tag{11}$$

12) Incoming and Outgoing Adjacency (IOA): A lanelet part of an outgoing group must not be adjacent to a lanelet part of an incoming group:

$$\begin{aligned} &\forall i_{\mathrm{i}} \in \mathcal{I} : \forall i_{\mathrm{inc}} \in \widehat{\mathrm{incg}}(i_{\mathrm{i}}) : \\ &\quad \forall i_{\mathrm{out}} \in \widehat{\mathrm{outg}}(i_{\mathrm{i}}) : \forall l^{\mathrm{id}}_1 \in \widehat{\mathrm{incl}}(i_{\mathrm{inc}}) : \\ &\qquad \forall l^{\mathrm{id}}_2 \in \mathrm{adj\_lanelets}(l^{\mathrm{id}}_1, \mathcal{L}) : l^{\mathrm{id}}_2 \notin \widehat{\mathrm{outgs}}(i_{\mathrm{out}}), \end{aligned} \tag{12}$$

where $\mathrm{adj\_lanelets}(l_{\mathrm{id}}, \mathcal{L})$ [1, Sec. IV.A] returns recursively the IDs of all adjacent lanelets of the lanelet with ID $l_{\mathrm{id}} \in \mathbb{N}_{\geq 0}$ until a lanelet has no adjacent lanelet.

13) Number of Incoming Groups (NIG): An intersection must consist of at least two incoming groups or one incoming group and a crossing:

$$\begin{aligned} \forall i_{\mathrm{i}} \in \mathcal{I} : \big(|\widehat{\mathrm{incg}}(i_{\mathrm{i}})| > 0 \wedge \exists i_{\mathrm{inc}} \in \widehat{\mathrm{incg}}(i_{\mathrm{i}}) : \\ \exists c \in \widehat{\mathrm{cros}}(i_{\mathrm{i}}) : \widehat{\mathrm{incg}}(c) = \widehat{\mathrm{id}}(i_{\mathrm{inc}})\big) \vee |\widehat{\mathrm{incg}}(i_{\mathrm{i}})| > 1. \end{aligned} \tag{13}$$

14) Left Outgoing Orientation (LOO): The orientation between a left outgoing lanelet and the corresponding incoming lanelets must be in $]\alpha_1, \alpha_2[$:

$$\begin{aligned} &\forall i_{\mathrm{i}} \in \mathcal{I} : \forall i_{\mathrm{inc}} \in \widehat{\mathrm{incg}}(i_{\mathrm{i}}) : \forall l_{\mathrm{id}} \in \widehat{\mathrm{lg}}(i_{\mathrm{inc}}) : \forall l \in \mathcal{L} : \\ &\quad \widehat{\mathrm{id}}(l) = l_{\mathrm{id}} \implies \alpha_1 < \mathrm{angle}(\widehat{\mathrm{incl}}(i_{\mathrm{inc}}), l, \mathcal{L}) < \alpha_2, \end{aligned} \tag{14}$$

where α_1 and α_2 define the allowed orientation interval and $\mathrm{angle}(il, l, \mathcal{L})$ computes the angle between incoming lanelets $il \subset \mathbb{N}_{\geq 0}$ and a lanelet $l \in \mathcal{L}$. We assume that the final vertices of incoming lanelets belonging to an incoming group are connected and the end of the incoming lanelets have a similar orientation.

IV. Map Verification

Our approach can work with several logics (cf. Sec. VI-B). However, the map verification must be able to provide detailed feedback about errors and a direct consideration of our HOL specifications is preferable. Therefore, a HOL inference algorithm is most suitable. The HOL verification algorithm must support lazy definitions of sets and finite domains (some inference tools assume only infinite domains requiring a workaround). To support all of the mentioned requirements, we use a custom HOL formula evaluator, which we introduce informally. It is based on an extended version of propositionalization, a standard approach in first-order logic inference. However, instead of directly instantiating all possible value combinations of the variables, we iteratively instantiate the variables. In general, the evaluation consists of two main steps, which are similar to the runtime verification monitor Hydra [25], but instead of temporal operators, we consider quantifiers over finite domains.

The first step is to execute the subformulas recursively in the expression tree of the HOL formula from the root node of the expression tree to the bottom. For example, in the exemplary expression tree in Fig. 3, the universal quantifier would first call the upper and-operator to collect its evaluation result. Afterward, this and-operator would call its two child nodes, and those would call their child nodes. Quantifiers instantiate the variables of the domain and iteratively execute the child nodes by providing the instantiated variables, e.g., the universal quantifier in Fig. 3 would iterate over all possible combinations of lanelets for x and y and execute the subformula.

The second step is to evaluate the operator of each node. The logical operators $\wedge$, $\vee$, and $\neg$ collect the results of the sub-elements and perform the corresponding logical operation. Analogously, the comparison operators, e.g., $<$, $>$, and $=$, perform the corresponding operation. We support constants and Boolean predicates (e.g., is_a(), is_b(), is_c(), is_d() in Fig. 3), where the latter have either constants, functions, or variables from quantified domains as parameters. The existential quantifier iterates over all variables and checks whether the subformula is satisfied once. The universal quantifier behaves similarly, except that all subformulas must be satisfied. The quantifiers evaluate all possible value combinations to find all existing specification violations and store relevant variable instantiations, e.g., all that lead to a violation of a universal quantifier. Based on the specification and the stored variable instantiations, the verification algorithm creates error elements used as input for the repairing functions.

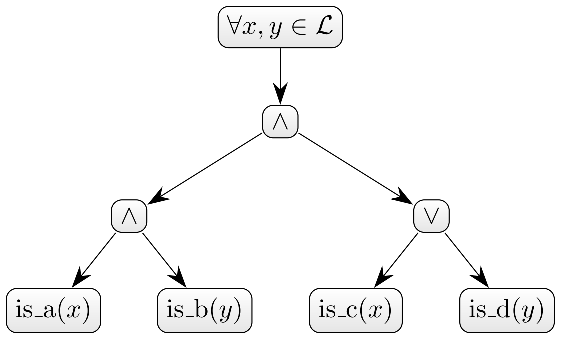
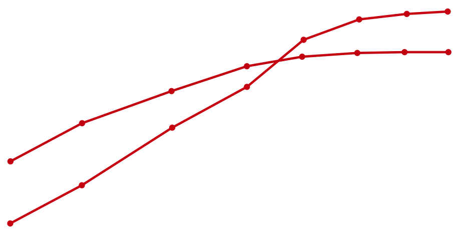
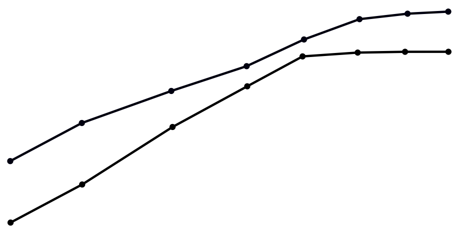


Fig. 3: Expression tree of the exemplary formula $\forall x, y \in \mathcal{L} : (\mathrm{is\_a}(x) \wedge \mathrm{is\_b}(y)) \wedge (\mathrm{is\_c}(x) \vee \mathrm{is\_d}(y))$.



(a) Intersecting polylines.



(b) Repaired polylines.

Fig. 4: Exemplary invalid and repaired boundary polylines of a lanelet.

V. Map Repairing

For each specification, we define a corresponding repairing function. Subsequently, we informally present selected repairing functions for map specifications from Sec. III, where the remaining repairing functions are described in Tab. II:

- *UI*: The elements with a duplicated ID are iteratively assigned a new ID corresponding to the successor of the maximum ID of all elements.
- *PI*: The conflicting vertices are extracted and switched between the polylines. Therefore, the overall shape of the lanelet stays similar. Fig. 4 shows intersecting and repaired lanelet boundaries.
- *SC*: The function replaces the initial and final vertices of the two lanelets with vertices created based on the arithmetic mean of the corresponding initial and final vertices.
- *ALPS*: The vertices of the polylines are replaced by vertices created based on the arithmetic mean of the

TABLE II: Description of repairing functions.

Spec.	Description
SR	Removes the corresponding successor/predecessor reference.
PS	Sets the corresponding lanelets as predecessor/successor.
ALR	Removes the corresponding adjacency reference.
ELT	Removes wrong lanelet types.
DLTL	Places traffic sign close to lanelet.
IOA	Makes predecessor lanelet to outgoing or cuts lanelets.
NIG	Infers missing incoming group.
LOO	Changes outgoing type of lanelet.

vertices of the polylines.

- *LMA*: The final vertices of the merging lanelet are fit to the other lanelet, and the vertices of the right boundary of the merging lanelet are moved so that the polyline is contained by the other lanelet (cf. Fig. 2b).
- *AA*: The function searches for lanelets or areas in the proximity of the relevant area and checks whether those are adjacent. If there exists an adjacent area or lanelet, the corresponding adjacency is set. The area is deleted if no adjacent lanelet or area can be found.

VI. Evaluation

We evaluate the specification-compliance of publicly-available CommonRoad maps. Additionally, we evaluate the computational performance of our approach and compare it to other logical modeling methods used in the literature. The code and maps used for the evaluation can be found at https://gitlab.lrz.de/tum-cps/commonroad-map-verification.

A. Verification of CommonRoad Maps

We use 671 publicly available CommonRoad maps based on version 2020a for the evaluation. The evaluation was executed on an Intel Xeon Platinum 8362 2.8GHz processor using a single thread for each map. The parameters d_{ts}, α_1, and α_2 are set to 10m, 225°, and 315°, respectively. We have evaluated more than 50 specifications. The map with the most errors per number of road elements was USA_US101-9. Fig. 5 and Fig. 6 show different CommonRoad maps, each with several error types. Only 47 of 671 CommonRoad maps satisfied our formalized map specifications. The errors which occurred the most were the parallel adjacency error (ALPS), traffic sign distance error (DLTS), and the successor connection error (SC). Our algorithm could repair all invalid maps. The computation times and other statistical data of the evaluation of the three maps are listed in Tab. III. Additionally, Tab. III summarizes the computation times and statistics for the different data sources of the available CommonRoad maps.

B. Comparison with Other Approaches

We consider three logical approaches for our comparison: Answer Set Programming (ASP), Ontologies/Description Logic (DL), and Satisfiability Modulo Theories (SMT). ASP and Ontologies/DL are used for verifying maps in [19], [24]. Moreover, SMT is used to repair trajectories considering

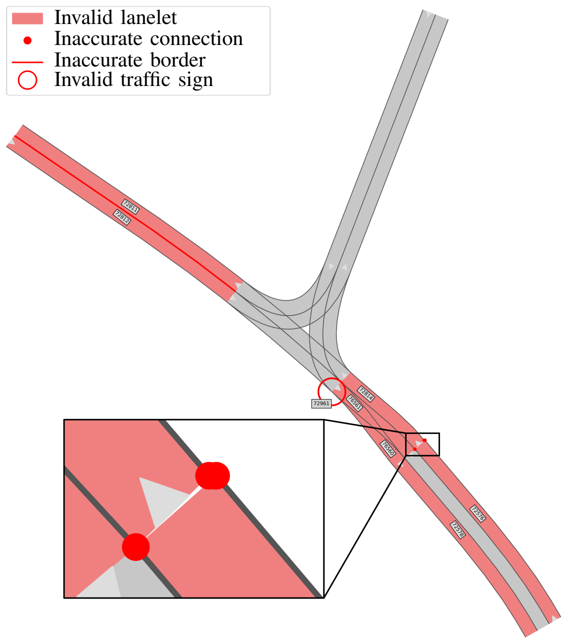


Fig. 5: Invalid CommonRoad map DEU_Guetersloh-20.

logical constraints [26]. We can only provide a short and informal introduction to the different approaches. However, a concrete implementation can be found in our code.

1) Ontologies/Description Logic: Ontologies structure information in a hierarchical and taxonomic way. We use a language based on the Web Ontology Language (OWL), i.e., OWL-DL, which can represent a subset of first-order logic and assumes an open world so that the correct truth value cannot be drawn from incomplete knowledge. Description logic is a formalism for knowledge representation and forms the logical basis of several ontologies.

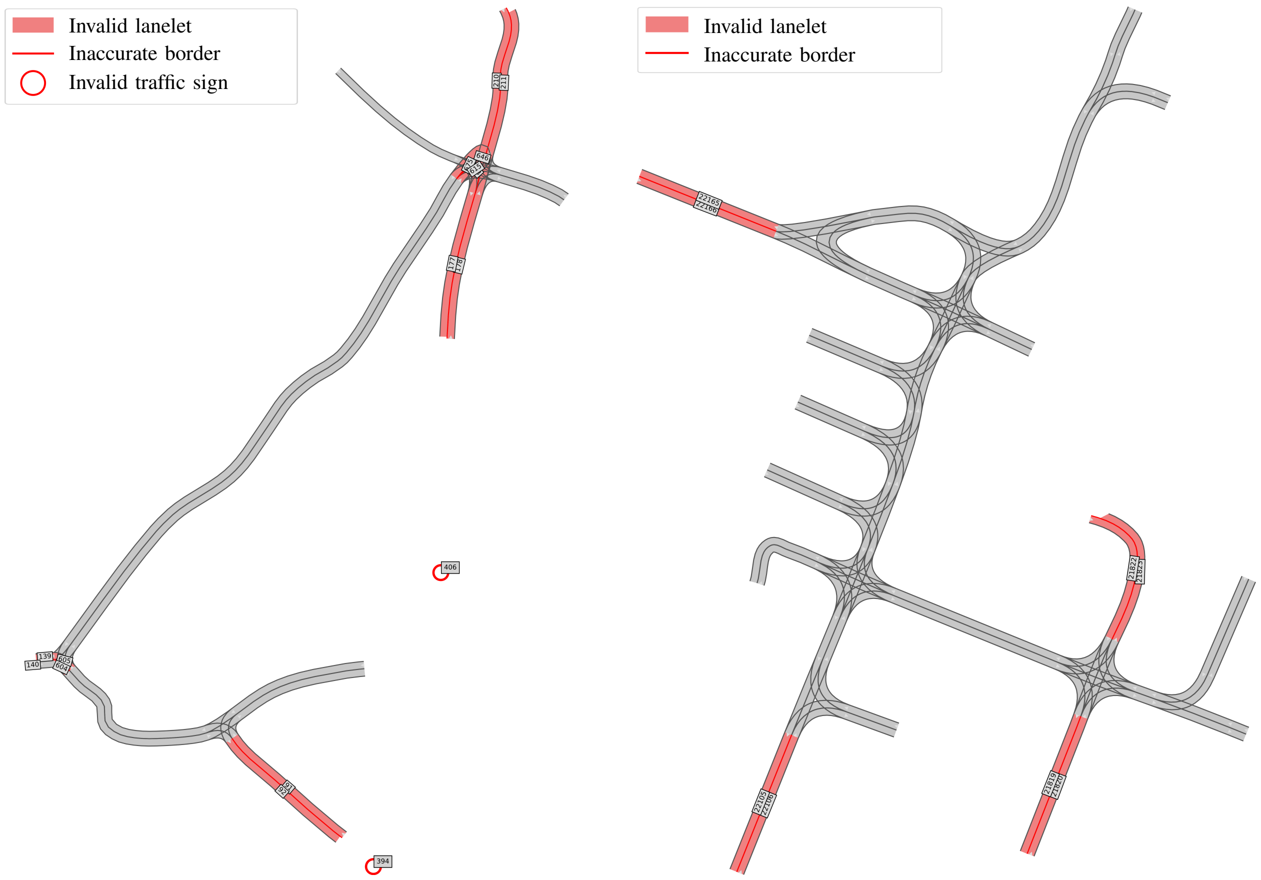
2) Answer Set Programming: ASP [27] is a declarative logic programming language using the concept of negation as failure and the related closed-world assumption. The problem and the domain knowledge are represented by a logic program consisting of a set of rules. In ASP, a rule is syntactically similar to Horn clauses and describes a logical consequence drawn under a condition.

3) Satisfiability Modulo Theories: Satisfiability Modulo Theories [28] determines the satisfiability of a logical first-order formula given a logical theory, e.g., linear arithmetic, arrays, and bit-vectors, and searches additionally for a suitable truth assignment. A theory $\mathcal{T}$ is a set of variable-free formulas and extends their expressiveness with operations that are not only based on Boolean logic, e.g., arithmetic summation or concatenation of strings. Each theory has a corresponding $\mathcal{T}$-solver that evaluates the operations along with Boolean logic.

4) Comparison: Tab. IV summarizes the computational performance of the four approaches. Since we have not formalized all specifications in all logics, we consider only 12 specifications for the comparison. Although our approach is implemented in Python, while the other approaches are mainly implemented in C and C++, our approach is faster

TABLE III: Evaluation of CommonRoad maps using 54 specifications.

Map	**Maps**	**Invalid Maps**	**Lanelets [km]**	$\lvert\mathcal{L}\rvert$	$\lvert\mathcal{TS} \cup \mathcal{TL}\rvert$	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{E}\rvert$	**HOL [s]**	**Repairing [s]**
DEU_Guetersloh-20	1	1	0.731	16	4	1	10	0.918	0.005
DEU_BadEssen-3	1	1	4.358	130	25	11	9	5.508	0.004
DEU_Lohmar-70	1	1	1.908	45	10	3	16	2.655	0.043
SUMO maps	203	175	274.080	8144	2767	640	1902	238.978	3.502
NGSIM maps	35	35	44.274	823	567	15	544	102.963	4.323
Scenario-Factory maps	320	315	705.015	19872	4292	1798	2426	529.846	1.061
Interactive maps	81	81	664.792	2089	2911	212	3991	831.989	19.470
Hand-Crafted maps	32	18	54.104	504	158	1	262	56.615	1.275
All maps accumulated	671	624	1742.265	31432	10695	2666	9125	1760.391	29.631



(a) Invalid CommonRoad map DEU_Lohmar-70.

(b) Invalid CommonRoad map DEU_BadEssen-3.

Fig. 6: Map errors of two CommonRoad maps consisting of several intersections.

than ASP, DL, and SMT.

Additionally, the other approaches have drawbacks in modeling our HOL formalization. The syntax of DL is similar to a subset of HOL. Unfortunately, variables are not offered in description logic, so that complex dependencies of elements in the road network cannot be represented. Therefore, description logic must be combined with logic programming, which makes the approach undecidable and negatively affects maintainability [29]. In ASP, decidability is ensured because of its closed-world assumption. However, the missing availability of operators limits ASP's readability and maintainability. The formulas in SMT must be carefully constructed due to the infinite boundness of the domains. Additionally, decidability is not guaranteed for SMT when evaluating the formulas for some combinations of operators, e.g., using a universal quantification without restricting the domains by subformulas.

VII. Conclusions

We presented the first approach for map verification and repairing that formalizes map specifications in higher-order logic, repairs elements from all map levels, provides a collection of map specifications, and is open-source. The overall algorithm can consider different types of logical reasoners. However, our custom reasoner is the most suitable considering computational performance and modeling

TABLE IV: Computational performance of all logical modeling approaches using 12 specifications.

Map	**ASP [s]**	**DL [s]**	**HOL [s]**	**SMT [s]**
DEU_Guetersloh-20	0.305	4.975	0.044	0.818
DEU_BadEssen-3	5.646	118.055	0.500	21.406
DEU_Lohmar-70	1.176	29.828	0.160	2.513
SUMO maps	152.936	2803.879	19.278	107.322
NGSIM maps	53.530	775.883	4.446	27.249
Scenario-Factory maps	673.023	10400.837	47.799	357.987
Interactive maps	1485.815	35372.567	33.823	622.131
Hand-Crafted maps	75.896	821.103	3.042	36.931
All maps accumulated	2441.200	50174.269	108.388	1151.620

efforts. The detected map inaccuracies show that many errors are still present in operational maps. This emphasizes the need for a formal map verification and repairing approach like ours. Our tool is part of the open-source CommonRoad Scenario Designer. Through the different interfaces to other map formats, our framework can also be used as a general map verification tool. Future work includes improvements of the computational performance to apply the tool in our research vehicle, e.g., for verifying maps generated on-the-fly based on sensor information. Additionally, we will regularly extend and update our specifications and repairing functions based on findings and user feedback.

ACKNOWLEDGMENT

The authors gratefully acknowledge partial financial support by the BMW Group within the CAR@TUM project and the Free State of Bavaria.

REFERENCES

[1] S. Maierhofer, A.-K. Rettinger, E. C. Mayer, and M. Althoff, "Formalization of interstate traffic rules in temporal logic," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2020, pp. 752–759.

[2] S. Maierhofer, P. Moosbrugger, and M. Althoff, "Formalization of intersection traffic rules in temporal logic," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2022, pp. 1135–1144.

[3] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2014, pp. 420–425.

[4] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017, pp. 719–726.

[5] F. Poggenhans, J. H. Pauls, J. Janosovits, S. Orf, M. Naumann, F. Kuhnt, and M. Mayr, "Lanelet2: A high-definition map framework for the future of automated driving," in *Proc. of the IEEE Int. Conf. on Intelligent Transportation Systems*, 2018, pp. 1672–1679.

[6] R. Liu, J. Wang, and B. Zhang, "High definition map for automated driving: Overview and analysis," *Journal of Navigation*, vol. 73, no. 2, p. 324–341, 2020.

[7] C. Guo, K. Kidono, J. Meguro, Y. Kojima, M. Ogawa, and T. Naito, "A low-cost solution for automatic lane-level map generation using conventional in-car sensors," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 8, pp. 2355–2366, 2016.

[8] K. Massow, B. Kwella, N. Pfeifer, F. Häusler, J. Pontow, I. Radusch, J. Hipp, F. Dölitzscher, and M. Haueis, "Deriving HD maps for highly automated driving from vehicular probe data," in *Proc. of the IEEE Int. Conf. on Intelligent Transportation Systems*, 2016, pp. 1745–1752.

[9] Y. Zhou, Y. Takeda, M. Tomizuka, and W. Zhan, "Automatic construction of lane-level HD maps for urban scenes," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2021, pp. 6649–6656.

[10] J. Hu, A. Razdan, J. C. Femiani, M. Cui, and P. Wonka, "Road network extraction and intersection detection from aerial images by tracking road footprints," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 45, no. 12, pp. 4144–4157, 2007.

[11] G. Máttyus, W. Luo, and R. Urtasun, "DeepRoadMapper: Extracting road topology from aerial images," in *Proc. of the IEEE Int. Conf. on Computer Vision*, 2017, pp. 3458–3466.

[12] M. Klischat, E. I. Liu, F. Höltke, and M. Althoff, "Scenario factory: Creating safety-critical traffic scenarios for automated vehicles," in *Proc. of the IEEE Int. Conf. on Intelligent Transportation Systems*, 2020, pp. 2964–2970.

[13] S. Maierhofer, M. Klischat, and M. Althoff, "CommonRoad Scenario Designer: An open-source toolbox for map conversion and scenario creation for autonomous vehicles," in *Proc. of the IEEE Int. Conf. on Intelligent Transportation Systems*, 2021, pp. 3176–3182.

[14] O. Hartmann, M. Gabb, R. Schweiger, and K. Dietmayer, "Towards autonomous self-assessment of digital maps," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2014, pp. 89–95.

[15] D. Pannen, M. Liebner, W. Hempel, and W. Burgard, "How to keep HD maps for automated driving up to date," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2020, pp. 2288–2294.

[16] A. Fabris, L. Parolini, S. Schneider, and A. Cenedese, "Correlation-based approach to online map validation," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2020, pp. 51–56.

[17] J.-H. Pauls, T. Strauss, C. Hasberg, M. Lauer, and C. Stiller, "Can we trust our maps? An evaluation of road changes and a dataset for map validation," in *Proc. of the Int. Conf. on Intelligent Transportation Systems*, 2018, pp. 2639–2644.

[18] C. Plachetka, N. Maier, J. Fricke, J.-A. Termöhlen, and T. Fingscheidt, "Terminology and analysis of map deviations in urban domains: Towards dependability for HD maps in automated vehicles," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2020, pp. 63–70.

[19] H. Beck, T. Eiter, and T. Krennwallner, "Inconsistency management for traffic regulations: Formalization and complexity results," in *Proc. of Logics in Artificial Intelligence*, 2012, pp. 80–93.

[20] B. Schwab and T. H. Kolbe, "Validation of parametric OpenDRIVE road space models," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. X-4/W2-2022, pp. 257–264, 2022.

[21] M. Buechel, G. Hinz, F. Ruehl, H. Schroth, C. Gyoeri, and A. Knoll, "Ontology-based traffic scene modeling, traffic regulations dependent situational awareness and decision-making for automated vehicles," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017, pp. 1471–1476.

[22] T. Eiter, H. Füreder, F. Kasslatter, J. X. Parreira, and P. Schneider, "Towards a semantically enriched local dynamic map," *International Journal of Intelligent Transportation Systems Research*, vol. 17, pp. 32–48, 2019.

[23] H. Qiu, A. Ayara, and B. Glimm, "Ontology-based map data quality assurance," in *European Semantic Web Conference*, 2021, pp. 73–89.

[24] B. Hummel, W. Thiemann, and I. Lulcheva, "Scene Understanding of Urban Road Intersections with Description Logic," in *Logic and Probability for Scene Interpretation*, ser. Dagstuhl Seminar Proceedings, vol. 8091, 2008, pp. 1–16.

[25] M. Raszyk, D. Basin, S. Krstić, and D. Traytel, "Multi-head monitoring of metric temporal logic," in *Proc. of Automated Technology for Verification and Analysis*, 2019, pp. 151–170.

[26] Y. Lin and M. Althoff, "Rule-compliant trajectory repairing using satisfiability modulo theories," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2022, pp. 449–456.

[27] G. Brewka, T. Eiter, and M. Truszczyński, "Answer set programming at a glance," *Commun. ACM*, vol. 54, no. 12, p. 92–103, 2011.

[28] C. Barrett and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Model Checking*. Springer, 2018, pp. 305–343.

[29] S. Grimm, "Knowledge representation and ontologies," in *Scientific data mining and knowledge discovery: Principles and foundations*. Springer, 2009, pp. 111–137.