From Specifications to Behavior: Maneuver Verification in a Semantic State Space

Klemens Esterle[1], Vincent Aravantinos[1] and Alois Knoll[2]

***Abstract*— To realize a market entry of autonomous vehicles in the foreseeable future, the behavior planning system will need to abide by the same rules that humans follow. Product liability cannot be enforced without a proper solution to the approval trap. In this paper, we define a semantic abstraction of the continuous space and formalize traffic rules in linear temporal logic (LTL). Sequences in the semantic state space represent maneuvers a high-level planner could choose to execute. We check these maneuvers against the formalized traffic rules using runtime verification. By using the standard model checker NuSMV, we demonstrate the effectiveness of our approach and provide runtime properties for the maneuver verification. We show that high-level behavior can be verified in a semantic state space to fulfill a set of formalized rules, which could serve as a step towards safety of the intended functionality.**

I. Introduction

A. Motivation

A lot of effort has been put into demonstrating the feasibility of behavior planning, but an approach guaranteeing safety of the intended function (SOTIF, see [1]) is still far away. Building up a safety case for an autonomous driving system based on mileage driven is not suitable due to the high-dimensional state space of real-world scenarios. Instead, advances in verifiability, safety assessment and explainability will be essential to make autonomous driving reliable [2].

Traffic rules have been designed to help humans manage the otherwise chaotic traffic environment. If all vehicles were fully autonomous, the current set of rules could be reduced or adapted. In a transition period with mixed traffic, however, we need to make sure that the planned behavior satisfies those specified rules at all times. First of all, obeying those rules would make the behavior of autonomous cars more understandable to other traffic participants. Secondly, it will help clarify the liability of the autonomous vehicle in case of an accident.

For the behavior of an autonomous vehicle to follow traffic rules, large state-machines have been used (e.g. [3–5]), requiring extensive and careful engineering. Transferring such state machines to different markets with other specifications does not scale well and quickly becomes hard to maintain. The problem of behavior planning is challenging due to the high-dimensional continuous state space, non-linear motion dynamics, interactions with other agents and their unobservable intentions. This has led to researchers investigating collaborative approaches [6], game-theoretic approaches [7], as well as probabilistic approaches [8].

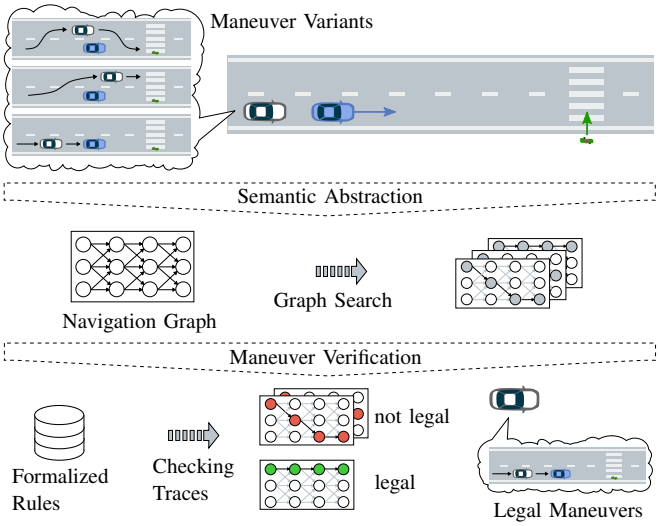

Fig. 1: Overview of our approach. The combinatorial maneuver variants are made accessible by building up a semantic graph structure of the predicted evolution of the scenario. A graph search yields maneuvers, which we check against formalized rules.

However, since these new approaches try to incorporate reactive predictions into the decision making process, the complexity is growing and it becomes even harder to fuse them with classic rule-based systems to abide by the traffic regulations and specifications. Rules get either hard-coded or are inherently learned from data, which are not transferable solutions in case of rule changes. Specification as the satisfaction of traffic rules should thus be formalized in some way, for instance as formal logic.

B. Contribution

Checking potentially many trajectory hypotheses for complex traffic rules in a continuous space entails high computational costs. In this paper, we address the problem of verifying the high-level behavior of an autonomous vehicle in a semantic state space to satisfy a set of traffic rules. The solution shall be easily integrated into a planning framework so that the final trajectory also satisfies the rules and does not need to be checked again. We make use of the work of [9], building our verification approach on top of that. Fig. 1 gives an overview of our approach.

The main contributions of this paper are:

- an LTL representation corresponding to the partitioning method of [9],

[1]Klemens Esterle and Vincent Aravantinos are with fortiss GmbH, An-Institut Technische Universität München, Munich, Germany

[2]Alois Knoll is with Robotics, Artificial Intelligence and Real-time Systems, Technische Universität München, Munich, Germany

- the formalization of some traffic rules to LTL for the introduced semantic state,
- a method to verify high-level maneuvers to comply to these rules,
- evaluations showing computational properties of the approach.

This paper is organized as follows: Section II presents work directly related to our paper. Section III defines the addressed problem. Section IV introduces the necessary concepts this work is based on. The proposed method is presented in Section V. In Section VI, we formalize a set of traffic rules. Section VII evaluates the algorithm's abilities followed by a discussion in Section VIII.

II. Related Work

Recent work such as [10–12] investigated verifying reach-and-avoid tasks[1] or routing tasks in linear temporal logic (LTL). LTL is a formal language that combines boolean and temporal operators. Plaku and Karaman [13] argue for its expressiveness on sequencing and partial ordering as well as persistency (*"move slowly"*). Wolff *et al.* [10] transform linear temporal logic constraints to mixed integer linear constraints. In this way, temporal constraints can be integrated into optimization-based motion planning without creating an abstraction to verify the LTL formula. Solving this MILP leads to an optimal trajectory satisfying the logical specifications. They apply this approach to reach-avoid tasks, whereas we focus on maneuver planning.

Rizaldi *et al.* [11] present an automata-based maneuver planner where each motion primitive is encoded as a state. They formalize reach ("*until you reach the goal*") and avoid tasks ("*do not collide with an obstacle*") to LTL to check the validity of a transition from one state to another. In contrast, we identify atomic propositions that allow us to express high-level maneuvers such as overtaking directly in LTL.

Rizaldi and Althoff were the first to use logic as a language to formalize Vienna Road Traffic rules by specifically using higher order logic (HOL) in [14]. Their work is extended in [15], where they use LTL to formalize German traffic rules. As they use high-level states such as *overtaking* or *merging* as atomic propositions, they need to verify the value of the *overtaking* proposition externally (i.e., not LTL). In contrast, we describe maneuvers such as overtaking using LTL as a sequence of states. Their approach checks for rule satisfaction in a continuous state space and can be used as the monitor system of an ego trajectory, whereas we formulate rules in a semantic state space.

Kohlhaas *et al.* [16] use a semantic state space that consists of the state and action space of the ego vehicle for a high-level maneuver planner. Relations of objects and road elements are represented in an ontology. They transform the semantic space as a directed graph and use a graph search to generate feasible solutions. They claim that their semantic state space is capable of representing traffic rules by the use of conditions. However, they do not describe which rules they model. In contrast, we present a set of rules from the Vienna Convention in regard to overtaking and passing orders.

[1]In reach-and-avoid tasks, the robot stays in a safe region until the goal region is reached.

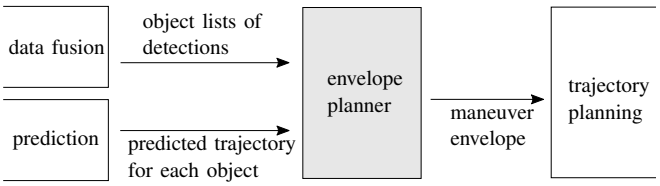

Fig. 2: The envelope planner unifies maneuver generation and verification. The semantic representation is calculated from an object list and a predicted trajectory of each object. Our contribution is the highlighted block.

Reyes Castro *et al.* [17] translate LTL specifications into an automaton. A second automaton is created as a discrete abstraction of an RRT*-explored workspace. A discrete search then creates feasible traces satisfying the LTL formula which guide the RRT* into feasible directions satisfying the specification. Similar to our approach, they check for the satisfaction of rules at an abstract layer. However, they only check for safety rules such as *"do not cross solid center lines"* or *"do not travel in the wrong direction"*. In contrast, the combinatorial approach we use directly applies lane constraints to the planning problem, thus satisfying these types of rules by design.

III. Problem Definition

We follow the behavior planning problem formulation we introduced in [18]. There, we separate the problem into high-level envelope planning and low-level trajectory planning inside a homotopy. Based on an object list and a prediction for each object, the envelope planner has to plan a maneuver envelope

$$\boldsymbol{\zeta}_p = [s_{max},\, s_{min},\, d_{left}(s_i),\, d_{right}(s_i)] \tag{1}$$

at each prediction time step θ_p in local street-wise coordinates with the arc length s and the perpendicular offset d. The maneuver envelope is then used for low-level trajectory planning. Fig. 2 shows our contribution embedded in a planning framework.

In this paper, our goal is to generate envelopes satisfying a set of rules. To do so, we generate a semantic trace τ representing a possible maneuver variant. When selecting such a maneuver, we need to consider rules such as passing orders and overtaking rules. We formulate the satisfaction of traffic regulations and social conventions on the high-level envelope planner as a model checking problem. Given a semantic trace τ and a set of rules R_i, the following should hold:

$$\tau \models \bigwedge_{i}^{n} R_i. \tag{2}$$

The operator $\models$ denotes the satisfaction relation of LTL.

IV. Preliminaries

A. Linear Temporal Logic

Temporal logics extend classical logics by temporal operators to reason not just about an absolute truth but about truths which might hold only at some points in time. Linear temporal logic (LTL) is one of the simplest, building upon propositional logic by adding temporal operators.

1) Syntax: Formally, the language ρ of LTL formulas is defined as

$$\rho ::= A \,|\, \neg A \,|\, \rho_1 \wedge \rho_2 \,|\, \rho_1 \vee \rho_2 \,|\rho_1 \Longrightarrow \rho_2 \,|\, \bigcirc \rho \,|\, \rho_1 \mathcal{U} \rho_2 \,|\, \Box \rho \,|\, \Diamond \rho,$$

where A denotes a set of *atomic* propositions, $\neg$ (resp. $\wedge$, $\vee$, $\Longrightarrow$) denote the boolean operators "not", "and", "or" and "implies", and $\bigcirc$, (resp. $\mathcal{U}$, $\Box$, $\Diamond$) denote the temporal operators "next", "until", "globally" (or "always"), "finally" (or "eventually").

Precedence is defined as follows: The unary operators bind stronger than the binary ones, e.g. $\rho_1 \,\mathcal{U} \bigcirc \rho_2$ is equal to $\rho_1 \,\mathcal{U}\, (\bigcirc \rho_2)$. The "until" operator is right-associative, e.g. $\rho_1 \mathcal{U} \rho_2 \mathcal{U} \rho_3$ is equal to $\rho_1 \mathcal{U} (\rho_2 \mathcal{U} \rho_3)$. $\neg$ and $\bigcirc$ bind equally strong. Operator $\mathcal{U}$ takes precedence over $\wedge$, $\vee$, $\Longrightarrow$.

We first need to characterize when a formula is true or not. In LTL, this is done with respect to a *trace*:

Definition 4.1: A *valuation* of A is a function that assigns to each proposition of A an element of $\{\mathbf{T}, \mathbf{F}\}$. A *trace* is an infinite sequence of valuations.

For instance, if $A = \{X, Y, Z\}$, then the function f defined by $f(X) = \mathbf{T}$, $f(Y) = \mathbf{F}$, $f(Z) = \mathbf{T}$ is a valuation of A. Intuitively, it represents the truth value of every variable in a given configuration. A trace represents the evolution of such configurations over time. Note that from an algorithmic viewpoint, a maneuver defined over a given planning horizon is finite. A finite sequence of valuations is turned into a trace by repeating the last valuation indefinitely.

2) Semantics: We can define the semantics of an LTL formula as follows:

Definition 4.2: For every formula ρ, every instant i and every trace τ, the relation $\tau, i \models \rho$ (pronounced "ρ holds for τ at instant i") is inductively defined as follows:

- $\tau, i \models a \in A$ if and only if (iff) $\tau_i = \mathbf{T}$,
- $\tau, i \models \rho_1 \wedge \rho_2$ iff $\tau, i \models \rho_1$ and $\tau, i \models \rho_2$,
- $\tau, i \models \rho_1 \vee \rho_2$ iff $\tau, i \models \rho_1$ or $\tau, i \models \rho_2$,
- $\tau, i \models \rho_1 \Longrightarrow \rho_2$ iff $\tau, i \not\models \rho_1$ or $\tau, i \models \rho_2$,
- $\tau, i \models \bigcirc \rho$ iff $\tau, i+1 \models \rho$,
- $\tau, i \models \Box \rho$ iff for all $j \geq i$, $\tau, j \models \rho$,
- $\tau, i \models \Diamond \rho$ iff there exists $j \geq i$ s.t. $\tau, j \models \rho$,
- $\tau, i \models \rho_1 \,\mathcal{U}\, \rho_2$ iff there exists $j \geq i$ such that $\tau, j \models \rho_2$ and for all k $i \leq k \leq j$, $\tau, k \models \rho_1$.

When we have a trace τ with the configurations s_i with $i \in 1, 2, 3$, we denote it by $(s_1 \rightarrow s_2 \rightarrow s_3)$. Table I provides examples for the temporal operators.

B. Free Space-Time Partitioning

To facilitate high-level decisions, an intermediate representation bridging the gap between obstacle lists from sensor fusion and semantic information is needed. We build on top of the work of Altché and De La Fortelle [9], that partitions collision-free space-time into discrete three-dimensional cells with geometric adjacency relations, described in the following.

TABLE I: Truth values of LTL formulas ρ are displayed in a receding fashion for the atomic propositions $A = \{x, y\}$ at time instants $i = 1, ..., 4$.

ρ	A	$i=1$	$i=2$	$i=3$	$i=4$	...
	x	**T**	**F**	**T**	**T**	**T**
	y	**F**	**T**	**F**	**F**	**F**
$\bigcirc x$		**F**	**T**	**T**	**T**	**T**
$\Box x$		**F**	**F**	**T**	**T**	**T**
$\Diamond y$		**T**	**T**	**F**	**F**	**F**
$y \,\mathcal{U}\, x$		**T**	**T**	**T**	**T**	**T**

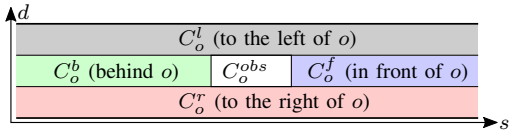

Fig. 3: Partitioning of the two-dimensional space around a single obstacle o into four collision-free regions C_o^i., from [9].

We first briefly introduce the concept of partitioning in 2D space. Let $\mathcal{Q}$ denote the extend of the road in the Frenet space. The free space $\mathcal{Q}_f$ around an obstacle o is partitioned into four collision-free regions – front, behind, left and right. For each obstacle, regions are derived in accordance with Fig. 3. Fig. 4 illustrates the concept of enumeration of the free space. The set-based intersection of the region "behind o_1" (denoted by $C^b_{o_1}$) and "to the right of o_2" ($C^r_{o_2}$) yields a new region "to the right of o_1 and behind o_2". This region is abbreviated with the signature $\sigma = (rb)$, where the order of the letters refers to the respective obstacle o_i.

Partitioning the 2D free space at each discrete time step yields a set of free space-time cells E^p_σ, denoted by $\mathcal{P}$. E^p_σ denotes the space-time cell corresponding to the signature σ at time step θ_p. The term *cell* is used here in contrast to 2D, where we only speak of *regions*.

From this discrete cell decomposition, adjacency information is calculated to create a graph structure. The discrete transition graph $\mathcal{G}$ is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the vertex set $\mathcal{V}$ and the edges set $\mathcal{E}$. We let $\mathcal{V} = \mathcal{P}$. Edges between two cells $E^p_{\sigma_1}$ and $E^{p+1}_{\sigma_2}$ at two consecutive time steps θ_p and θ_{p+1} exist if the cells are adjacent at both time steps. For the definition of adjacency $\text{adj}_{\theta_p}(\cdot, \cdot)$, we refer the reader to [9]. Formally, $\mathcal{E}$ is defined as:

$$\mathcal{E} = \left\{ (E^p_{\sigma_1}, E^{p+1}_{\sigma_2}) | E^p_{\sigma_1}, E^{p+1}_{\sigma_2} \in \mathcal{V}, \text{adj}_{\theta_p}(\sigma_1, \sigma_2) = 1 \right\}.$$

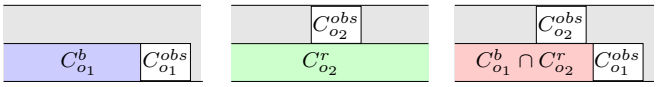

Fig. 4: Set-based union of two regions $C^b_{o_1}$ and $C^r_{o_2}$ to form a region that describes the free space "to the right of o_1 and behind o_2".

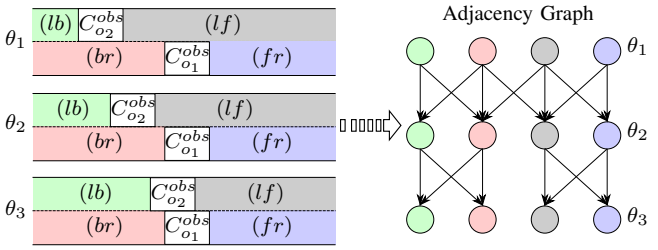

Fig. 5: Mapping of an obstacle-related partitioning of the free space to a directed graph, that incorporates the adjacency information of the free space-time partitions.

Fig. 5 illustrates the relationship between the free space-time partitioning and the adjacency graph, called *navigation graph*. Since the obstacle o_2 moves faster than o_1, the cells with the signatures (br) and (lf) are no longer adjacent at θ_3. Therefore, no connection exists in the adjacency graph between the corresponding vertices.

Each path $(E^0_{\sigma_1}, ..., E^{m+1}_{\sigma_{m+1}})$ in $\mathcal{G}$ corresponds to a set of homotopic collision-free trajectories (that may, however, be infeasible). This reduces the behavior generation problem to a discrete selection of a maneuver variant and trajectory planning for the obtained maneuver variant.

V. Maneuver Verification

A. Semantic State Space Definition

The challenge of formalizing traffic rules in LTL starts with the selection of an expressive semantic state space and abstracting that as atomic propositions. Based on the navigation graph, we introduce a semantic state space for each agent in the scenario except the ego vehicle:

- Obstacle-related:
  - position of the free space in relation to the traffic participant (in front f, behind b, to the left l, to the right r),
  - traffic participant type vehicle v, pedestrian p, cyclist c, rail-borne r (denoted as a subscript of the position).
- Road type-related (pedestrian crosswalk $\mathcal{R}_{pc}$, carriageway $\mathcal{R}_{cw}$).

Let us formalize these notions in a form suitable for LTL. We first need to define atomic propositions. There exist only two road type propositions: $\mathcal{R}_{pc}$ and $\mathcal{R}_{cw}$. For obstacle propositions, let $v_1, ..., v_{n_v}$ (resp. $p_1, ..., p_{n_p}$, resp. $c_1, ..., c_{n_c}$) be the set of all vehicles (resp. pedestrians, resp. cyclists) in the scenario. Then, for every obstacle $o \in \{v_1, ..., v_{n_v}, p_1, ..., p_{n_p}, c_1, ..., c_{n_c}\}$, the following are atomic propositions: f_o, b_o, l_o, r_o.

Let α be a representation of the current situation. We can create a corresponding LTL trace, in the following denoted by τ_α (recall that a trace is a sequence of functions assigning to every atomic variable a truth value at a given instant). First, we have $\tau_{\alpha_i}(\mathcal{R}_{pc}) = \mathbf{T}$ (resp. $\tau_{\alpha_i}(\mathcal{R}_{cw}) = \mathbf{T}$) iff the road type is a pedestrian crosswalk (resp. on the carriageway) at instant i.

For every instant i and every object o, we have:

- $\tau_{\alpha_i}(f_o) = \mathbf{T}$ iff ego is in front of o at instant i,
- $\tau_{\alpha_i}(b_o) = \mathbf{T}$ iff ego is behind o at instant i,
- $\tau_{\alpha_i}(l_o) = \mathbf{T}$ iff ego is to the left of o at instant i,
- $\tau_{\alpha_i}(r_o) = \mathbf{T}$ iff ego is to the right of o at instant i.

B. Road type-based Partitioning

Following our state space definition, we extend the partitioning of the road space $\mathcal{Q}$ (see Section IV-B) to include road type information. Instead of checking the road-type for each free space-time cell when evaluating a trace, we choose to encode the property of road-type in the discrete graph. This way, we keep the direct mapping from a trace to a maneuver intact instead of introducing an intermediate step of processing a discrete path and cutting out road-type related regions.

We calculate a set of road regions $\mathcal{P}^r$, where the superscript r denotes the partitioning related to the road type. We then use $\mathcal{P}^r$ as the initial set of regions for the free space-time partitioning, following Algorithm 1 in [9].

C. Runtime Verification

Classical model checking would require translating the graph structure into an automaton. The model checker would then generate traces that comply with the formalized rules. However, we choose to go in another direction by performing runtime verification. We do not create an automaton from the full graph but only from a graph traversal from the root node to a goal node thus yielding only a trace instead of a "full-fledged" automaton. We then use a model checker to check whether such a trace τ satisfies the specified rules according to (2). The benefit of our formulation is that we leave the general concept of [9, 19] untouched, which is to assign heuristic costs to each graph solution, then rank them along those costs and select a finite number of graph solutions for trajectory planning. Using additional information as graph costs such as time gaps would not directly be possible if we translated the graph to an automaton for model checking. Another benefit of runtime verification is its possible use in combination with other planners as the trace to be checked does not necessarily have to be a graph solution.

However, the creation of a single automaton for each trace instead of the whole graph creates additional computational overhead. Note that we do not know beforehand whether the selected path from the transition graph is dynamically feasible. It thus may be necessary to check many traces. The model checking approach on the other hand would always provide valid plans.

Algorithm 1 shows the proposed method. For a given planning interval $\Delta\theta$, we construct the semantic finite abstraction of the current state of all relevant traffic participants and their predicted motions by first partitioning the state space (Line 6). The function generateFiniteAbstraction() contains the partitioning along the road type (Section V-B) and the free space (see Section IV-B). For a detailed description of the free space partitioning and the calculation of adjacency in generateAdjacencyGraph(), we refer the reader to [9]. From

that, we obtain a graph representation (Line 7). The starting node $\sigma_{ego}^{\theta_0}$ represents the cell where the ego vehicle is located at the initial planning time step. As we do not know the final cell of the optimal maneuver yet, all cells at the final time step $\mathcal{V}^{\theta_{n-1}}$ are possible goal cells. We perform a graph search to get all possible traces T (Line 11). We finally calculate the time margins using a function calculateTimeMargins() as in [9]. We assume the existence of a function sort() to sort the traces by their costs (Line 14). We then verify each semantic trace τ using the proposed verification method.

We turn a finite time trace into an infinite trace by repeating the last state indefinitely. This may cause inaccuracies, as we evaluate the next-operator on the last (repeated) state. Investigating these side-effects in detail and evaluating finite-time semantics is the subject of future work.

As the instants i correlate with prediction time steps, a verified trace from T_{sat} satisfying the specifications can be directly mapped to a maneuver envelope as defined in (1), serving as the input to the trajectory planner.

VI. Vienna Convention Traffic Rules as LTL

Based on the Vienna Convention of Traffic Rules [20], we have selected rules that cover the interaction of the ego vehicle with only one other traffic participant. Extending this to rules that cover more traffic participants is the subject of future work.

Traffic Rule 1: Do not overtake a vehicle on its right side, except in congested traffic, where overtaking on the right is also allowed *(Article 11.1, 11.6)*.
The rule concerns roads with two or more lanes in the same direction such as highways. The rule shall only apply in congested traffic. Formalizing congestion requires the speed of the vehicles, which we choose not to include in our semantic state space definition. Here, we assume the scene interpretation module to provide a signal `CONGESTED`. It could for example be set to true if all other vehicles are slower than a certain threshold. We formulate R_1 as:

$$R_1(v) : \neg\texttt{CONGESTED} \implies \square\neg(b_v \wedge \bigcirc(b_v \,\mathcal{U}\, r_v \,\mathcal{U}\, f_v)).$$

This means that except for the case of congested traffic, we do not allow a temporal sequence in which the ego vehicle starts behind a vehicle, goes to the right of the vehicle and then goes in front of the vehicle. The until operator allows to follow this rule independently of the number of instants that the ego vehicle stays to the right of the other vehicle. If the first occurrence of b_v were not in the formula, the trace $(r_v \rightarrow f_v)$ would not be valid. We thus limit the forbidden set by adding b_v followed by the ordering $b_v \,\mathcal{U}\, r_v \,\mathcal{U}\, f_v$. Fig. 6 illustrates a scenario in which the obtained trace for the overtaking maneuver on the right does violate R_1. Note that since the rule applies to each obstacle, we perform the rule checking for each obstacle separately. For simplicity reasons, Fig. 6 displays only an abbreviated signature of the trace omitting road type information.

Traffic Rule 2: Do not overtake a vehicle that slows down or waits in front of a pedestrian walk *(Article 11.9)*.

We rephrase as follows: *A vehicle shall not be overtaken if a pedestrian crosswalk is immediately in front of the vehicle*. This is a slightly modified version of the actual rule as we do not model the slow-down but only the position of the crosswalk in relation to the vehicle. This way we impose that a vehicle which is not slowing down should not be overtaken as well, which we consider a reasonable assumption as well. Formally, this results in

$$R_2(v) : \square\neg(b_v \wedge \bigcirc(b_v \,\mathcal{U}\, l_v \,\mathcal{U}\, (f_v \wedge \mathcal{R}_{pc}))).$$

Algorithm 1 Maneuver Verification on Navigation Graph

1: **Input** $\Delta\theta$: planning interval
2: **Input** n: number of time steps
3: **Input** env: environment
4: **Output** T_{sat}: set of rule-satisfying traces
5: $T_{sat} \leftarrow \emptyset$
6: $\mathcal{P} \leftarrow$ generateFiniteAbstraction(env, $\Delta\theta$, n)
7: $\mathcal{G} \leftarrow$ generateAdjacencyGraph($\mathcal{P}$, env)
8: $\sigma_{ego}^{\theta_0} \leftarrow$ getRootVertice($\mathcal{G}$, env)
9: $\mathcal{V}^{\theta_{n-1}} \leftarrow$ getGoalVerticeCandidates($\mathcal{G}$, env)
10: **for each** $\sigma^{\theta_{n-1}}$ in $\mathcal{V}^{\theta_{n-1}}$ **do**
11: $\quad\tau \leftarrow$ findAllTraces($\mathcal{G}$, $\sigma_{ego}^{\theta_0}$, $\sigma^{\theta_{n-1}}$)
12: $\quad$costs$[\tau] \leftarrow$ calculateTimeMargins(τ)
13: **end for**
14: $T_{sorted} \leftarrow$ sort(T, costs)
15: **while** $T_{sorted} \neq \emptyset$ **do**
16: $\quad$**if** satisfiesLTLSpec(τ) **then**
17: $\quad\quad T_{sat} \leftarrow T_{sat} \,\mathcal{U}\, \tau$
18: $\quad$**end if**
19: **end while**

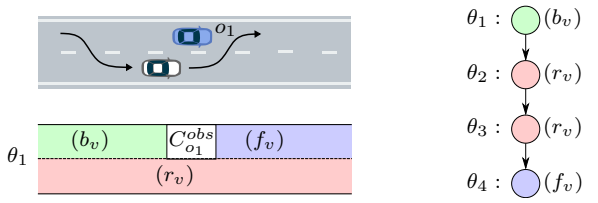

Fig. 6: Traffic Rule 1: Possible maneuver overtaking the red vehicle o_1 on the right. The partitioning around o_1 at θ_1 is displayed, which stays similar for the time steps $\theta_{1..4}$. The semantic trace $(b_v \rightarrow r_v \rightarrow r_v \rightarrow f_v)$ of that maneuver violates R_1.

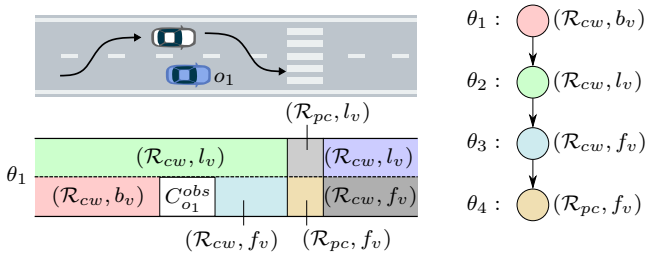

Fig. 7: Traffic Rule 2: The red vehicle approaches a pedestrian walk. A possible maneuver overtaking the red vehicle o_1 on the left is illustrated. The partitioning around o_1 at θ_1 is displayed. The semantic trace $(\mathcal{R}_{cw}, b_v \rightarrow \mathcal{R}_{cw}, l_v \rightarrow \mathcal{R}_{cw}, f_v \rightarrow \mathcal{R}_{pc}, f_v)$ of that maneuver violates R_2.

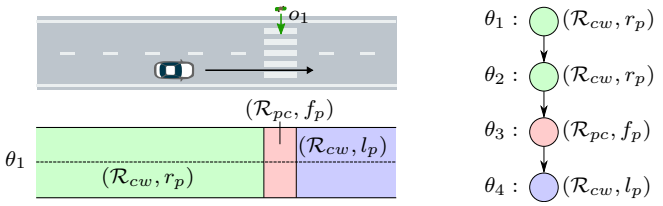

Fig. 8: Traffic Rule 3: A pedestrian to the left of the ego vehicle intends to use a pedestrian crossing. A possible maneuver to pass before the pedestrian crosses the street is illustrated. The semantic trace $(\mathcal{R}_{cw}, r_p \rightarrow \mathcal{R}_{cw}, r_p \rightarrow \mathcal{R}_{pc}, f_p \rightarrow \mathcal{R}_{cw}, l_p)$ of that maneuver violates R_3.

TABLE II: Example traces τ_i and whether they satisfy rule R_1. We omit the road type information $\mathcal{R}$ in the LTL trace as it does not have any impact on the satisfaction of R_1.

trace τ_i	θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	$\tau_i \models R_1$
τ_1	b_v	b_v	l_v	f_v			**T**
τ_2	b_v	l_v	l_v	b_v			**T**
τ_3	b_v	b_v	r_v	b_v			**T**
τ_4	r_v	r_v	f_v	f_v			**T**
τ_5	b_v	r_v	r_v	f_v			**F**
τ_6	b_v	r_v	f_v	f_v			**F**
τ_7	b_v	r_v	f_v	r_v			**F**
τ_8	b_v	r_v	r_v	b_v	r_v	f_v	**F**

The rule is similar to R_1, mirroring the overtaking maneuver on the right instead of the left. The last temporal state we request for such a trace to be forbidden needs to be in front of the respective vehicle but also of the road type *pedestrian crosswalk* $\mathcal{R}_{pc}$. Fig. 7 illustrates a scenario where the maneuver to overtake the red vehicle right in front of a pedestrian crosswalk will be forbidden as it violates R_2.

Traffic Rule 3: Stop if pedestrians are using or intend to use a pedestrian crossing. *(Article 21.3)*
We formulate it as

$$R_3(p) : \square\, \neg(\mathcal{R}_{pc} \wedge f_p),$$

such that no state in the trace is allowed that is a pedestrian walk and lies in front of a pedestrian. The formulation covers pedestrians coming from the left and right. Fig. 8 illustrates a maneuver that R_3 forbids.

VII. Evaluation

A. Evaluating the Formalized Rules

We will first use synthetic traces to show the effectiveness of our formalized rules. Table II shows that the trace τ_5 $(b_v \rightarrow r_v \rightarrow r_v \rightarrow f_v)$ representing a maneuver to overtake on the right violates R_1. It can also be seen that the sequence defining a maneuver to overtake on the right $(b_v \rightarrow r_v \rightarrow f_v)$ in the trace τ_8 can be captured and classified as violating as well, no matter how many states the trace has. However, there are some drawbacks from the selected state space. The relation "right of o_i" states that o_i is in the next lane to the right. Still, we currently cannot capture that o_i could be $30\mathrm{m}$ before or behind the ego vehicle. An extension of the state space could be beneficial to formalize such rules more clearly. We leave this to future work.

TABLE III: Example traces τ_i and whether they satisfy rule R_2.

trace τ_i	θ_0	θ_1	θ_2	θ_3	$\tau_i \models R_2$
τ_1	$R_{cw}; b_v$	$R_{cw}; b_v$	$R_{cw}; l_v$	$R_{cw}; f_v$	**T**
τ_2	$R_{cw}; b_v$	$R_{pc}; b_v$	$R_{cw}; l_v$	$R_{cw}; f_v$	**T**
τ_3	$R_{cw}; b_v$	$R_{cw}; b_v$	$R_{cw}; l_v$	$R_{pc}; f_v$	**F**

TABLE IV: Example traces τ_i and whether they satisfy rule R_3

trace τ_i	θ_0	θ_1	θ_2	θ_3	$\tau_i \models R_3$
τ_1	$R_{cw}; r_p$	$R_{cw}; f_p$	$R_{cw}; f_p$	$R_{pc}; l_p$	**T**
τ_2	$R_{cw}; l_p$	$R_{cw}; f_p$	$R_{cw}; f_p$	$R_{pc}; r_p$	**T**
τ_3	$R_{cw}; l_p$	$R_{pc}; f_p$	$R_{pc}; f_p$	$R_{cw}; r_p$	**F**

Rule R_2 forbids overtaking of a vehicle that is right in front of a pedestrian walk. Table III shows example traces and whether they satisfy rule R_2 or not. In regard to the obstacle-relation $(b_v \rightarrow b_v \rightarrow l_v \rightarrow f_v)$, $\tau_{i|1..4}$ and τ_5 are identical. However, they differ in the road type of their final state, which is why τ_5 violates R_2 and the others do not.

Rule R_3 forbids any state that is a pedestrian crosswalk and in front of a pedestrian. Table IV shows a trace τ_1 with $(r_p \rightarrow f_p \rightarrow f_p \rightarrow l_p)$. This represents a situation in which a pedestrian is to the left of the street oriented towards the street, while the ego vehicle passes the pedestrian. As can be seen in Table IV, rule R_3 forbids such a trace if the pedestrian stands at a pedestrian crosswalk, represented by the state $\mathcal{R}_{pc}, f_p$. If not, the trace will be classified as legal.

B. Maneuver Verification

We want to answer the following questions:

- Is it feasible to check all maneuvers?
- How long does it take to check a maneuver?
- How does the complexity of the problem influence the runtime?

1) Implementation: As explained in Section V-C, we do runtime verification instead of model checking. Although it might appear as an overkill, we use NuSMV, a powerful model checking tool [21], since this will allow us to easily modify and extend our approach in the future. All algorithms have been implemented in MATLAB. The simulations were performed on a 2.6GHz desktop PC with 16GB RAM.

2) Scenarios: We evaluate three different scenarios: A two-lane same-direction scenario similar to [9], a two-lane scenario with a pedestrian crossing, and a three-lane highway *CommonRoad* scenario [22]. We assume full observability and use a constant-acceleration model to predict the other participants. Fig. 9 illustrates the scenarios. We compare all scenarios for a planning horizon of $4\mathrm{s}$ for two different grades of discretization. We compute the predicted occupancy of each obstacle between consecutive time instants $[t_k, t_{k+1}]$. This ensures that we do not miss any occupancy when partitioning at larger step sizes. However, the occupied areas grow with the size of the step size. Some maneuvers might thus not be available at a broader discretization.

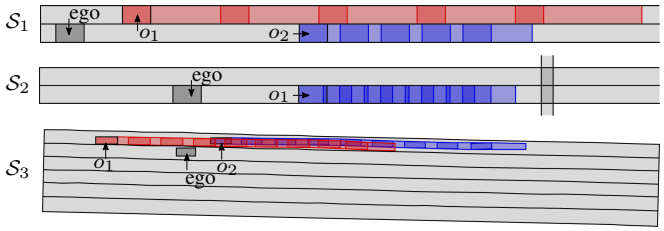

Fig. 9: Scenarios $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$.

3) Computational Results: Table V summarizes our computational results. The size of the adjacency graph depends on the number of time steps, the number of obstacles, the number of unique road types and the layout of the scenario.

At this early stage, we use a naive implementation calculating every possible path from the starting node to all possible goal nodes. This leads to an explosion of the number of traces for $\mathcal{S}_2$. To understand the reason for that, let us have a look at the following traces:

$$\tau_1 : (b_v \rightarrow b_v \rightarrow l_v)$$
$$\tau_2 : (b_v \rightarrow l_v \rightarrow l_v).$$

The graph approach distinguishes between the traces $\tau_{1,2}$ although they might (and in this case do) represent the same homotopy class. To introduce a topology grouping step after the graph search omitting those duplicate traces or to use branch-and-bound techniques will be the subject of future work.

The number of traces for $\mathcal{S}_2$ is much higher than for $\mathcal{S}_1$. Because $\mathcal{S}_2$ contains a pedestrian walk, we obtain four additional cells. We get a high number of adjacent cells leading up to a high number of possible traces for two reasons: First, the adjacency definition in [9] lets two cells be adjacent if the closures intersect at a single point, which trades a lower number of infeasible traces in a kinematic sense for lower computational costs in the adjacency calculation. Secondly, we do not distinguish between forward and backward driving, which also introduces additional connections between cells. Improving these two points in combination with the topology grouping step will reduce the number of traces. We list the computational costs for calculating a single trace using the Dijkstra algorithm, for which we set the weights of the graph to the time gaps introduced in [9]. Improving the weights of the graph will also be the subject of future work.

We observe for $\mathcal{S}_1$ with the smaller step size that a small number of traces get rejected. These traces represent maneuvers going to the left lane and then starting an overtaking maneuver on the right which consequently violate R_1. Running $\mathcal{S}_1$ with the coarser discretization does not allow such complex maneuvers, which is why all of the 16 traces are valid.

We chose Scenario $\mathcal{S}_2$ to demonstrate the necessity for our rule-checking approach. Roughly forty percent of the traces get rejected, because they violate the rules. Checking a single trace takes around 22ms per obstacle in our implementation.

TABLE V: Computation Results for Maneuver Verification.

	$\mathcal{S}_1$		$\mathcal{S}_2$		$\mathcal{S}_3$	
	Scenario Description					
Planning horizon	4s	4s	4s	4s	4s	4s
Step size	1s	0.5s	1s	0.5s	1s	0.5s
No. of obstacles	2	2	1	1	2	2
No. of road types	1	1	2	2	1	1
	Semantic Partitioning					
Graph Nodes	20	36	35	63	20	36
Graph Edges	32	70	124	248	40	80
	Maneuver Verification					
No. of all traces	16	672	139	75441	76	4240
No. of sat. traces	16	640	83	35531	61	1840
	Computation times [s]					
Partitioning	0.073	0.129	0.144	0.206	0.290	0.426
Graph generation	0.014	0.021	0.020	0.037	0.013	0.020
Dijkstra Search	0.001	0.000	0.000	0.000	0.001	0.000
Calculating costs	1e-05	9e-06	1e-05	1e-05	8e-06	1e-05
Verification of π^s	0.044	0.045	0.022	0.024	0.043	0.046

This could be further sped up as we currently create a model file for every trace that we check, save that on the hard drive and then call the model checker. Future work will focus on reducing the computational time.

Scenario $\mathcal{S}_3$, which depicts a scene from the popular NGSIM dataset, shall show the feasibility of our approach in more realistic scenarios. Compared to $\mathcal{S}_1$, it takes roughly three times as much time. Reasons for that are the arbitrary shape of the road and the number of lanes.

The verification of a trace τ for $\mathcal{S}_2$ takes about half the time compared to $\mathcal{S}_1$ and $\mathcal{S}_3$. The reason is that $\mathcal{S}_1$ and $\mathcal{S}_3$ consider two obstacles, which brings the necessity to check the rules for each obstacle in τ separately. We evaluated the runtime of a trace verification with ten rules instead of three, which did not change the runtime significantly. With the need to check potentially many traces, the model checker might seem like a bottleneck. However, model checking tools support a huge set of formulas. To tune the verification algorithm for online use in a car running at a high frequency, a model checker only supporting our limited formula could be implemented more efficiently.

We have not discussed sequential planning yet, which means the use of the maneuver verification approach in a receding horizon manner. A maneuver to overtake a vehicle on the right might be forbidden, but a maneuver to only change to the right lane might be legal. At the next planning step, passing the vehicle and changing back to the left lane might be feasible as well. Possible ways to cope with that could be to incorporate past states $s_{-1}, s_{-2}, ...$ into the trace checking. A detailed analysis of such effects is the subject of future work.

VIII. Conclusion and Future Work

To formalize traffic rules using LTL, we introduced a novel link between combinatorial behavior planning and model checking. We showed that LTL can indeed be used to verify

high-level maneuver plans. This may help shift part of the development from writing code to writing specifications. Although we discretize the continuous environment and evaluate the rules on a higher, more abstract level, the rules still hold on the continuous level due to the homotopy property of each trace.

We selected rules that cover only one other traffic participant. Future work will need to consider interactions with more than one road user. Further investigations will also concentrate on the improvement of the graph costs, where we currently use the notion of time gaps, a relatively sparse definition that yields many traces with the same costs. We briefly touched the limited expressiveness of the semantic space in Section VII-A, which we will try to extend in the future while carefully weighing increased computational complexity against expressiveness.

When expressing traffic rules in the real world, there is a difference between "should" and "must". Some rules must never be violated, others should not be violated except in emergency situations. Future work includes investigating the difference between soft and hard rules as well as developing a heuristic for a graph search algorithm that incorporates knowledge of previously checked traces.

As a probabilistic prediction will be essential for robust behavior planning, the approach needs to be extended to incorporate the prediction of multiple possible maneuvers. This could open the door for probabilistic model checkers. Alternatively, our method could be used as an additional safety layer which verifies a probabilistic behavior planner.

References

[1] International Organization for Standardization, *Iso/pas 21448:2019: road vehicles – safety of the intended functionality*, 2019.

[2] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 187–210, May 2018.

[3] C. Urmson, J. Anhalt, D. Bagnell, *et al.*, "Autonomous driving in urban environments: boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, Aug. 2008.

[4] M. Montemerlo, J. Becker, S. Bhat, *et al.*, "Junior: the stanford entry in the urban challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 569–597, Sep. 2008.

[5] J. Leonard, J. How, S. Teller, *et al.*, "A perception-driven autonomous urban vehicle," *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, Oct. 2008.

[6] T. Kessler and A. Knoll, "Multi vehicle trajectory coordination for automated parking," *IEEE Intelligent Vehicles Symposium*, pp. 2–7, 2017.

[7] D. Lenz, T. Kessler, and A. Knoll, "Tactical cooperative planning for autonomous vehicles using mcts," *IEEE Intelligent Vehicles Symposium*, no. Iv, pp. 1–7, Jun. 2016.

[8] C. Hubmann, J. Schulz, M. Becker, D. Althoff, and C. Stiller, "Automated driving in uncertain environments: planning with interaction and uncertain maneuver prediction," *IEEE Transactions on Intelligent Vehicles*, vol. 3, no. 1, pp. 5–17, Mar. 2018.

[9] F. Altché and A. De La Fortelle, "Partitioning of the free space-time for on-road navigation of autonomous ground vehicles," in *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017*, 2018, pp. 2126–2133.

[10] E. M. Wolff, U. Topcu, and R. M. Murray, "Optimization-based trajectory generation with linear temporal logic specifications," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2014, pp. 5319–5325.

[11] A. Rizaldi, F. Immler, B. Schürmann, and M. Althoff, "A formally verified motion planner for autonomous vehicles," in, Springer, Cham, Oct. 2018, pp. 75–90.

[12] C. I. Vasile, J. Tumova, S. Karaman, C. Belta, and D. Rus, "Minimum-violation scltl motion planning for mobility-on-demand," in *Proceedings - IEEE International Conference on Robotics and Automation*, IEEE, May 2017, pp. 1481–1488.

[13] E. Plaku and S. Karaman, "Motion planning with temporal-logic specifications: progress and challenges," *AI Communications*, vol. 29, no. 1, pp. 151–162, Aug. 2016.

[14] A. Rizaldi and M. Althoff, "Formalising traffic rules for accountability of autonomous vehicles," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, IEEE, Sep. 2015, pp. 1658–1665.

[15] A. Rizaldi, J. Keinholz, M. Huber, *et al.*, "Formalising and monitoring traffic rules for autonomous vehicles in isabelle/hol," in, Springer, Cham, 2017, pp. 50–66.

[16] R. Kohlhaas, D. Hammann, T. Schamm, and J. M. Zollner, "Planning of high-level maneuver sequences on semantic state spaces," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, IEEE, Sep. 2015, pp. 2090–2096.

[17] L. I. Reyes Castro, P. Chaudhari, J. Tumova, *et al.*, "Incremental sampling-based algorithm for minimum-violation motion planning," in *52nd IEEE Conference on Decision and Control*, IEEE, Dec. 2013, pp. 3217–3224.

[18] K. Esterle, P. Hart, J. Bernhard, and A. Knoll, "Spatiotemporal motion planning with combinatorial reasoning for autonomous driving," in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2018.

[19] J. Park, S. Karumanchi, and K. Iagnemma, "Homotopy-based divide-and-conquer strategy for optimal trajectory planning via mixed-integer programming," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1101–1115, Oct. 2015.

[20] Economic Commission for Europe Inland Transport Committee, *Convention on road traffic - done at vienna on 8 november 1968*, 1968.

[21] A. Cimatti, E. Clarke, E. Giunchiglia, *et al.*, "Nusmv 2: an opensource tool for symbolic model checking," in, Springer, Berlin, Heidelberg, 2002, pp. 359–364.

[22] M. Koschi and M. Althoff, "Spot: a tool for set-based prediction of traffic participants," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, Jun. 2017, pp. 1686–1693.