Efficiently Ensuring Traffic Rule Compliance of Motion Plans by Incorporating Scenario Knowledge

Florian Lercher, Paul Reisenberg, and Matthias Althoff

***Abstract*—Autonomous vehicles must obey the rules of the road to safely participate in road traffic. To enforce these rules during motion planning, they are often formalized in temporal logic. Such formalizations need to be very general to cover all possible traffic situations, resulting in large and complex logic formulas. During motion planning, however, we are usually confronted with a concrete scenario in which parts of the formulas may be irrelevant. Since specification-compliant motion planning under complex specifications is computationally challenging, we aim to simplify the traffic rules by removing these irrelevant parts. To this end, we first present a general algorithm that augments linear temporal logic formulas with scenario-specific knowledge. Then, we provide a method for extracting knowledge from traffic scenarios to augment traffic rules. We can formally guarantee that the augmented specification is equivalent to the original formula in the given scenario. Therefore, subsequent motion planning modules that handle temporal logic specifications need only consider the augmented formulas. We benchmark our approach in recorded real-world scenarios to demonstrate that it can significantly accelerate specification-compliant motion planning.**

***Index Terms*—Autonomous vehicles, temporal logic simplification, traffic rules, motion planning.**

I. Introduction

AUTONOMOUS vehicles need to comply with the rules of the road if they are to safely participate in road traffic. To formally guarantee compliance, we require a formal specification of these rules. While simple traffic rules, like collision avoidance, can be expressed using safety specifications, temporal logics have emerged as the language for specifying complex rules [1]. In particular, metric temporal logic (MTL) and linear temporal logic (LTL) have been used to formalize parts of the German [2]–[4] and Chinese [5] traffic law.

Specification-compliant motion planning aims to find a dynamically feasible trajectory for an autonomous vehicle that adheres to a given temporal logic specification. However, enforcing complex temporal logic specifications is computationally expensive, since they impose many discrete constraints that the planner needs to consider in addition to the continuous constraints arising from the vehicle dynamics [6]. Formalizing traffic rules in temporal logic often leads to complex specifications, since they have to cover every possible traffic situation. For example, a specification might need to distinguish many cases to determine the priorities at an intersection [4]. Thus, the motion planner might be too slow to provide a feasible trajectory in time when applying the formalized rules naively.

However, we are usually interested in obtaining a feasible trajectory for one specific traffic situation. In this case, the general traffic rule specifications can often be greatly simplified. As a simple example, the right before left rule (R-IN3 from [4, Tab. VI]) is always satisfied if we consider a situation in which no vehicles are approaching the intersection from our right. This makes the entire rule irrelevant for motion planning, as we will fulfill the rule regardless of the chosen trajectory. Therefore, we propose a novel algorithm to simplify traffic rule specifications by augmenting them using knowledge about the current traffic situation. We guarantee that any trajectory that complies with the augmented specifications also satisfies the original rules. Hence, subsequent motion planners only need to enforce the augmented rules, which can significantly accelerate planning. We demonstrate this using real-world scenarios.

A. Related Work

We review previous work on specification-compliant motion planning to understand how complex temporal logic specifications impact state-of-the-art algorithms. Then, we investigate existing techniques for simplifying temporal logic formulas.

1) Specification-Compliant Motion Planning: A simple approach to specification-compliant motion planning is combining a classical planner (e.g., [7], [8]) with a temporal logic monitor to reject violating trajectories [9]. However, this may lead to frequent replanning as the specification is only considered after motion planning. Approaches that consider the specification during planning are roughly divided into three classes: *sampling-based*, *optimization-based*, and *multilayer* methods. Subsequently, we describe each class in more detail.

Sampling-based methods randomly sample the state space to plan trajectories. Often, they employ variants of the RRT* algorithm [8] to explore the product of the dynamical system and an automaton obtained from the specification [10]–[15]. To ensure specification compliance, each sampled system state is labeled according to its satisfied atomic propositions and matched against the transitions of the specification automaton. Others encode the specification in the cost function of RRT* by leveraging the quantitative semantics of signal temporal logic (STL) [15]–[17]. Since the cost function needs to be evaluated very often, it should be efficient to compute.

In optimization-based approaches, the trajectory planning task is formulated as a constrained optimization problem, frequently using mixed-integer constraints to encode the specification and system dynamics [18]–[22]. These approaches

This work was supported by the German Research Foundation (DFG) under grants AL 1185/20-1 and GRK 2428, and by the German Federal Ministry of Education and Research (BMBF) within the Munich Cluster for the Future of Mobility in Metropolitan Regions (MCube) under grant 03ZU1105AA.

The authors are with the School of Computation, Information and Technology, Technical University of Munich, 85748 Garching, Germany. florian.lercher@tum.de, paul.reisenberg@tum.de, althoff@tum.de

This version of the contribution has been accepted for publication, but is not the Version of Record. The Version of Record is available online at: https://dx.doi.org/10.1109/TITS.2025.3645702

can suffer from complex specifications, as each predicate introduces new decision variables for each time step it needs to be evaluated. In [23], the authors alleviate this issue: They only need a new decision variable for each disjunctive subformula, i.e., each case the specification distinguishes. Other approaches such as [24], which is based on graphs of convex sets [25], or [26], which uses dynamic programming, rely on a specification automaton for their encoding.

Multilayer approaches guide a continuous motion planner using a discrete abstraction of the system dynamics. They first compute an abstract plan that adheres to the specification, and then search for a trajectory that concretizes the plan. For many approaches, the abstraction must preserve the labeling of the system states with atomic propositions [27]–[32]. Therefore, the more propositions appear in the specification, the finer the required abstractions. The authors of [33] check the feasibility of their abstract plans by solving constrained reachability problems. This allows them to work with abstractions of varying coarseness; the coarsest consists of only a single state. In [34], [35], the abstraction takes the form of an overapproximative reachability graph; the specification is considered by splitting the reachable sets according to the atomic propositions [34] or an automaton [35]. Finally, [36], [37] construct provably safe high-level controllers using purely spatial reasoning; temporal aspects are modeled by state transitions in the controllers.

2) Temporal Logic Simplification: Simplification methods aim to rewrite a given temporal logic formula so that it is more efficient to use in subsequent algorithms, while preserving its original meaning. Simplification is often used as a preprocessing step to expedite the translation of LTL to Büchi automata. For example, [38] provides a set of rewrite rules based on syntactically identified implications between subformulas. In [39], the authors define *pure eventuality* and *pure universality* formulas, i.e., LTL formulas with a specific syntactic structure, for which special simplifications are possible; [40] provides rewrite rules for *alternating* formulas. State-of-the-art model checkers like Spot [41] and Maude [42] implement many of these rules. Similar techniques exist for mission-time LTL [43]. Formula rewriting is also relevant for translation to ω-automata other than Büchi automata, e.g., [44] defines a normal form for LTL formulas that facilitates the translation to deterministic Rabin automata. Finally, [45] simplifies LTL formulas by considering their *implicants* and *implicates*. In contrast to our approach, these methods rely only on the formula itself and do not consider outside information.

Rewriting-based monitoring algorithms [46]–[48] process a Boolean trace step by step, rewriting the formula at each step so that it represents the remaining constraints for the next step. Thus, in a sense, they simplify the formula based on the information from the trace. However, they are unsuitable for our problem as we often do not have a full Boolean trace. In particular for traffic rules depending on the unknown behavior of the ego vehicle, we might not know whether an atomic proposition holds. Our approach can handle this uncertainty.

B. Contributions

Motivated by the availability of scenario-specific knowledge in autonomous driving, we approach LTL simplification from a new angle. Our key contributions are

- a novel, general algorithm to augment LTL formulas using outside knowledge in the form of dependencies between propositions that determines an equivalent but simpler formula, and
- a computationally efficient procedure to extract knowledge from traffic scenarios using a conservative approximation for the unknown behavior of the ego vehicle, which enables us to apply our augmentation algorithm to the traffic rule-compliant motion planning problem.

Serving as a preprocessing step for any motion planning algorithm that accepts an LTL specification, our approach uses knowledge extracted from the current traffic scenario to significantly lower the complexity of the input formula; in particular, it reduces the number of required predicate evaluations and case distinctions. Thereby, our approach can considerably reduce the computation times of subsequent motion planning modules. We demonstrate this by an extensive evaluation using recorded real-world interstate and intersection scenarios. Depending on the motion planning algorithm, our approach results in a speedup factor of 5.8 to 68 in the experiments we have conducted. Our approach requires the current traffic scenario as input; it is not our goal to simplify traffic rules independently of a concrete situation.

This paper is organized as follows: After discussing preliminaries and our problem statement in Sec. II, we present our solution concept in Sec. III. We introduce our knowledge representation in Sec. IV before explaining the general augmentation algorithm in Sec. V. In Sec. VI, we describe how we extract knowledge from traffic scenarios. Finally, we evaluate our approach in Sec. VII before drawing a conclusion in Sec. VIII.

II. Preliminaries and Problem Statement

We provide the necessary background on set operations before introducing LTL. After explaining our setup for autonomous driving and briefly discussing set-based reachability analysis, we present our problem statement.

A. Set Operations

For sets $\mathcal{A}$ and $\mathcal{B}$, their *Minkowski sum* $\mathcal{A} \oplus \mathcal{B}$ is given as $\{a+b \mid a \in \mathcal{A}, b \in \mathcal{B}\}$. We will write $\mathcal{A} \oplus b$ instead of $\mathcal{A} \oplus \{b\}$.

Let the natural numbers $\mathbb{N}_0$ be our fixed *discrete time domain*. We work with *intervals* over $\mathbb{N}_0$, admitting ∞ as the right endpoint. Thus, for $n, m \in \mathbb{N}_0$, we have $[n, m] := \{n, n+1, \dots, m\}$ and $[n, \infty) := \{n, n+1, \dots\}$ unless stated otherwise. We will use $\lfloor I \rfloor$ to denote the *lower bound* of a non-empty interval I.

B. Linear Temporal Logic

Let $\mathcal{AP}$ be a fixed set of *atomic propositions*. Since traffic rules often require explicit timing constraints [3]–[5], we introduce LTL [49] using a timed until operator $\mathbf{U}_I$ rather than the usual next $\mathbf{X}$ and untimed until $\mathbf{U}$ operators. An LTL *formula* φ over $\mathcal{AP}$ is constructed according to the grammar

$$\varphi ::= a \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \, \mathbf{U}_I \, \varphi_2,$$

where $a \in \mathcal{AP}$ and I is a non-empty interval over $\mathbb{N}_0$. Intuitively, $\varphi_1 \mathbf{U}_I \varphi_2$ demands that, within the interval I, φ_2 holds eventually and until then φ_1 is true. Formally, we interpret LTL formulas over *traces*, i.e., infinite sequences of mappings from $\mathcal{AP}$ to the Boolean values $\top$ and $\bot$. Given a trace $\tau = \langle \tau_0, \tau_1, \dots \rangle$ and an LTL formula φ, we inductively define the *satisfaction* of φ at time step $k \in \mathbb{N}_0$, written as $\tau, k \models \varphi$, by

$$\begin{aligned}
\tau, k &\models a && \iff \tau_k(a) = \top \\
\tau, k &\models \neg\varphi && \iff \tau, k \not\models \varphi \\
\tau, k &\models \varphi_1 \wedge \varphi_2 && \iff \tau, k \models \varphi_1 \text{ and } \tau, k \models \varphi_2 \\
\tau, k &\models \varphi_1 \mathbf{U}_I \varphi_2 && \iff \exists k' \in I \oplus k : \tau, k' \models \varphi_2 \text{ and} \\
& && \quad \forall k'' \in [\lfloor I \rfloor + k, k' - 1] : \tau, k'' \models \varphi_1.
\end{aligned}$$

Observe that $\mathbf{U}_I$ is similar to the until operator in mission-time LTL [50, Sec. 2]. We use the common abbreviations for truth $\top$[1], falsehood $\bot$, disjunction $\vee$, implication $\rightarrow$, and equivalence $\leftrightarrow$. Moreover, we define $\mathbf{F}_I \varphi := \top \mathbf{U}_I \varphi$ (finally) and $\mathbf{G}_I \varphi := \neg \mathbf{F}_I \neg\varphi$ (globally) as usual. According to the definition, the following equivalences hold:

$$\begin{aligned}
\tau, k &\models \mathbf{F}_I \varphi \iff \exists k' \in I \oplus k : \tau, k' \models \varphi \text{ and} \\
\tau, k &\models \mathbf{G}_I \varphi \iff \forall k' \in I \oplus k : \tau, k' \models \varphi.
\end{aligned}$$

We can convince ourselves that our timed until $\mathbf{U}_I$ is equally expressive as the standard combination of untimed until $\mathbf{U}$ and next $\mathbf{X}$ by arguing similarly as in [34, Sec. VI-A].

1) Positive Normal Form: Our algorithm requires that negations only occur directly in front of atomic propositions, i.e., we want our LTL formulas to be in *positive normal form* [51, Sec. 5.1.5]. To this end, we introduce the set of *literals*

$$\mathcal{L} := \mathcal{AP} \cup \{\neg a \mid a \in \mathcal{AP}\} \cup \{\top, \bot\}.$$

The negation of a literal is again a literal. An LTL formula φ in positive normal form adheres to the grammar

$$\varphi ::= \lambda \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \mathbf{G}_I \varphi \mid \varphi_1 \mathbf{U}_I \varphi_2,$$

where $\lambda \in \mathcal{L}$. As we can bring any LTL formula into positive normal form (cf. [51, Sec. 5.1.5]), we assume without loss of generality that all LTL formulas are in positive normal form.

2) Monitoring: In previous work, we used a *three-valued* semantics to monitor a trace with uncertainties against an LTL formula φ [52, Sec. 4.2]. This semantics is an extension of Kleene's three-valued propositional logic [53] to LTL. In addition to the Boolean values $\top$ and $\bot$, the three-valued semantics admits a third value $\dashv$ to denote that a result is *unknown*. So, the monitoring algorithm accepts a three-valued input trace $\tilde{\tau}$ and returns a three-valued sequence $\tilde{\mu}$ so that

$$\begin{aligned}
\tilde{\mu}_k = \top &\implies \tau, k \models \varphi \text{ and} \\
\tilde{\mu}_k = \bot &\implies \tau, k \not\models \varphi
\end{aligned} \tag{1}$$

for all $k \in \mathbb{N}_0$ and traces τ that agree with $\tilde{\tau}$ wherever it is not $\dashv$ (cf. [52, Theorem 1]). Even though the algorithm is originally defined for STL, adapting it to the discrete time setting of LTL is straightforward as presented in Appendix A.

[1]We purposefully use the same symbol as for the Boolean value representing truth here to simplify the notation.

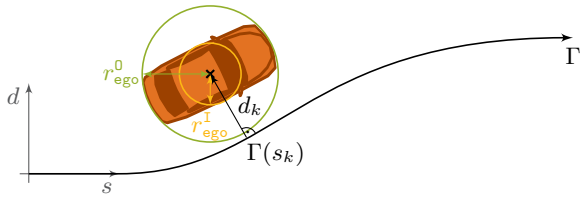


Fig. 1. The ego vehicle in the curvilinear coordinate system along the reference path Γ at time step k. The center of the ego vehicle is marked by a cross, and we show its inscribed and circumscribed circle.

C. Autonomous Driving Setup

We consider a *traffic scenario* $\mathfrak{S}$ with a finite *time horizon* $k_{\text{h}} \in \mathbb{N}_0$ and a fixed *time increment* $\Delta t \in \mathbb{R}_{>0}$. Thus, the discrete time step $k \in [0, k_{\text{h}}]$ corresponds to the continuous time $k\Delta t$. We assume that the scenario includes the initial state of the *ego vehicle*, i.e., the vehicle we control. Moreover, the scenario provides an environment model similar to that in [3], [4]. Most importantly, the environment model includes a description of the road network based on lanelets [54] and a behavior prediction for the other traffic participants. The type of prediction depends on the use case: For planning the intended motion of the ego vehicle, using a most likely trajectory is sufficient, while a set-based prediction [55] is more appropriate for determining fail-safe trajectories.

We assume that a *reference path* $\Gamma : \mathbb{R} \rightarrow \mathbb{R}^2$ for the ego vehicle is provided, which serves as the basis for a curvilinear coordinate system [56], [57]. As shown in Fig. 1, this coordinate system represents a Cartesian position by its *arc length* s along Γ and its *orthogonal deviation* d from $\Gamma(s)$.

We describe the dynamics of the ego vehicle constrained by its motion planner using the discrete-time model

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k),$$

where $\mathbf{x}_k \in \mathcal{X}_k$ represents the state of the ego vehicle and $\mathbf{u}_k \in \mathcal{U}_k$ the system input. Here, $\mathcal{X}_k \subseteq \mathbb{R}^{n_{\text{x}}}$ and $\mathcal{U}_k \subseteq \mathbb{R}^{n_{\text{u}}}$ are the feasible states and inputs at step $k \in [0, k_{\text{h}}]$; in particular, $\mathcal{X}_0$ denotes the set of possible initial states of the ego vehicle from $\mathfrak{S}$ including measurement uncertainties. A *trajectory* $\mathbf{x}_{[0,k_{\text{h}}]} := \langle \mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{k_{\text{h}}} \rangle$ is *feasible* if $\mathbf{x}_0 \in \mathcal{X}_0$ and

$$\forall k \in [0, k_{\text{h}} - 1] : \exists \mathbf{u}_k \in \mathcal{U}_k : \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) \in \mathcal{X}_{k+1}.$$

Given a state $\mathbf{x}$, we use $\text{proj}_{\square}(\mathbf{x})$ to denote its projection to the dimensions specified by $\square$, e.g., we have $\text{proj}_{(s,d)}(\mathbf{x}) = (s, d)^{\mathsf{T}}$ for $\mathbf{x} = (s, \dot{s}, d, \dot{d})^{\mathsf{T}}$. We lift the projection operator to sets of states, where it is applied element-wise. We denote the *occupancy* of the ego vehicle, i.e., its occupied positions in the curvilinear coordinate system, in the state $\mathbf{x}$ by $\text{occ}(\mathbf{x})$. Since the ego vehicle may have an arbitrary shape, we over-approximate the shape using its circumscribed circle with the radius $r^{\text{O}}_{\text{ego}}$ as shown in Fig. 1; to underapproximate the shape, we use its inscribed circle with the radius $r^{\text{I}}_{\text{ego}}$.

To express traffic rules as LTL formulas, each atomic proposition $a \in \mathcal{AP}$ is typically associated with a *predicate* p_a that returns a Boolean value [2]–[5]. The value of the predicate $p_a(\mathbf{x}_k, \mathfrak{S}_k)$ at time step $k \in [0, k_{\text{h}}]$ may depend on the state $\mathbf{x}_k$ of the ego vehicle and the state of the environment $\mathfrak{S}_k$

according to our environment model. Using these predicates, we can obtain the trace $\tau(\mathbf{x}_{[0,k_\mathrm{h}]})$ corresponding to the trajectory $\mathbf{x}_{[0,k_\mathrm{h}]}$, where $\tau(\mathbf{x}_{[0,k_\mathrm{h}]})_k$ is the mapping $a \mapsto p_a(\mathbf{x}_k, \mathfrak{S}_k)$ for $a \in \mathcal{AP}$ and $k \in [0, k_\mathrm{h}]$. A trajectory $\mathbf{x}_{[0,k_\mathrm{h}]}$ *satisfies* the formula φ if $\tau(\mathbf{x}_{[0,k_\mathrm{h}]}) \models \varphi$. Observe that $\tau(\mathbf{x}_{[0,k_\mathrm{h}]})$ is a finite trace, so we employ the finite LTL semantics from [47, Sec. 4.1] here, which infinitely repeats the last element of the trace. Further, note that the MTL traffic rules in [3], [4] admit an equivalent LTL formulation, since we consider scenarios with a fixed time step size [34, Sec. VI-A].

D. Set-Based Reachability Analysis

To soundly overapproximate the unknown behavior of the ego vehicle, we employ set-based reachability analysis. Intuitively, the reachable set at time step $k \in [0, k_\mathrm{h}]$ contains all states the ego vehicle could reach by following a feasible trajectory for k steps. Starting from $\mathcal{R}_0^\mathsf{e} := \mathcal{X}_0$, we define the exact *reachable set* of the ego vehicle as

$$\mathcal{R}_{k+1}^\mathsf{e} := \{\mathbf{x}_{k+1} \in \mathcal{X}_{k+1} \mid \exists \mathbf{x}_k \in \mathcal{R}_k^\mathsf{e} : \exists \mathbf{u}_k \in \mathcal{U}_k : \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k)\}.$$

For this paper, we assume that $\mathcal{R}^\mathsf{e}$ is subsumed by the reachable set of a point-mass model of the ego vehicle with its center as the reference point. The model is described in our curvilinear coordinate system along the reference path Γ. Hence, the state $\mathbf{x}_k = (s_k, \dot{s}_k, d_k, \dot{d}_k)^\mathsf{T}$ at time step $k \in [0, k_\mathrm{h}]$ consists of position and velocity in s- and d-direction; the input $\mathbf{u}_k = (\ddot{s}_k, \ddot{d}_k)^\mathsf{T}$ controls the respective accelerations. As stated in [34, Eq. (6)], the discrete-time system dynamics is

$$\mathbf{x}_{k+1} = \begin{pmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{x}_k + \begin{pmatrix} \frac{1}{2}\Delta t^2 & 0 \\ \Delta t & 0 \\ 0 & \frac{1}{2}\Delta t^2 \\ 0 & \Delta t \end{pmatrix} \mathbf{u}_k. \tag{2}$$

Since this model is linear, we can very efficiently compute an overapproximation $\mathcal{R}$ of its reachable set using set propagation techniques [58]. For our purposes, using intervals over $\mathbb{R}^4$ as set representation yields a sufficient accuracy. Now, we can formalize our assumption from above as

$$\forall k \in [0, k_\mathrm{h}] : \mathcal{R}_k \supseteq \mathrm{proj}_{(s,\dot{s},d,\dot{d})}(\mathcal{R}_k^\mathsf{e}). \tag{3}$$

This is a reasonable assumption, as many motion planners internally use a dynamics model subsumed by the curvilinear point-mass model with appropriate bounds on the velocities and accelerations [7], [34], [35], [59], [60]. If one does not wish to make this assumption, one can use more conservative overapproximations of $\mathcal{R}^\mathsf{e}$, e.g., in the style of [55].

E. Problem Statement

Suppose we want to enforce a traffic rule, given as an LTL formula φ, during motion planning for the ego vehicle in the scenario $\mathfrak{S}$. We aim to find an augmented formula $\mathcal{A}[\![\varphi]\!]_0$ that is equivalent to φ in $\mathfrak{S}$ but more efficient for motion planning. Formally, we require

$$\tau(\mathbf{x}_{[0,k_\mathrm{h}]}) \models \varphi \iff \tau(\mathbf{x}_{[0,k_\mathrm{h}]}) \models \mathcal{A}[\![\varphi]\!]_0 \tag{4}$$

for all feasible trajectories $\mathbf{x}_{[0,k_\mathrm{h}]}$ of the ego vehicle in the scenario $\mathfrak{S}$. Intuitively, this guarantees that any trajectory planned for the augmented formula also satisfies the original rule. At the same time, the augmentation does not exclude any potential solution, i.e., there is no feasible trajectory that satisfies the original rule but violates the augmentation.

Our problem statement gives rise to two subproblems:

- How can we extract knowledge from the scenario $\mathfrak{S}$ that helps us to simplify φ?
- How can we augment φ using this scenario-specific knowledge to obtain an equivalent but more efficient specification $\mathcal{A}[\![\varphi]\!]_0$?

We will answer the first question specialized to the domain of rule-compliant motion planning in autonomous driving and provide a general solution for the second problem.

When integrating the presented approach with our overall solution concept described in [61], real-time guarantees are not required to ensure safety due to the guaranteed availability of a fail-safe trajectory. Specifically, we prepare a verified fail-safe trajectory in every planning cycle that we fall back to if no feasible trajectory is found in time during the next cycle [61, Sec. VIII-C]; as we start driving from standstill in a safe state, the initial fail-safe trajectory can just remain in standstill.

III. Solution Concept

To explain our solution concept, let us consider the scenario in Fig. 2 in which we want to ensure that the ego vehicle $V_\mathtt{ego}$ keeps a safe following distance to the two other vehicles V_1 and V_2. To simplify exposition, we use an abridged version of the safe distance rule (R_G1 from [3, Tab. I]) that ignores cut-ins. If we naively instantiate this rule for V_1 and V_2, we obtain the formula $\mathbf{G}_{[0,\infty)}\, \varphi_\mathtt{sd}$, where $\varphi_\mathtt{sd}$ is defined as

$$\begin{aligned} &\big(\mathrm{in\_same\_lane}(V_\mathtt{ego}, V_1) \wedge \mathrm{in\_front\_of}(V_\mathtt{ego}, V_1) \\ &\quad \rightarrow \mathrm{keeps\_safe\_distance\_prec}(V_\mathtt{ego}, V_1)\big) \\ \wedge\ &\big(\mathrm{in\_same\_lane}(V_\mathtt{ego}, V_2) \wedge \mathrm{in\_front\_of}(V_\mathtt{ego}, V_2) \\ &\quad \rightarrow \mathrm{keeps\_safe\_distance\_prec}(V_\mathtt{ego}, V_2)\big). \end{aligned} \tag{5}$$

Intuitively, the rule requires that the ego vehicle maintains a safe distance to V_i whenever it drives behind V_i in the same lane for all $i \in \{1, 2\}$. According to [3, Sec. IV-C], a distance to V_i is considered safe, if the ego vehicle can come to standstill without hitting V_i, assuming that V_i fully brakes. Throughout the remainder of the article, we will use this example as our running example; thus, we abbreviate $\mathrm{in\_same\_lane}(V_\mathtt{ego}, V_i)$ as l_i, $\mathrm{in\_front\_of}(V_\mathtt{ego}, V_i)$ as f_i, and $\mathrm{keeps\_safe\_distance\_prec}(V_\mathtt{ego}, V_i)$ as s_i.

The formalized rule, as given above, is designed to work for any traffic situation. If we only want to apply it in the specific scenario shown in Fig. 2, parts of the formula are irrelevant. Thus, we want to extract knowledge from the scenario that helps us to identify and remove these irrelevant parts. For example, consider time step k_1 of our scenario (see Fig. 2a). At this time step, we can say with certainty that the ego vehicle is at a safe distance to V_2, since the reachable set, which contains all feasible future behaviors of $V_\mathtt{ego}$, is far enough behind V_2. Therefore, the second implication in $\varphi_\mathtt{sd}$ becomes

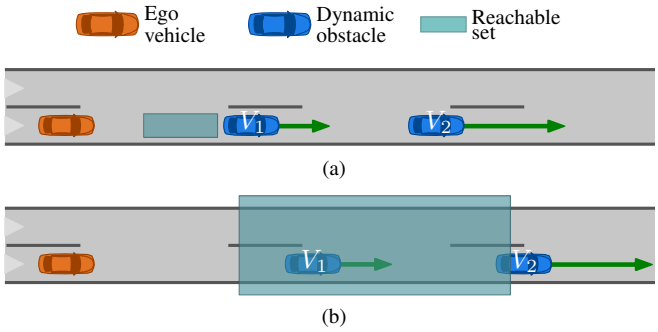


Fig. 2. An interstate scenario in which the ego vehicle needs to obey the safe distance rule. We show the reachable set of the ego vehicle and the obstacle predictions (the arrows indicate velocity) at time steps (a) k_1 and (b) k_2. The ego vehicle is always shown in its initial state.

irrelevant at this step, as it is always satisfied, meaning that we can safely omit constraints pertaining to V_2 during motion planning. Similarly, we can determine that the ego vehicle surely drives behind V_1 in the same lane. However, we cannot tell whether it is at a safe distance to V_1, as the reachable set is too close to V_1. During motion planning, it is thus sufficient to ensure that the ego vehicle unconditionally maintains a safe distance to V_1 at step k_1, i.e., we can simplify $\varphi_{\mathtt{sd}}$ to s_1 at this step.

At time step k_2 (see Fig. 2b), we cannot determine the value of any predicate with certainty, because the reachable set is too large. However, we can still utilize the relationships between the vehicles V_1 and V_2. For example, we know that V_2 is certainly in front of the ego vehicle if V_1 is also in front, i.e., $\mathrm{f}_1 \rightarrow \mathrm{f}_2$. Similarly, staying at a safe distance to V_1 automatically also maintains a safe distance to V_2 (so $\mathrm{s}_1 \rightarrow \mathrm{s}_2$), as V_1 follows V_2 and drives slower. Finally, we have $\mathrm{l}_1 \leftrightarrow \mathrm{l}_2$, because V_1 and V_2 are in the same lane. Using this knowledge, we can rewrite $\varphi_{\mathtt{sd}}$ at time step k_2 to

$$\varphi'_{\mathtt{sd}} := (\mathrm{l}_1 \wedge (\mathrm{f}_2 \wedge \mathrm{f}_1) \rightarrow \mathrm{s}_1) \wedge (\mathrm{l}_1 \wedge \mathrm{f}_2 \rightarrow (\mathrm{s}_1 \vee \mathrm{s}_2)). \quad (6)$$

Note that l_2 was replaced by l_1, since they are equivalent. Moreover, $\varphi'_{\mathtt{sd}}$ enables short-circuiting, which is not possible with $\varphi_{\mathtt{sd}}$. Short-circuiting means that the evaluation of a logical expression terminates as soon as the result is known, e.g., if the motion planner has already determined that it needs to enforce s_1, it does not need to consider s_2 at all, because $\mathrm{s}_1 \vee \mathrm{s}_2$ short-circuits if s_1 is true. We can also see that $\varphi'_{\mathtt{sd}}$ requires fewer case distinctions than $\varphi_{\mathtt{sd}}$ by looking at its minimal disjunctive normal form

$$\neg\mathrm{l}_1 \vee \neg\mathrm{f}_2 \vee \mathrm{s}_1 \vee (\neg\mathrm{f}_1 \wedge \mathrm{s}_2),$$

which distinguishes only four cases; the minimal disjunctive normal form of $\varphi_{\mathtt{sd}}$ requires nine cases.

To keep this example short, let us assume that, for the time steps $[0, k_1 - 1]$, we can extract the same knowledge as for k_1. Similarly, suppose that the knowledge from k_2 applies to $[k_1 + 1, k_2 - 1]$ as well. Then, our augmentation algorithm integrates the rewrites described above into the original formula $\mathbf{G}_{[0,\infty)}\, \varphi_{\mathtt{sd}}$ and returns

$$(\mathbf{G}_{[0,k_1]}\, \mathrm{s}_1) \wedge (\mathbf{G}_{[k_1+1,k_2]}\, \varphi'_{\mathtt{sd}}) \wedge (\mathbf{G}_{[k_2+1,\infty)}\, \varphi_{\mathtt{sd}}).$$

This formula is equivalent to $\mathbf{G}_{[0,\infty)}\, \varphi_{\mathtt{sd}}$ in the scenario from Fig. 2, but more efficient to use for specification-compliant motion planning algorithms, since it requires fewer predicate evaluations and case distinctions.

Since our knowledge representation and augmentation algorithm are independent of the application domain, we present them first in Sec. IV and V, respectively. In Sec. VI, we then discuss our knowledge extraction method that is specialized to the use case of rule-compliant motion planning.

IV. Knowledge Representation

The scenario-specific knowledge we used in Sec. III can be expressed as implications between literals at each time step. This motivates to represent our knowledge about a scenario as a sequence $\mathcal{K} = \langle \mathcal{K}_0, \mathcal{K}_1, \dots \rangle$ of *knowledge graphs*; we call $\mathcal{K}$ a time-varying *knowledge base*. For $k \in \mathbb{N}_0$, the nodes of $\mathcal{K}_k$ are the literals $\mathcal{L}$. The directed *edges* $\mathcal{E}(\mathcal{K}_k)$ of $\mathcal{K}_k$ represent implications between literals. A trace τ *conforms* with $\mathcal{K}$, written as $\tau \models \mathcal{K}$, if it honors the implications encoded in the edges of $\mathcal{K}_k$ for all $k \in \mathbb{N}_0$. Formally, we have $\tau \models \mathcal{K}$ if and only if

$$\forall k \in \mathbb{N}_0 : \forall (\lambda, \lambda') \in \mathcal{E}(\mathcal{K}_k) : (\tau_k \models \lambda \rightarrow \lambda').$$

Running Example. Recall from Sec. III that at time step k_2 we know that $\mathrm{s}_1 \rightarrow \mathrm{s}_2$, $\mathrm{f}_1 \rightarrow \mathrm{f}_2$, and $\mathrm{l}_1 \leftrightarrow \mathrm{l}_2$ hold. If we consider only the thick edges, Fig. 3a shows the knowledge graph $\tilde{\mathcal{G}}_{k_2}$ that represents this knowledge. Suppose that $\mathcal{K}$ is a knowledge base so that $\mathcal{K}_{k_2} := \tilde{\mathcal{G}}_{k_2}$. Then, no trace τ with the valuation of atomic propositions at time step k_2 given by

$$\tau_{k_2}(a) := \begin{cases} \top & \text{if } a \in \{\mathrm{s}_1, \mathrm{l}_2, \mathrm{f}_1, \mathrm{f}_2\} \\ \bot & \text{if } a \in \{\mathrm{s}_2, \mathrm{l}_1\} \end{cases}$$

can conform with $\mathcal{K}$ because $\mathrm{s}_1 \rightarrow \mathrm{s}_2$ and $\mathrm{l}_1 \leftrightarrow \mathrm{l}_2$ are violated at time step k_2. In contrast, a trace τ with

$$\tau_{k_2}(a) := \begin{cases} \top & \text{if } a \in \{\mathrm{s}_1, \mathrm{s}_2, \mathrm{f}_2\} \\ \bot & \text{if } a \in \{\mathrm{l}_1, \mathrm{l}_2, \mathrm{f}_1\} \end{cases}$$

conforms with $\mathcal{K}$, assuming there are no violations for time steps other than k_2.

We say that a knowledge graph $\mathcal{K}_k$ is *complete* if and only if it a) contains all trivial implications, i.e., $(\lambda, \top), (\bot, \lambda) \in \mathcal{E}(\mathcal{K}_k)$ for all $\lambda \in \mathcal{L}$, and b) is closed under application of the contrapositive, i.e., $(\lambda, \lambda') \in \mathcal{E}(\mathcal{K}_k)$ implies $(\neg\lambda', \neg\lambda) \in \mathcal{E}(\mathcal{K}_k)$. We can make any knowledge graph $\mathcal{K}_k$ complete by adding the missing edges.

Running Example. The edges added to make $\tilde{\mathcal{G}}_{k_2}$ complete are highlighted in Fig. 3a. Completing $\tilde{\mathcal{G}}_{k_2}$ results in the knowledge graph $\mathcal{G}_{k_2}$.

We lift the notion of completeness to knowledge bases $\mathcal{K}$ by requiring that $\mathcal{K}_k$ is complete for all $k \in \mathbb{N}_0$. Completing a knowledge base does not change the set of conforming traces, since all newly added edges are either trivial or logical consequences[2] of already existing edges.

[2] $\lambda \rightarrow \lambda'$ is logically equivalent to $\neg\lambda' \rightarrow \neg\lambda$.

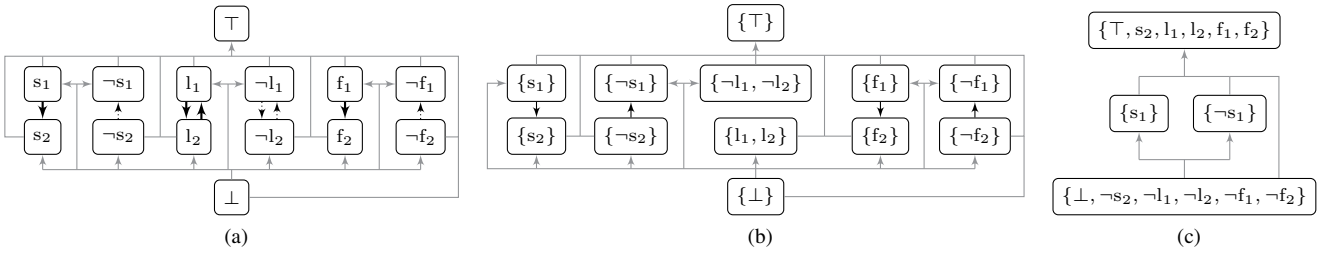


Fig. 3. Knowledge graphs for our running example from Sec. III. (a) The knowledge graph $\tilde{\mathcal{G}}_{k_2}$ at time step k_2 (thick edges only); to make $\tilde{\mathcal{G}}_{k_2}$ complete and obtain $\mathcal{G}_{k_2}$, we need to add the shown trivial and contrapositive edges (gray and dotted, respectively). (b) Component graph $\mathcal{G}_{k_2}^{\mathsf{SCC}}$ of $\mathcal{G}_{k_2}$. (c) Component graph $\mathcal{G}_{k_1}^{\mathsf{SCC}}$ of the knowledge graph $\mathcal{G}_{k_1}$ for time step k_1. In (b) and (c), we also show the trivial edges in gray.

V. Linear Temporal Logic Augmentation

Suppose φ is an LTL formula and $\mathcal{K}$ a time-varying knowledge base encoding our scenario-specific knowledge. Our goal is to augment φ using this knowledge. As shown in Sec. III, we want to rewrite φ under the assumption that all traces we will consider later conform with $\mathcal{K}$. Consequently, our rewrites need to preserve the meaning of the original φ only on these traces. This gives us some leeway to make the augmented formula more efficient for subsequent algorithms, e.g., by reducing the number of predicates and case distinctions involved.

The *augmentation* $\mathcal{A}[\![\varphi]\!]_k$ of a formula φ depends on the time step $k \in \mathbb{N}_0$ we are considering, e.g., in our running example from Sec. III, the augmentation of $\varphi_{\mathtt{sd}}$ at time step k_1 is s_1, but at time step k_2 it is $\varphi'_{\mathtt{sd}}$. Ultimately, we are only interested in the augmentation of the top-level formula φ at time step 0, i.e., $\mathcal{A}[\![\varphi]\!]_0$. To compute this augmentation, we may need the augmentations of the subformulas of φ at time steps $k > 0$, since the temporal operators look forward in time.

As explained at the beginning of this section, the augmentation of a formula has to preserve the meaning of the original formula on all traces conforming with our knowledge base $\mathcal{K}$. To formalize this, we fix an arbitrary trace τ so that $\tau \models \mathcal{K}$ and say that the augmentation $\mathcal{A}[\![\varphi]\!]_k$ of φ at time step $k \in \mathbb{N}_0$ is *sound* if and only if

$$\tau, k \models \varphi \iff \tau, k \models \mathcal{A}[\![\varphi]\!]_k.$$

Subsequently, we first define the augmentations $\mathcal{A}[\![\varphi]\!]_k$ by recursion on the structure of φ (Sec. V-A to V-C). We augment literals using the knowledge base directly, while the augmentations of compound formulas are composed of the augmentations of their subformulas. For all our definitions, we assume that straightforward simplifications involving $\top$ and $\bot$, e.g., rewriting $\bot \wedge \varphi$ to $\bot$, are performed implicitly. Each definition is accompanied by a lemma that proves its soundness, assuming that the recursive calls provide sound augmentations. These lemmas each show one case of an inductive proof that our recursive definition produces sound augmentations (Sec. V-D). Finally, we present an efficient algorithm for computing the augmentation of a formula with respect to a knowledge base (Sec. V-E).

A. Literals

Let us start with the base case of augmenting literals and fix a time step $k \in \mathbb{N}_0$. Since edges of $\mathcal{K}_k$ represent implications, the literals $\lambda, \lambda' \in \mathcal{L}$ are logically equivalent at step k, if we can reach λ from λ' and λ' from λ in $\mathcal{K}_k$. Generalizing this observation, all literals belonging to the same strongly connected component[3] of $\mathcal{K}_k$ must be equivalent. We thus collapse each strongly connected component into a single node by computing the *component graph* $\mathcal{K}_k^{\mathsf{SCC}}$ of $\mathcal{K}_k$ [62, Sec. 22.5]. Each node of $\mathcal{K}_k^{\mathsf{SCC}}$ is a set of equivalent literals $\Lambda \subseteq \mathcal{L}$, and an edge $(\Lambda_1, \Lambda_2) \in \mathcal{E}(\mathcal{K}_k^{\mathsf{SCC}})$ indicates that every literal in Λ_2 is true if any literal in Λ_1 is true. We formalize this in the lemma below and provide a proof in Appendix B.

Lemma 1. *Let $\mathcal{K}$ be a knowledge base and τ a trace so that $\tau \models \mathcal{K}$. For all $k \in \mathbb{N}_0$, the following statements hold:*

1) *For all nodes Λ of $\mathcal{K}_k^{\mathsf{SCC}}$, we have $\tau_k \models \lambda \leftrightarrow \lambda'$, where $\lambda, \lambda' \in \Lambda$ are arbitrary.*
2) *For all $(\Lambda_1, \Lambda_2) \in \mathcal{E}(\mathcal{K}_k^{\mathsf{SCC}})$, we have $\tau_k \models \lambda_1 \to \lambda_2$, where $\lambda_1 \in \Lambda_1$ and $\lambda_2 \in \Lambda_2$ are arbitrary.*

Running Example. Fig. 3b displays the component graph $\mathcal{G}_{k_2}^{\mathsf{SCC}}$ of $\mathcal{G}_{k_2}$ from Fig. 3a. Note that every node of the component graph contains a set of literals; most importantly l_1 and l_2 form a strongly connected component of $\mathcal{G}_{k_2}$ and thus belong to the same node in $\mathcal{G}_{k_2}^{\mathsf{SCC}}$. The same applies to $\neg\mathrm{l}_1$ and $\neg\mathrm{l}_2$. For later examples, we also show the component graph $\mathcal{G}_{k_1}^{\mathsf{SCC}}$ of the knowledge graph $\mathcal{G}_{k_1}$ at time step k_1 in Fig. 3c.

All literals belonging to the same node in $\mathcal{K}_k^{\mathsf{SCC}}$ are equivalent. Thus, we enforce only the literal with the lowest complexity from each node Λ in $\mathcal{K}_k^{\mathsf{SCC}}$, because this already ensures that all other literals from Λ hold; we call this literal the *representative* of Λ. Hence, it makes sense to replace a literal $\lambda \in \mathcal{L}$ by the representative of its node in $\mathcal{K}_k^{\mathsf{SCC}}$, i.e.,

$$\operatorname*{arg\,min}_{\lambda' \in \mathrm{node}(\mathcal{K}_k^{\mathsf{SCC}}, \lambda)} (c(\lambda')), \tag{7}$$

where $\mathrm{node}(\mathcal{K}_k^{\mathsf{SCC}}, \lambda)$ denotes the node of the component graph that contains the literal $\lambda \in \mathcal{L}$, and c measures the complexity of enforcing a literal in subsequent algorithms. Any reasonable complexity measure should be 0 for $\top$ and $\bot$, and positive for every other literal.

Moreover, we want to leverage the implications encoded in the edges of $\mathcal{K}_k^{\mathsf{SCC}}$. It is only necessary to enforce a literal $\lambda \in \mathcal{L}$, if none of the literals that imply λ have already been enforced. Hence, we collect the representatives of all nodes

[3] A *strongly connected component* is a maximal subgraph in which each node can reach every other node [62, Appendix B.4].

that reach $\mathrm{node}(\mathcal{K}_k^{\mathtt{SCC}}, \lambda)$ in a disjunction to enable short-circuiting. We arrive at the following augmentation for literals:

$$\mathcal{A}[\![\lambda]\!]_k := \bigvee_{\Lambda \in \mathrm{reaching}(\mathcal{K}_k^{\mathtt{SCC}}, \lambda)} \arg\min_{\lambda' \in \Lambda}(c(\lambda')), \tag{8}$$

where $\mathrm{reaching}(\mathcal{K}_k^{\mathtt{SCC}}, \lambda)$ denotes the set of nodes from which there exists a path to $\mathrm{node}(\mathcal{K}_k^{\mathtt{SCC}}, \lambda)$ in $\mathcal{K}_k^{\mathtt{SCC}}$; observe that $\mathrm{node}(\mathcal{K}_k^{\mathtt{SCC}}, \lambda) \in \mathrm{reaching}(\mathcal{K}_k^{\mathtt{SCC}}, \lambda)$ by definition. Note that $\mathrm{reaching}(\mathcal{K}_k^{\mathtt{SCC}}, \lambda)$ is a set of sets of literals since a node represents a set of literals. We can efficiently precompute the reaching nodes using, e.g., the Floyd-Warshall algorithm [62, Sec. 25.2]. Even though (8) increases the size of the formula, it can make the formula more efficient to use since it may reduce the number of cases to distinguish; we demonstrate this in Sec. VII-B3.

Running Example. Let $\mathcal{K}$ be a time-varying knowledge base so that $\mathcal{K}_k := \mathcal{G}_{k_1}$ for $k \in [0, k_1]$, $\mathcal{K}_k := \mathcal{G}_{k_2}$ for $k \in [k_1+1, k_2]$, and $\mathcal{E}(\mathcal{K}_k) := \emptyset$ for $k \in [k_2+1, \infty)$, i.e., the knowledge base we informally defined at the end of Sec. III. Moreover, suppose $c(\top) := 0$, $c(\bot) := 0$, and $c(\lambda) := 1$ for $\lambda \in \mathcal{L} \setminus \{\top, \bot\}$. Applying (8), we obtain

$$\mathcal{A}[\![\mathrm{s}_2]\!]_k = \begin{cases} \top \vee \mathrm{s}_1 \vee \neg \mathrm{s}_1 \vee \bot & \text{if } k \in [0, k_1] \\ \mathrm{s}_2 \vee \mathrm{s}_1 \vee \bot & \text{if } k \in [k_1+1, k_2] \\ \mathrm{s}_2 & \text{if } k \in [k_2+1, \infty) \end{cases}.$$

Note that we assume the first two cases are implicitly simplified to $\top$ and $\mathrm{s}_2 \vee \mathrm{s}_1$, respectively. Furthermore, we obtain

$$\mathcal{A}[\![\mathrm{l}_2]\!]_k = \begin{cases} \top \vee \mathrm{s}_1 \vee \neg \mathrm{s}_1 \vee \bot & \text{if } k \in [0, k_1] \\ \mathrm{l}_1 \vee \bot & \text{if } k \in [k_1+1, k_2] \\ \mathrm{l}_2 & \text{if } k \in [k_2+1, \infty) \end{cases},$$

if we break ties in complexity by alphabetical order.

Lemma 2. *Given a time step $k \in \mathbb{N}_0$, the augmentation $\mathcal{A}[\![\lambda]\!]_k$ for any literal $\lambda \in \mathcal{L}$ is sound.*

Proof. Let $k \in \mathbb{N}_0$ be arbitrary. We need to show that

$$\tau, k \models \lambda \iff \tau, k \models \mathcal{A}[\![\lambda]\!]_k.$$

Let us consider the forward direction first. We know that, at time step k, λ is equivalent to every literal that belongs to its node in $\mathcal{K}_k^{\mathtt{SCC}}$ according to Lemma 1. In particular, we find

$$\tau, k \models \lambda \iff \tau, k \models \arg\min_{\lambda' \in \mathrm{node}(\mathcal{K}_k^{\mathtt{SCC}}, \lambda)} (c(\lambda')).$$

Using that $\mathrm{node}(\mathcal{K}_k^{\mathtt{SCC}}, \lambda) \in \mathrm{reaching}(\mathcal{K}_k^{\mathtt{SCC}}, \lambda)$ by definition, we continue our derivation to establish the forward direction:

$$\begin{aligned} \dots &\implies \tau, k \models \bigvee_{\Lambda \in \mathrm{reaching}(\mathcal{K}_k^{\mathtt{SCC}}, \lambda)} \arg\min_{\lambda' \in \Lambda}(c(\lambda')) \\ &\overset{(8)}{\iff} \tau, k \models \mathcal{A}[\![\lambda]\!]_k. \end{aligned}$$

For the backward direction, we assume $\tau, k \models \mathcal{A}[\![\lambda]\!]_k$. By (8), there must exist $\Lambda \in \mathrm{reaching}(\mathcal{K}_k^{\mathtt{SCC}}, \lambda)$ so that $\tau, k \models \arg\min_{\lambda' \in \Lambda}(c(\lambda'))$. Consequently, there is a path in $\mathcal{K}_k^{\mathtt{SCC}}$ from Λ to $\mathrm{node}(\mathcal{K}_k^{\mathtt{SCC}}, \lambda)$. By repeated application of Lemma 1 to the edges of this path, we find that τ_k satisfies a chain of implications that enables us to derive

$$\begin{aligned} \tau, k \models \arg\min_{\lambda' \in \Lambda}(c(\lambda')) &\implies \tau, k \models \arg\min_{\lambda' \in \mathrm{node}(\mathcal{K}_k^{\mathtt{SCC}}, \lambda)} (c(\lambda')) \\ &\iff \tau, k \models \lambda. \end{aligned}$$

□

B. Boolean Connectives

Next, we consider formulas that have a Boolean connective as their top-level operator. As the semantics of these formulas only depends on one time step, augmentation is straightforward after we recursively determine augmentations for the subformulas: We just combine the augmented subformulas using the Boolean connective. Thus, we define the augmentations for conjunctive and disjunctive formulas at step $k \in \mathbb{N}_0$ as

$$\mathcal{A}[\![\varphi_1 \wedge \varphi_2]\!]_k := \mathcal{A}[\![\varphi_1]\!]_k \wedge \mathcal{A}[\![\varphi_2]\!]_k \text{ and} \tag{9}$$

$$\mathcal{A}[\![\varphi_1 \vee \varphi_2]\!]_k := \mathcal{A}[\![\varphi_1]\!]_k \vee \mathcal{A}[\![\varphi_2]\!]_k. \tag{10}$$

Lemma 3. *Suppose $\mathcal{A}[\![\varphi_1]\!]_k$ and $\mathcal{A}[\![\varphi_2]\!]_k$ are sound augmentations for φ_1 and φ_2 at time step $k \in \mathbb{N}_0$. Then, $\mathcal{A}[\![\varphi_1 \wedge \varphi_2]\!]_k$ and $\mathcal{A}[\![\varphi_1 \vee \varphi_2]\!]_k$ are also sound.*

Proof. Using that $\mathcal{A}[\![\varphi_1]\!]_k$ and $\mathcal{A}[\![\varphi_2]\!]_k$ are sound augmentations, this immediately follows from the semantics of LTL. □

C. Temporal Connectives

Finally, we need to handle formulas whose top-level connective is a temporal operator, i.e., $\mathbf{G}_I$ or $\mathbf{U}_I$. Let us consider formulas of the form $\mathbf{G}_I \varphi$ first. Intuitively, we want that the augmentation of this formula at time step $k \in \mathbb{N}_0$ requires that the augmentation $\mathcal{A}[\![\varphi]\!]_{k'}$ of φ holds at all steps $k' \in I \oplus k$. If the augmentation of φ was constant over $I \oplus k$, we could realize this as

$$\mathbf{G}_I \, \mathcal{A}[\![\varphi]\!]_{\lfloor I \rfloor + k},$$

which uses only a single $\mathbf{G}_I$. In general, however, the augmentation might change several times within $I \oplus k$ as in the example from Sec. III. Hence, we determine a partitioning of $I \oplus k$ into subintervals $I_0, \dots, I_n$ so that the augmentation of φ is constant in all subintervals; we require the number of these subintervals to be minimal. Applying our idea from above, we arrive at the following definition:

$$\mathcal{A}[\![\mathbf{G}_I \, \varphi]\!]_k := \bigwedge_{i \in [0,n]} \mathbf{G}_{I_i \oplus -k} \, \mathcal{A}[\![\varphi]\!]_{\lfloor I_i \rfloor}. \tag{11}$$

Note that we need to back-shift I_i by k time steps in the globally operator to obtain an offset relative to k instead of 0.

Running Example. Let $\mathbf{G}_{[0,\infty)} \, \varphi_{\mathtt{sd}}$ be the abridged version of the safe distance rule as introduced in (5) and let our knowledge base $\mathcal{K}$ be defined as before. We find that

$$\mathcal{A}[\![\varphi_{\mathtt{sd}}]\!]_k = \begin{cases} \mathrm{s}_1 & \text{if } k \in [0, k_1] \\ \varphi'_{\mathtt{sd}} & \text{if } k \in [k_1+1, k_2] \\ \varphi_{\mathtt{sd}} & \text{if } k \in [k_2+1, \infty) \end{cases},$$

where $\varphi'_{\mathtt{sd}}$ is defined as in (6). Hence, we partition $[0, \infty)$ into the subintervals $[0, k_1]$, $[k_1+1, k_2]$, and $[k_2+1, \infty)$. Then, the augmentation $\mathcal{A}[\![\mathbf{G}_{[0,\infty)} \, \varphi_{\mathtt{sd}}]\!]_0$ of our traffic rule is

$$(\mathbf{G}_{[0,k_1]} \, \mathrm{s}_1) \wedge (\mathbf{G}_{[k_1+1,k_2]} \, \varphi'_{\mathtt{sd}}) \wedge (\mathbf{G}_{[k_2+1,\infty)} \, \varphi_{\mathtt{sd}}).$$

Lemma 4. *Given $k \in \mathbb{N}_0$, $\mathcal{A}[\![\mathbf{G}_I\,\varphi]\!]_k$ is a sound augmentation of $\mathbf{G}_I\,\varphi$ provided that $\mathcal{A}[\![\varphi]\!]_{k'}$ is sound for all $k' \in I \oplus k$.*

Proof. Let $k \in \mathbb{N}_0$ be arbitrary. Using the semantics of the globally operator and our assumption that $\mathcal{A}[\![\varphi]\!]_{k'}$ is sound for all $k' \in I \oplus k$, we derive

$$\begin{aligned}\tau, k \models \mathbf{G}_I\,\varphi &\iff \forall k' \in I \oplus k : \tau, k' \models \varphi \\ &\iff \forall k' \in I \oplus k : \tau, k' \models \mathcal{A}[\![\varphi]\!]_{k'} \\ &\iff \forall i \in [0, n] : \\ &\qquad \forall k' \in I_i : \tau, k' \models \mathcal{A}[\![\varphi]\!]_{\lfloor I_i \rfloor},\end{aligned}$$

where $I_0, \ldots, I_n$ are defined as in (11). The last step is correct because $\bigcup_{i \in [0,n]} I_i = I \oplus k$. Moreover, the augmentation of φ is constant in each I_i, so considering the lower bound $\lfloor I_i \rfloor$ is sufficient. By rewriting I_i as $(I_i \oplus -k) \oplus k$, it becomes clear that we can express the inner universal quantifier using $\mathbf{G}_{I_i \oplus -k}$; the outer quantifier becomes a conjunction:

$$\begin{aligned}\ldots &\iff \forall i \in [0, n] : \\ &\qquad \forall k' \in (I_i \oplus -k) \oplus k : \tau, k' \models \mathcal{A}[\![\varphi]\!]_{\lfloor I_i \rfloor} \\ &\iff \forall i \in [0, n] : \tau, k \models \mathbf{G}_{I_i \oplus -k}\, \mathcal{A}[\![\varphi]\!]_{\lfloor I_i \rfloor} \\ &\iff \tau, k \models \bigwedge_{i \in [0,n]} \mathbf{G}_{I_i \oplus -k}\, \mathcal{A}[\![\varphi]\!]_{\lfloor I_i \rfloor} \\ &\overset{(11)}{\iff} \tau, k \models \mathcal{A}[\![\mathbf{G}_I\,\varphi]\!]_k. \qquad \square\end{aligned}$$

Next, we consider formulas of the form $\varphi_1\,\mathbf{U}_I\,\varphi_2$. Similar to above, we want to recreate the semantics of $\varphi_1\,\mathbf{U}_I\,\varphi_2$ using the augmentations of its subformulas φ_1 and φ_2. Its augmentation at time step $k \in \mathbb{N}_0$ depends on the augmentations of φ_1 and φ_2 in the interval $I \oplus k$. Once more, we face the issue that the augmentations might not be constant over the entire interval. Analogously to before, we determine a minimal partitioning of $I \oplus k$ into subintervals $I_0, \ldots, I_n$ so that the augmentations of φ_1 and φ_2 are constant in all subintervals. If our original formula is to hold, there must exist at least one subinterval I_i so that φ_1 remains true until φ_2 holds within I_i. Moreover, it is required that φ_1 always holds in $I \oplus k$ before we reach I_i. To express the second requirement, we can reuse our augmentation for the globally operator. Thus, we define

$$\begin{aligned}\mathcal{A}[\![\varphi_1\,\mathbf{U}_I\,\varphi_2]\!]_k := \bigvee_{i \in [0,n]} &\mathcal{A}[\![\varphi_1]\!]_{\lfloor I_i \rfloor}\, \mathbf{U}_{I_i \oplus -k}\, \mathcal{A}[\![\varphi_2]\!]_{\lfloor I_i \rfloor} \\ &\wedge \mathcal{A}[\![\mathbf{G}_{[\lfloor I \rfloor, \lfloor I_i \rfloor - 1 - k]}\,\varphi_1]\!]_k.\end{aligned} \tag{12}$$

Like in (11), we need to back-shift I_i by k steps to adjust the offset of the temporal operators.

For the soundness proof of the until augmentation, we use the same basic ideas as for the globally operator. However, the proof becomes technically more involved due to the more complex semantics of until. Therefore, we only state the lemma here and defer its proof to Appendix B.

Lemma 5. *Given $k \in \mathbb{N}_0$, $\mathcal{A}[\![\varphi_1\,\mathbf{U}_I\,\varphi_2]\!]_k$ is a sound augmentation of $\varphi_1\,\mathbf{U}_I\,\varphi_2$ provided that $\mathcal{A}[\![\varphi_1]\!]_{k'}$ and $\mathcal{A}[\![\varphi_2]\!]_{k'}$ are sound for all $k' \in I \oplus k$.*

D. Soundness

We prove that our recursive definition from Sec. V-A to V-C yields sound augmentations.

Theorem 1. *Let φ be an LTL formula. For all time steps $k \in \mathbb{N}_0$, the augmentation $\mathcal{A}[\![\varphi]\!]_k$ computed by recursively applying Eq.* (8) *to* (12) *is a sound augmentation of φ. In other words, we have*

$$\tau, k \models \varphi \iff \tau, k \models \mathcal{A}[\![\varphi]\!]_k$$

for all time steps $k \in \mathbb{N}_0$ and traces $\tau \models \mathcal{K}$; in particular,

$$\tau \models \varphi \iff \tau \models \mathcal{A}[\![\varphi]\!]_0.$$

Proof. We proceed by structural induction on φ. In Lemma 2, we have already shown the base case where $\varphi = \lambda$ for some literal $\lambda \in \mathcal{L}$. For the inductive cases, we begin with the case $\varphi = \mathbf{G}_I\,\varphi'$. By the induction hypothesis, we know that $\mathcal{A}[\![\varphi']\!]_{k'}$ is sound for all $k' \in \mathbb{N}_0$. We apply Lemma 4 to derive that $\mathcal{A}[\![\mathbf{G}_I\,\varphi']\!]_k$ must be a sound augmentation of $\mathbf{G}_I\,\varphi'$ for all $k \in \mathbb{N}_0$, which concludes this case. The remaining cases are analogous, using Lemma 3 and 5 instead of Lemma 4. $\square$

E. Algorithm

In Alg. 1, we present an efficient algorithm for computing the augmentation of a formula φ with respect to a time-varying knowledge base $\mathcal{K}$. The main part of the algorithm is the recursive function AUGMENT. In addition to φ and $\mathcal{K}$ the function takes a set of time steps $\mathcal{T} \subseteq \mathbb{N}_0$ as input and ensures that the augmentation $\mathcal{A}[\![\varphi]\!]_k$ is computed at all time steps $k \in \mathcal{T}$. Initially, we set $\mathcal{T}$ to $\{0\}$ as the top-level formula needs to be enforced only in the initial time step. In the remainder of this section, we explain AUGMENT in more detail; Appendix C provides a theoretical complexity analysis of Alg. 1.

Shortcut via Monitoring (ll. 5–8): We use the three-valued monitoring algorithm introduced in Sec. II-B2 to shortcut the cases in which we can rewrite entire subformulas to $\top$ or $\bot$. Using monitoring is more efficient than computing the augmentations using Eq. (8) to (12) for two reasons:

- We need to handle only truth values instead of formulas.
- We can process entire time intervals at once instead of having to consider every time step individually.

For monitoring, we extract a three-valued trace from our knowledge base $\mathcal{K}$ (function TRACE3 in l. 5). In this trace, the value for an atomic proposition $a \in \mathcal{AP}$ at time step $k \in \mathbb{N}_0$ is $\top$ if $\top \in \mathrm{node}(\mathcal{K}_k^{\mathsf{SCC}}, a)$ and $\bot$ if $\bot \in \mathrm{node}(\mathcal{K}_k^{\mathsf{SCC}}, a)$; otherwise, it is $\dashv$. If the monitor returns the verdict $\top$ for a formula φ at time step k, we have $\tau, k \models \varphi$ for all traces τ so that $\tau \models \mathcal{K}$ according to (1). Thus, $\top$ is a sound augmentation of φ at time step k. Similarly, if the verdict is $\bot$, the formula never holds, and we can rewrite it to $\bot$. Finally, in l. 8, we limit $\mathcal{T}$ to the time steps with verdict $\dashv$, since we already know the augmentation for the other steps. Note that the monitoring algorithm from [52] is structurally recursive in φ, enabling us to reuse its intermediate results in our recursive calls.

Recursive Calls (ll. 9–12): We recursively compute the augmentations for all subformulas of φ. To ensure that all augmentations we need later are present, we have to adjust the set of required time steps based on the top-level operator in φ (l. 9). Here, the function INTERVAL yields I for formulas starting with $\mathbf{G}_I$ or $\mathbf{U}_I$, and $[0, 0]$ for all other formulas.

Algorithm 1 LTL Augmentation

Input: LTL formula φ, knowledge base $\mathcal{K}$
Output: Augmented LTL formula

1: COMPLETE($\mathcal{K}$) ▷ Make $\mathcal{K}$ complete according to Sec. IV
2: AUGMENT(φ, $\mathcal{K}$, $\{0\}$)
3: **return** $\mathcal{A}[\![\varphi]\!]_0$

4: **function** AUGMENT(φ, $\mathcal{K}$, $\mathcal{T}$)
5: $\tilde{\mu} \leftarrow$ MONITOR(φ, TRACE3($\mathcal{K}$))
6: $\mathcal{A}[\![\varphi]\!]_{\tilde{\mu}=\top} \leftarrow \top$ ▷ Set to $\top$ where $\tilde{\mu}_k = \top$
7: $\mathcal{A}[\![\varphi]\!]_{\tilde{\mu}=\bot} \leftarrow \bot$ ▷ Set to $\bot$ where $\tilde{\mu}_k = \bot$
8: $\mathcal{T} \leftarrow \{k \in \mathcal{T} \mid \mu_k = \dashv\}$

9: $\mathcal{T}' \leftarrow \mathcal{T} \oplus$ INTERVAL(φ)
10: **for** $\varphi' \in$ CHILDREN(φ) **do**
11: AUGMENT(φ', $\mathcal{K}$, $\mathcal{T}'$)
12: **end for**

13: **if** CHILDREN(φ) $= \emptyset$ **then**
14: $k_\mathrm{r} \leftarrow$ Last step in $\mathcal{T}$ where knowledge $\mathcal{K}$ changes
15: **else**
16: $k_\mathrm{r} \leftarrow$ Last step in $\mathcal{T}'$ where a child augmentation changes
17: **end if**
18: **for** $k \in \mathcal{T}$ **do** ▷ Iterate in ascending order
19: $\mathcal{A}[\![\varphi]\!]_k \leftarrow$ Eq. (8) to (12)
20: **if** $k \geq k_\mathrm{r}$ **then**
21: ▷ Assign the last value to all remaining steps
22: $\mathcal{A}[\![\varphi]\!]_{\mathcal{T} \cap [k+1,\infty)} \leftarrow \mathcal{A}[\![\varphi]\!]_k$
23: **return**
24: **end if**
25: **end for**
26: **end function**

Computing the Augmentation (ll. 13–25): We compute the augmentation of φ according to Eq. (8) to (12) for all time steps in $\mathcal{T}$. Note that this could be an infinite number of time steps. In practice, however, our time-varying knowledge base $\mathcal{K}$ will always start repeating the same knowledge graph after a finite number of time steps due to the finite time horizon k_h of the scenario. Consequently, the augmentations of literals also stop changing after this point. Reasoning inductively, the augmentations of compound formulas φ must also repeat after a finite number of steps, because $\mathcal{A}[\![\varphi]\!]_k$ only depends on the augmentation of the subformulas of φ at times $k' \geq k$ as we only use future temporal operators. We determine this time step $k_\mathrm{r} \leq k_\mathrm{h}$ in ll. 13–17. After passing k_r, we assign the last determined augmentation to all remaining steps (l. 22). Thus, it is important to iterate over $\mathcal{T}$ in ascending order.

VI. KNOWLEDGE EXTRACTION FOR TRAFFIC SCENARIOS

As presented in the previous section, our augmentation algorithm is independent of the application domain as long as we can provide scenario-specific knowledge in the form of a time-varying knowledge base. To make the algorithm applicable to the traffic rule-compliant motion planning problem, we show how to extract knowledge bases from traffic scenarios, using the traffic rules from [3, Tab. I] and [4, Tab. VI] as examples. If we want meaningful results, the knowledge base $\mathcal{K}$ must only contain implications that are actually true in our scenario $\mathfrak{S}$. Formally, we call $\mathcal{K}$ *sound* for a scenario $\mathfrak{S}$ if $\tau(\mathbf{x}_{[0,k_\mathrm{h}]}) \models \mathcal{K}$ for all feasible trajectories $\mathbf{x}_{[0,k_\mathrm{h}]}$ of the ego vehicle in $\mathfrak{S}$, where we extend $\tau(\mathbf{x}_{[0,k_\mathrm{h}]})$ to an infinite trace by repeating its last element, matching the finite LTL semantics used in Sec. II-C. By plugging the definitions from Sec. II-C into Theorem 1, we can prove the following corollary, showing that we satisfy the requirement (4) from our problem statement.

Corollary 1. *Suppose φ is an LTL formula and $\mathcal{K}$ is a sound knowledge base for the scenario $\mathfrak{S}$. Let $\mathcal{A}[\![\varphi]\!]_0$ be the augmentation of φ with respect to $\mathcal{K}$ at time step 0. For any feasible trajectory τ of the ego vehicle in $\mathfrak{S}$, we have*

$$\tau(\mathbf{x}_{[0,k_\mathrm{h}]}) \models \varphi \iff \tau(\mathbf{x}_{[0,k_\mathrm{h}]}) \models \mathcal{A}[\![\varphi]\!]_0.$$

To extract a knowledge base from a scenario $\mathfrak{S}$, we start with an empty time-varying knowledge base $\mathcal{K}$ that consists only of knowledge graphs without any edges. Then, we attempt to evaluate predicates occurring in our specification as far as possible, which allows us to add edges of the form $(\top, \lambda)$ and $(\lambda, \bot)$ (Sec. VI-A and VI-B). To handle predicates depending on the unknown behavior of the ego vehicle, we exploit that the reachable set $\mathcal{R}$ from Sec. II-D subsumes all feasible behaviors. Finally, we check the scenario against a list of predefined traffic situations in order to extract further implications between literals (Sec. VI-C).

Recall that our approach is intended as a preprocessing step that makes the LTL formula more efficient for subsequent algorithms. Therefore, we need to be mindful of its computational cost: Extracting knowledge and augmenting the formula should not take longer than using the original formula in the subsequent algorithms. Thus, we do not strive to extract all possible knowledge from the scenario, but try to obtain most of it with reasonable effort.

A. Ego-Independent Predicates

The results of *ego-independent* predicates are independent of the behavior of the ego vehicle and only require the given scenario $\mathfrak{S}$. For each time step $k \in [0, k_\mathrm{h}]$ and atomic proposition $a \in \mathcal{AP}$, we add the edge $(\top, a)$ to $\mathcal{K}_k$ if $p_a(\mathfrak{S}_k) = \top$; otherwise, if $p_a(\mathfrak{S}_k) = \bot$, we add $(a, \bot)$. Examples for ego-independent predicates from [3], [4] include $\mathrm{on\_access\_ramp}(V_\circ)$ and $\mathrm{turning\_right}(V_\circ)$, where $V_\circ$ denotes a vehicle other than the ego vehicle.

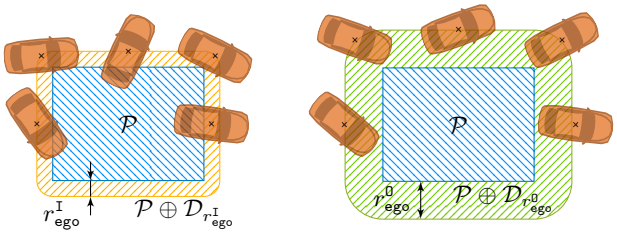
B. Ego-Dependent Predicates

In contrast to the ego-independent predicates, *ego-dependent* predicates may yield different results depending on the behavior of the ego vehicle. Since we do not know the exact behavior in advance, we evaluate these predicates against all feasible trajectories using our reachable set $\mathcal{R}$ from Sec. II-D. To evaluate an ego-dependent predicate p_a, we derive a three-valued version $\tilde{p}_a$ of p_a that accepts a set of ego vehicle states in place of a single state. The derived predicate returns a Boolean value if all provided states satisfy or violate p_a, and $\dashv$ otherwise. For each time step $k \in [0, k_\mathrm{h}]$ and atomic proposition $a \in \mathcal{AP}$, we extend the edge set of $\mathcal{K}_k$ by

$$\begin{cases} \{(\top, a)\} & \text{if } \tilde{p}_a(\mathcal{R}_k, \mathfrak{S}_k) = \top, \\ \{(a, \bot)\} & \text{if } \tilde{p}_a(\mathcal{R}_k, \mathfrak{S}_k) = \bot, \text{ and} \\ \emptyset & \text{if } \tilde{p}_a(\mathcal{R}_k, \mathfrak{S}_k) = \dashv. \end{cases}$$

TABLE I
DEPENDENCIES BETWEEN PREDICATES FOR PREDEFINED TRAFFIC SITUATIONS

Traffic Situation	**Condition**	**Added Implications**
Vehicle V_1 drives behind V_2.	$\mathrm{rear}(V_1) \leq \mathrm{rear}(V_2)$	$\mathrm{in\_front\_of}(V_{\mathtt{ego}}, V_1) \rightarrow \mathrm{in\_front\_of}(V_{\mathtt{ego}}, V_2)$
Vehicle V_1 drives faster than V_2.	$\mathrm{proj}_v(V_1) \geq \mathrm{proj}_v(V_2)$	$\mathrm{drives\_faster}(V_{\mathtt{ego}}, V_1) \rightarrow \mathrm{drives\_faster}(V_{\mathtt{ego}}, V_2)$
Vehicle V_1 occupies the same lanes as V_2.	$\mathrm{lanes}(V_1) = \mathrm{lanes}(V_2)$	$\mathrm{in\_same\_lane}(V_{\mathtt{ego}}, V_1) \leftrightarrow \mathrm{in\_same\_lane}(V_{\mathtt{ego}}, V_2)$
If vehicles V_1 and V_2 brake fully, V_2 reaches standstill further down the road than V_1.	$\mathrm{rear}(V_1) + \frac{\mathrm{proj}_v(V_1)^2}{2\lvert a^{\min}_{V_1}\rvert} \leq \mathrm{rear}(V_2) + \frac{\mathrm{proj}_v(V_2)^2}{2\lvert a^{\min}_{V_2}\rvert}$	$\mathrm{keeps\_safe\_distance\_prec}(V_{\mathtt{ego}}, V_1) \rightarrow \mathrm{keeps\_safe\_distance\_prec}(V_{\mathtt{ego}}, V_2)$
The reference path of vehicles V_1 and V_2 contains the same intersection lanelets.	$\{l_1 \in \mathrm{ref\_path\_lanelets}(V_1) \mid \mathrm{type}(l_1) = \mathit{intersection}\} = \{l_2 \in \mathrm{ref\_path\_lanelets}(V_2) \mid \mathrm{type}(l_2) = \mathit{intersection}\}$	$\mathrm{in\_intersection\_conflict\_area}(V_{\mathtt{ego}}, V_1) \leftrightarrow \mathrm{in\_intersection\_conflict\_area}(V_{\mathtt{ego}}, V_2)$



(a) (b)

Fig. 4. Evaluating predicates depending on the occupancy of the ego vehicle. (a) If the vehicle center (marked by a cross) is within $\mathcal{P} \oplus \mathcal{D}_{r^{\mathtt{I}}_{\mathtt{ego}}}$, the occupancy always intersects with $\mathcal{P}$. (b) If the center is outside $\mathcal{P} \oplus \mathcal{D}_{r^{\mathtt{0}}_{\mathtt{ego}}}$, the occupancy never intersects with $\mathcal{P}$.

Running Example. We define the three-valued version of the predicate $\mathrm{in\_front\_of}(V_{\mathtt{ego}}, V_{\mathtt{o}})$ [3, Sec. IV-A], which requires that the front-most point of the ego vehicle lies behind the rear-most point of the vehicle $V_{\mathtt{o}}$. Thus, if the rear position of $V_{\mathtt{o}}$ exceeds even the maximum possible front position of the ego vehicle, the predicate must always hold. Conversely, if even the minimum possible front position of the ego vehicle is larger than the rear position of $V_{\mathtt{o}}$, the predicate is definitely violated. Formally, the three-valued evaluation of the predicate at time step $k \in [0, k_{\mathtt{h}}]$ yields the value

$$\begin{aligned} &\top \quad \text{if } \max_{\mathbf{x} \in \mathcal{R}_k}(\mathrm{proj}_s(\mathbf{x})) + r^{\mathtt{0}}_{\mathtt{ego}} < \mathrm{rear}_k(V_{\mathtt{o}}), \\ &\bot \quad \text{if } \min_{\mathbf{x} \in \mathcal{R}_k}(\mathrm{proj}_s(\mathbf{x})) + r^{\mathtt{I}}_{\mathtt{ego}} \geq \mathrm{rear}_k(V_{\mathtt{o}}), \text{ and} \\ &\dashv \quad \text{otherwise,} \end{aligned}$$

where $\mathrm{rear}_k(V_{\mathtt{o}})$ denotes the s-coordinate of the rear-most point of $V_{\mathtt{o}}$ at step k. Here, we need to over- and underapproximate the shape of the ego vehicle by its circumscribed and inscribed circle, respectively, since $\mathcal{R}$ does not provide information about the orientation of the vehicle.

Our reachable set $\mathcal{R}$ only considers the center point of the ego vehicle. However, many traffic rule predicates depend on the occupancy of the ego vehicle, e.g., to determine which lanes it occupies [3], [4]. This often amounts to checking whether the occupancy $\mathrm{occ}(\mathbf{x})$ at a state $\mathbf{x}$ intersects with some set of curvilinear positions $\mathcal{P} \subseteq \mathbb{R}^2$. Since $\mathcal{R}$ lacks the orientation information required to determine the exact occupancy, we over- and underapproximate the shape of the ego vehicle by its circumscribed and inscribed circle, respectively. By bloating $\mathcal{P}$ according to these circles, we can perform our intersection check without explicitly determining the occupancy:

$$\begin{aligned} \mathrm{proj}_{(s,d)}(\mathbf{x}) \in \mathcal{P} \oplus \mathcal{D}_{r^{\mathtt{I}}_{\mathtt{ego}}} &\implies \mathrm{occ}(\mathbf{x}) \cap \mathcal{P} \neq \emptyset \text{ and} \\ \mathrm{proj}_{(s,d)}(\mathbf{x}) \notin \mathcal{P} \oplus \mathcal{D}_{r^{\mathtt{0}}_{\mathtt{ego}}} &\implies \mathrm{occ}(\mathbf{x}) \cap \mathcal{P} = \emptyset, \end{aligned}$$

where $\mathcal{D}_r \subseteq \mathbb{R}^2$ denotes the disk with radius r centered around the origin. We visualize this in Fig. 4. Consequently, the occupancy of the ego vehicle at time step $k \in [0, k_{\mathtt{h}}]$ certainly intersects with $\mathcal{P}$ if $\mathrm{proj}_{(s,d)}(\mathcal{R}_k) \subseteq \mathcal{P} \oplus \mathcal{D}_{r^{\mathtt{I}}_{\mathtt{ego}}}$; conversely, it is surely disjoint from $\mathcal{P}$ if $\mathrm{proj}_{(s,d)}(\mathcal{R}_k) \cap (\mathcal{P} \oplus \mathcal{D}_{r^{\mathtt{I}}_{\mathtt{ego}}}) = \emptyset$.

Running Example. Using this approach, we define the three-valued version of $\mathrm{in\_same\_lane}(V_{\mathtt{ego}}, V_{\mathtt{o}})$ [3, Sec. IV-A], which requires that the occupancy of the ego vehicle intersects with a lane occupied by the vehicle $V_{\mathtt{o}}$. At time step $k \in [0, k_{\mathtt{h}}]$, the three-valued predicate yields

$$\begin{aligned} &\top \quad \text{if } \mathrm{proj}_{(s,d)}(\mathcal{R}_k) \subseteq \mathrm{lanes}_k(V_{\mathtt{o}}) \oplus \mathcal{D}_{r^{\mathtt{I}}_{\mathtt{ego}}}, \\ &\bot \quad \text{if } \mathrm{proj}_{(s,d)}(\mathcal{R}_k) \cap (\mathrm{lanes}_k(V_{\mathtt{o}}) \oplus \mathcal{D}_{r^{\mathtt{0}}_{\mathtt{ego}}}) = \emptyset, \text{ and} \\ &\dashv \quad \text{otherwise,} \end{aligned}$$

where $\mathrm{lanes}_k(V_{\mathtt{o}})$ is the set of positions covered by at least one of the lanes occupied by $V_{\mathtt{o}}$ at time step k.

We have to consider numerical errors in the implementation. Therefore, we add safety margins so that the predicates conservatively return $\dashv$ when operating near their decision boundary.

C. Dependencies Between Predicates

Even when an ego-dependent predicate evaluates to $\dashv$ due to the uncertainty of the behavior of the ego vehicle, we can still encode its relationship with other predicates in our knowledge base. For example, if the vehicle V_1 drives behind another vehicle V_2 in our scenario $\mathfrak{S}$, we know that V_2 must be in front of the ego vehicle if V_1 is in front of it, as explained in Sec. III. Tab. I lists more of these traffic situations from which we can derive implications between predicates. Each entry consists of an informal description of the situation, a formal condition in terms of the environment model from [3], [4], and the implications added to the knowledge graph $\mathcal{K}_k$ when the condition is met at time step $k \in [0, k_{\mathtt{h}}]$. We refer the reader to [3, Sec. II and IV] and [4, Sec. III and IV] for a formal definition of the functions used in the conditions. We

do not claim that the list in Tab. I is exhaustive; it contains the situations we found simple to detect and frequent enough to justify checking for them. Adding more situations will improve the quality of the augmented formulas at the cost of increasing the computation time. It must be ensured that this cost does not outweigh the benefit of the improved augmentation for the planning algorithms in use.

VII. Evaluation

For our evaluation, we use the interstate and intersection traffic rules from [3], [4]. For each scenario, we first naively instantiate the traffic rules and then apply our approach to obtain an augmented version. We use the naive and augmented rules in two subsequent motion planning modules:

- a combination of the popular sampling-based motion planner from [7], which we will refer to as the *reactive planner*, with a traffic rule monitor [61, Sec. V] and
- the reachability analysis from [35], which computes specification-compliant reachable sets that serve as constraints narrowing the search space of a motion planner.

After briefly discussing the experiment setup, we benchmark our approach on recorded real-world scenarios from the exiD dataset [63]. Using scenarios from the inD dataset [64], we then demonstrate that our approach is also applicable to intersection traffic rules.

A. Experiment Setup

Our prototype implementation is integrated into the CommonRoad[4] [65] framework and available online[5]. In our experiments, we consider nominal motion planning and thus use most likely trajectories as predictions for the other traffic participants. The runtime measurements were performed on a laptop with an Intel Core i7-12700H 4.7 GHz processor; the reported computation times are averaged over five executions.

We assume that the shape of the ego vehicle is a rectangle with length 4.5 m and width 1.6 m. According to the physical limitations of a typical vehicle, we set the bounds for velocities and accelerations of our point-mass model (2) as follows:

$$-13.9\,\mathrm{m/s} \leq \dot{s} \leq 63.9\,\mathrm{m/s} \qquad -10.0\,\mathrm{m/s} \leq \dot{d} \leq 10.0\,\mathrm{m/s}$$
$$-9.5\,\mathrm{m/s^2} \leq \ddot{s} \leq 11.5\,\mathrm{m/s^2} \qquad -2.0\,\mathrm{m/s^2} \leq \ddot{d} \leq 2.0\,\mathrm{m/s^2}$$

Most traffic rules from [3], [4] must be evaluated with respect to all other vehicles in the field of view of the ego vehicle, which we assume to be a disk with a radius of 200 m. The traffic rule parameters are the same as in [3, Tab. II] and [4, Tab. VII]. We use the simplest possible literal complexity measure c (cf. Sec. V-A) with $c(\top) := 0$, $c(\bot) := 0$, and $c(\lambda) := 1$ for all $\lambda \in \mathcal{L} \setminus \{\top, \bot\}$. More sophisticated complexity measures could be obtained by statistical analysis of recorded datasets.

B. Interstate Scenarios

For our benchmark on interstate scenarios, we extracted 200 CommonRoad scenarios from the exiD dataset (location: Merzenich-Rather) [63]; an example scenario is shown in Fig. 5a. For these scenarios, we have $k_\mathrm{h} := 30$ and $\Delta t := 0.08\,\mathrm{s}$, which corresponds to a planning horizon of 2.4 s. We consider the safe distance rule (R_G1) and the entering vehicles rule (R_I5) from [3]. In comparison to our abridged version from Sec. III, R_G1 allows the safe distance to be violated for a brief time window after another vehicle cuts into the lane of the ego vehicle. Intuitively, R_I5 requires that the ego vehicle does not obstruct other vehicles entering the interstate via an access ramp by changing to the right-most lane of the main carriageway. We refer the reader to [3, Tab. I] for the formulation of the rules in temporal logic. We chose these two rules since they are the most complex rules from [3] with regard to the structure of the temporal logic formula; the remaining rules just require some propositional constraint to hold at every time step.

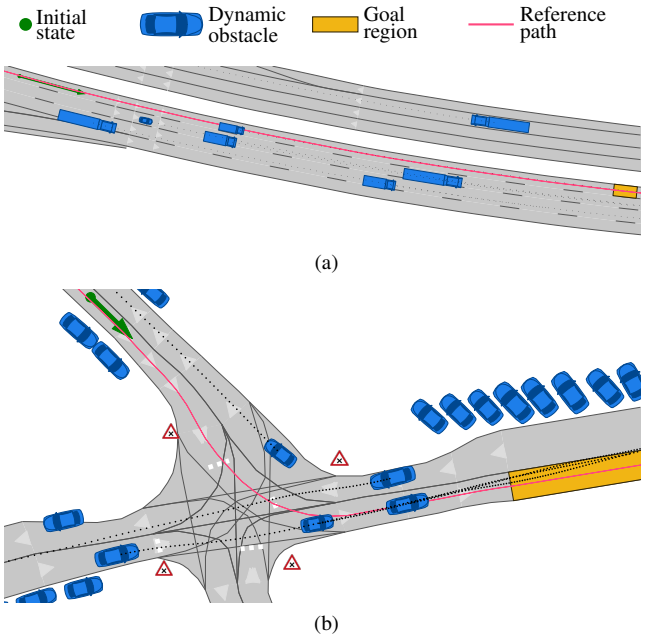


Fig. 5. Example scenarios from (a) the exiD dataset (location: Merzenich-Rather) and (b) the inD dataset (location: Frankenburg).

1) Trajectory Monitoring: To investigate the effects of our augmented rules on trajectory monitoring, we use the reactive planner to sample 100 dynamically feasible and collision-free trajectories. Then, we apply the traffic rule monitor to evaluate the rule compliance of each trajectory. Our computation time measurements do not include the time it takes to plan the trajectories because it is independent of the used traffic rules.

The boxplots in Fig. 6a compare the total computation time with naive and augmented traffic rules. For the augmented rules, the figure also breaks down the computation time by the three steps of the algorithm: extracting knowledge from the scenario (cf. Sec. VI), augmenting the formula with the extracted knowledge (cf. Alg. 1), and monitoring the planned trajectories against the augmented rules. The first two steps make up only 12 % of the total runtime on average. With an average computation time of 4.0 ms, Alg. 1 takes negligible effort in comparison to the knowledge extraction. Considering the total computation time, the augmented traffic rules accelerate the algorithm on average by factor 5.8 compared to

[4]https://commonroad.in.tum.de
[5]https://redirect.cps.cit.tum.de/traffic-rule-augmentation

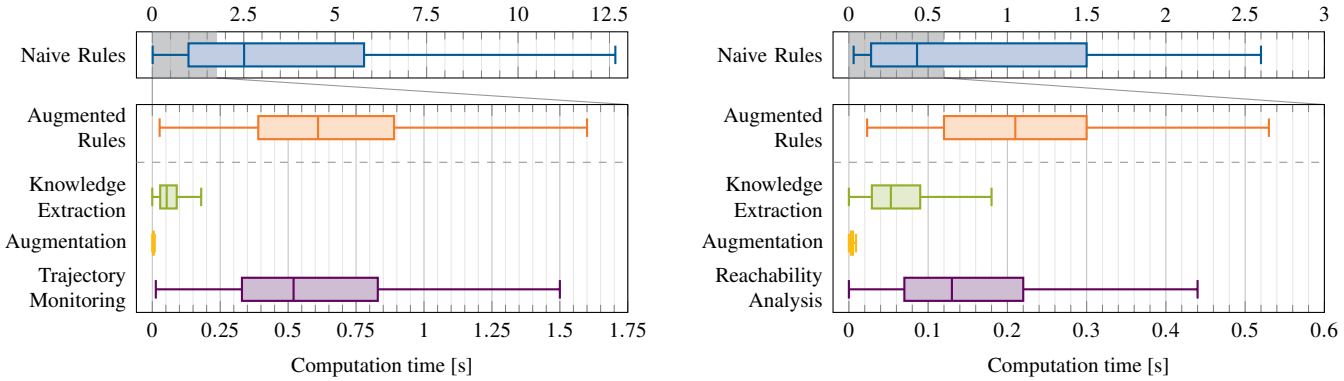


Fig. 6. Comparison of the computation time with naive and augmented traffic rules of (a) trajectory monitoring and (b) specification-compliant reachability analysis on scenarios from the exiD dataset (outliers and timeouts are not shown). Note that the bottom plots show a much smaller portion of the computation time axis than the top plots as indicated by the shaded areas.

using the naive rules. With the augmented rules the maximum observed runtime is 4.3 s, while it is 24 s for the naive rules.

For all trajectories, the monitoring verdicts on the augmented traffic rules match the verdicts for the naive rules. This confirms that (4) also holds in practice.

2) Specification-Compliant Reachability Analysis: To demonstrate that our augmented traffic rules benefit a variety of subsequent motion planning modules, we also apply them to the specification-compliant reachability analysis presented in [35]. For our experiments, we set a timeout of 20 s for the computation of the reachable sets. When deployed on a real vehicle, this timeout would be lower and trigger the execution of a previously computed fail-safe trajectory [61]. We choose a larger timeout value here to obtain meaningful results without requiring a highly optimized implementation of the reachability analysis.

When running the reachability analysis with the naive traffic rules, we observe a timeout in 124 of our 200 scenarios. Using the augmented rules, the algorithm completes in time for all but three of these scenarios corresponding to a reduction of the number of timeouts by 98 %. Fig. 6b, which is analogous to Fig. 6a, considers only the scenarios that did not time out. Unsurprisingly, we see very similar computation times for knowledge extraction and augmentation as in Sec. VII-B1; the two steps combined take 30 % of the total computation time here. In the 76 scenarios that did not time out for the naive rules, we measured an average speedup factor of 12 for the total computation time. If we make the best-case assumption that the scenarios that timed out for the naive rules, but not for the augmented rules, had a computation time equal to the timeout of 20 s, the average speedup factor increases to 68. Ignoring timeouts, the maximum observed runtime is 9.0 s for the augmented rules and 13 s for the naive rules.

3) Effect of Implications: In Sec. V-A, we claimed that using the implications encoded in the edges of the knowledge graph, in addition to equivalences, for the augmentation of literals can make the formula simpler to use despite increasing its length. We investigate this claim by modifying our augmentation algorithm to use (7) instead of (8) to determine the augmentations of literals. Using the augmented rules produced by this modified algorithm in the specification-compliant reachability analysis increases the total computation time by 21 % on average compared to the original augmented rules. However, we did not observe any significant change in the total computation time for trajectory monitoring. We attribute this to the fact that the monitor can often short-circuit the evaluation before the implications come into play, since it always works with a concrete trajectory. In contrast, the reachability analysis has to exhaustively explore all specification-compliant maneuvers. Thus, it can benefit considerably from the reduction of the involved case distinctions resulting from leveraging the implications (cf. Sec. III).

Based on the literature review in Sec. I-A1, we believe that these results transfer to other planning algorithms: Most existing algorithms scale with the number of predicates and the number of time steps they need to be evaluated at. Whenever we can rewrite an atomic proposition to $\top$, $\bot$, or an equivalent proposition that already appears elsewhere in the formula, these algorithms no longer need to consider the atomic proposition, thereby improving their performance. Intuitively, we can replace potentially many predicate evaluations during execution (e.g., to monitor different trajectories) with a single evaluation during knowledge extraction. In contrast, the implications from Tab. I make external dependencies between predicates explicit, which the algorithm needs to be capable of exploiting. Thus, equivalences are generally useful, while the benefit of implications depends on the characteristics of the subsequent planning algorithms.

C. Intersection Scenarios

Finally, we demonstrate that our approach is also beneficial for intersection traffic rules. To this end, we extracted 10 CommonRoad scenarios from the inD dataset (location: Frankenburg) [64] in which the ego vehicle must cross an unregulated intersection; Fig. 5b shows an example. We have $k_\mathrm{h} := 90$ and $\Delta t := 0.08\,\mathrm{s}$, so the planning horizon is 7.2 s.

We consider the stop sign rule (R-IN1), the right before left rule (R-IN3), the priority rule (R-IN4), and the turning

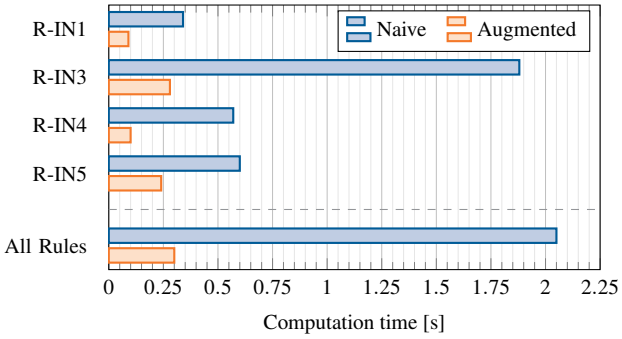


Fig. 7. Average total computation time of trajectory monitoring per traffic rule on scenarios from the inD dataset.

left rule (R-IN5) from [4, Tab. VI]. Since the intersection in our scenarios does not have any signs regulating the priority, it is intuitively clear that R-IN1 and R-IN4 are irrelevant. In contrast, R-IN3 will always be at least partially relevant.

We proceed as in Sec. VII-B1 except that we plan only 20 trajectories using the reactive planner. Fig. 7 shows the average total computation times broken down by individual rules. The computation times of the individual rules do not add up, since several rules share predicates and there is some overhead for starting the monitor. When evaluating all rules together, using the augmented rules results in an average speedup factor of 8.3. Knowledge extraction and augmentation account for 16 % of the total time; again, the time required by Alg. 1 (on average 3.0 ms) is negligible compared to the knowledge extraction (on average 58 ms). The highest observed runtime is 0.59 s for the augmented rules and 3.3 s for the naive rules. As in Sec. VII-B1, the verdicts on naive and augmented rules agree.

Our algorithm reduced the computation time for each individual rule, suggesting that applying it to the complete rule set makes sense. As expected, we observe the highest computation time for R-IN3. Moreover, we can simplify R-IN1 and R-IN4 to $\top$ in all scenarios, which is consistent with our intuition from above and explains their low computation times.

VIII. Conclusion

Our novel approach significantly lowers the complexity of traffic rule specifications that must be considered during traffic rule-compliant motion planning. In contrast to existing LTL simplification techniques, our augmentation algorithm leverages knowledge about the situation in which the formula is applied. Based on this knowledge, the algorithm removes the parts of the specification that are certainly irrelevant in the given situation. For the specific case of traffic rule-compliant motion planning, we have shown how such knowledge can be obtained from the concrete traffic scenario at hand. Since the output of our algorithm is just another LTL formula, it can be easily integrated with existing motion planners that accept LTL specifications. Our experiments show that the reduced complexity of the augmented specifications can considerably lower the computation time required by such planners.

Concerning future work, we are confident that our approach is transferable to other application domains where we need to determine a plan that is constrained by an LTL specification, such as robotics and power systems. Since our augmentation algorithm supports general LTL formulas, the main problem is finding and extracting suitable scenario-specific knowledge.

Appendix

A. Changes to the Three-Valued Monitoring Algorithm

The three-valued monitoring algorithm from [52, Sec. 4.2] is designed for STL, so some adjustments are required to apply it to LTL. Switching from the continuous time domain of STL to the discrete time domain of LTL is simple: As the algorithm is based on interval arithmetic, we just need to replace intervals over $\mathbb{R}_{\geq 0}$ with intervals over $\mathbb{N}_0$. Moreover, we need to accommodate for the different semantics of our temporal connectives $\mathbf{U}_I$ and $\mathbf{G}_I$. In [52], the semantics is encoded in *satisfaction sequences*[6] that indicate at which time steps a formula is satisfied. The three-valued semantics builds on top of recursively defined operators that compute the satisfaction sequence of a formula given the satisfaction sequences of its subformulas. We provide these operators for $\mathbf{U}_I$ and $\mathbf{G}_I$; for the propositional connectives, we reuse the operators of [52, Sec. 4.1] as their semantics is unchanged.

Defining the operator for $\mathbf{G}_I$ is simple: We observe that $\mathbf{G}_I\,\varphi$ is false at time step $k \in \mathbb{N}_0$ if and only if φ is violated somewhere in $I \oplus k$. We thus gather all time intervals in which φ is $\bot$ and back-shift them by I; the satisfaction sequence of $\mathbf{G}_I\,\varphi$ is $\bot$ in these intervals and $\top$ otherwise. For a Boolean sequence $\beta = \langle \beta_0, \beta_1, \dots \rangle$ and $k \in \mathbb{N}_0$, we define

$$(\mathbf{G}_I\,\beta)_k := \begin{cases} \bot & \text{if } k \in \bigcup_{i \in [0,n]} I_i \oplus -I \\ \top & \text{otherwise} \end{cases}, \tag{13}$$

where $I_0, \dots, I_n$ are the time intervals in which β is $\bot$, and $-I := \{-x \mid x \in I\}$. We show that $\mathbf{G}_I$ implements the semantics of the timed globally operator.

Lemma 6. *Let β be a Boolean sequence. For all $k \in \mathbb{N}_0$, $(\mathbf{G}_I\,\beta)_k$ is $\top$ if and only if $\beta_{k'} = \top$ for all $k' \in I \oplus k$; otherwise, $(\mathbf{G}_I\,\beta)_k$ is $\bot$. In particular, if β is the satisfaction sequence of φ, then $\mathbf{G}_I\,\beta$ is the satisfaction sequence of $\mathbf{G}_I\,\varphi$.*

Proof. We show the converse of the statement for $k \in \mathbb{N}_0$. We define $I_0, \dots, I_n$ as in (13) and derive:

$$\begin{aligned} (\mathbf{G}_I\,\beta)_k = \bot &\iff \exists i \in [0,n] : k \in I_i \oplus -I \\ &\iff \exists i \in [0,n] : \exists k' \in I \oplus k : k' \in I_i \\ &\iff \exists k' \in I \oplus k : \beta_{k'} = \bot. \end{aligned}$$

For the second to last step, we observe that $\mathcal{A} \oplus (-\mathcal{B})$ is the set $\{x \mid \exists a \in \mathcal{B} \oplus x : a \in \mathcal{A}\}$. The final step is correct as $\beta_{k'}$ is $\bot$ if and only if $k' \in I_i$ for some $i \in [0,n]$. □

Before we can define the operator for $\mathbf{U}_I$, we need to introduce the notion of unitary sequences (cf. [52, Sec. 2.3]). We say that a Boolean sequence β is *unitary*, if there exists exactly one interval I_β^+ so that $\beta_k = \top$ for all $k \in I_\beta^+$ and $\beta_k = \bot$ otherwise. Moreover, we need a helper function $\mathrm{ex}_{\mathbf{r}}(I)$ to expand an interval I by one to the right, which will serve

[6] These are called satisfaction *signals* in the original paper, since it works with continuous time instead of discrete time steps.

a similar purpose as the right-closure from [52, Sec. 2.1]: We define $\mathrm{ex}_{\mathtt{r}}([n,m]) := [n, m+1]$ and $\mathrm{ex}_{\mathtt{r}}([n,\infty)) := [n,\infty)$. Closely following [52, Sec. 4.1], we define our operator for $\mathbf{U}_I$. We first consider only unitary sequences and adjust [52, Eq. (1)] to our semantics: For unitary sequences β, β' and an interval I over $\mathbb{N}_0$, we define the unitary until $\overline{\mathbf{U}}_I$ so that $(\beta\, \overline{\mathbf{U}}_I\, \beta')_k = \top$ if and only if $k \in (I_1 \cup I_2) \cap \mathbb{N}_0$, where I_1 is

$$\Big[\Big(\big(I^+_{\beta'} \cap \mathrm{ex}_{\mathtt{r}}(I^+_\beta)\big) \oplus -\big((I \setminus \{\lfloor I \rfloor\}) \oplus -\lfloor I \rfloor\big)\Big) \cap I^+_\beta\Big] \oplus -\lfloor I \rfloor$$

and $I_2 := I^+_{\beta'} \oplus -\lfloor I \rfloor$. Arguing similarly to [52, Lem. 1], we can show that $\overline{\mathbf{U}}_I$ implements our timed until semantics for unitary sequences, i.e., for all $k \in \mathbb{N}_0$, we have

$$(\beta\, \overline{\mathbf{U}}_I\, \beta')_k = \begin{cases} \top & \text{if } \exists k' \in I \oplus k : \beta'_{k'} = \top \text{ and} \\ & \quad \forall k'' \in [\lfloor I \rfloor + k, k' - 1] : \beta_{k''} = \top \\ \bot & \text{otherwise} \end{cases}.$$

The key observation for the proof is that I_1 covers the cases in which $k' > \lfloor I \rfloor + k$ and I_2 handles the case $k' = \lfloor I \rfloor + k$, in which the universal quantifier is vacuously satisfied. We proceed as in [52, Lem. 2] to generalize $\overline{\mathbf{U}}_I$ to arbitrary Boolean sequences by decomposing them into unitary sequences.

We can convince ourselves that the operators we just defined are *monotone*, i.e., if their inputs are true at more time steps, their output will also be true at more steps. Thus, we proceed as in [52, Sec. 4.2] to derive the three-valued semantics.

B. Omitted Proofs

Lemma 1. *Let $\mathcal{K}$ be a knowledge base and τ a trace so that $\tau \models \mathcal{K}$. For all $k \in \mathbb{N}_0$, the following statements hold:*

1) *For all nodes Λ of $\mathcal{K}_k^{\mathsf{SCC}}$, we have $\tau_k \models \lambda \leftrightarrow \lambda'$, where $\lambda, \lambda' \in \Lambda$ are arbitrary.*
2) *For all $(\Lambda_1, \Lambda_2) \in \mathcal{E}(\mathcal{K}_k^{\mathsf{SCC}})$, we have $\tau_k \models \lambda_1 \to \lambda_2$, where $\lambda_1 \in \Lambda_1$ and $\lambda_2 \in \Lambda_2$ are arbitrary.*

Proof. Let $k \in \mathbb{N}_0$ be arbitrary. We begin with 1). Suppose $\lambda, \lambda' \in \Lambda$ for a node Λ of $\mathcal{K}_k^{\mathsf{SCC}}$. As λ and λ' belong to the same strongly connected component of $\mathcal{K}_k$, there is a path from λ to λ' in $\mathcal{K}_k$. Since $\tau \models \mathcal{K}$, τ_k satisfies a chain of implications from λ to λ'. Thus, we find $\tau_k \models \lambda \to \lambda'$, as implication is transitive. We can derive $\tau_k \models \lambda' \to \lambda$ analogously.

We continue with 2). Let $\lambda_1 \in \Lambda_1$ and $\lambda_2 \in \Lambda_2$ be arbitrary for an edge $(\Lambda_1, \Lambda_2) \in \mathcal{E}(\mathcal{K}_k^{\mathsf{SCC}})$. By the definition of the component graph [62, Sec. 22.5], this edge exists only if there are $\lambda'_1 \in \Lambda_1$ and $\lambda'_2 \in \Lambda_2$ so that $(\lambda'_1, \lambda'_2) \in \mathcal{E}(\mathcal{K}_k)$. Arguing like above, we find that $\tau_k \models \lambda_i \leftrightarrow \lambda'_i$ for all $i \in \{1, 2\}$. Moreover, we have $\tau_k \models \lambda'_1 \to \lambda'_2$, since $\tau \models \mathcal{K}$. Chaining these together, we obtain $\tau_k \models \lambda_1 \to \lambda_2$. □

Lemma 5. *Given $k \in \mathbb{N}_0$, $\mathcal{A}[\![\varphi_1\, \mathbf{U}_I\, \varphi_2]\!]_k$ is a sound augmentation of $\varphi_1\, \mathbf{U}_I\, \varphi_2$ provided that $\mathcal{A}[\![\varphi_1]\!]_{k'}$ and $\mathcal{A}[\![\varphi_2]\!]_{k'}$ are sound for all $k' \in I \oplus k$.*

Proof. Suppose $k \in \mathbb{N}_0$ is arbitrary. We derive:

$$\begin{aligned} \tau, k \models \varphi_1\, \mathbf{U}_I\, \varphi_2 &\iff \exists k' \in I \oplus k : \tau, k' \models \varphi_2 \text{ and} \\ &\qquad \forall k'' \in [\lfloor I \rfloor + k, k' - 1] : \tau, k'' \models \varphi_1 \\ &\iff \exists i \in [0, n] : \exists k' \in I_i : \tau, k' \models \varphi_2 \text{ and} \\ &\qquad \forall k'' \in [\lfloor I \rfloor + k, k' - 1] : \tau, k'' \models \varphi_1, \end{aligned}$$

where $I_0, \dots I_n$ are defined as in (12). The last step is correct since $\bigcup_{i \in [0,n]} I_i = I \oplus k$. Next, we split the universal quantifier into separate cases for $k'' \in I_i$ and $k'' \notin I_i$. The second case is independent of k', so we can express it using the globally operator:

$$\begin{aligned} \dots &\iff \exists i \in [0, n] : \exists k' \in I_i : \tau, k' \models \varphi_2 \text{ and} \\ &\qquad \forall k'' \in [\lfloor I_i \rfloor, k' - 1] : \tau, k'' \models \varphi_1 \\ &\qquad \text{and } \forall k'' \in [\lfloor I \rfloor, \lfloor I_i \rfloor - 1 - k] \oplus k : \tau, k'' \models \varphi_1 \\ &\iff \exists i \in [0, n] : \exists k' \in I_i : \tau, k' \models \varphi_2 \text{ and} \\ &\qquad \forall k'' \in [\lfloor I_i \rfloor, k' - 1] : \tau, k'' \models \varphi_1 \\ &\qquad \text{and } \tau, k \models \mathbf{G}_{[\lfloor I \rfloor, \lfloor I_i \rfloor - 1 - k]}\, \varphi_1. \end{aligned}$$

Applying our assumption and Lemma 4, we replace the formulas by their augmentations. Considering the augmentations of φ_1 and φ_2 at $\lfloor I_i \rfloor$ suffices as they are constant in each I_i:

$$\begin{aligned} \dots &\iff \exists i \in [0, n] : \exists k' \in I_i : \tau, k' \models \mathcal{A}[\![\varphi_2]\!]_{\lfloor I_i \rfloor} \text{ and} \\ &\qquad \forall k'' \in [\lfloor I_i \rfloor, k' - 1] : \tau, k'' \models \mathcal{A}[\![\varphi_1]\!]_{\lfloor I_i \rfloor} \\ &\qquad \text{and } \tau, k \models \mathcal{A}[\![\mathbf{G}_{[\lfloor I \rfloor, \lfloor I_i \rfloor - 1 - k]}\, \varphi_1]\!]_k \\ &\iff \exists i \in [0, n] : \tau, k \models \mathcal{A}[\![\varphi_1]\!]_{\lfloor I_i \rfloor}\, \mathbf{U}_{I_i \oplus -k}\, \mathcal{A}[\![\varphi_2]\!]_{\lfloor I_i \rfloor} \\ &\qquad \text{and } \tau, k \models \mathcal{A}[\![\mathbf{G}_{[\lfloor I \rfloor, \lfloor I_i \rfloor - 1 - k]}\, \varphi_1]\!]_k. \end{aligned}$$

In the second step, we express the combination of existential and universal quantifier using until. It becomes clear that this is correct if we rewrite I_i as $(I_i \oplus -k) \oplus k$. Turning the remaining existential quantifier into a disjunction concludes the proof:

$$\begin{aligned} \dots &\iff \tau, k \models \bigvee_{i \in [0,n]} \mathcal{A}[\![\varphi_1]\!]_{\lfloor I_i \rfloor}\, \mathbf{U}_{I_i \oplus -k}\, \mathcal{A}[\![\varphi_2]\!]_{\lfloor I_i \rfloor} \\ &\qquad \wedge \mathcal{A}[\![\mathbf{G}_{[\lfloor I \rfloor, \lfloor I_i \rfloor - 1 - k]}\, \varphi_1]\!]_k \\ &\overset{(12)}{\iff} \tau, k \models \mathcal{A}[\![\varphi_1\, \mathbf{U}_I\, \varphi_2]\!]_k. \end{aligned}$$

□

C. Complexity of the LTL Augmentation Algorithm

As shown in Sec. VII, the runtime of Alg. 1 is negligible compared to knowledge extraction for the application to rule-compliant motion planning. Nevertheless, a runtime complexity analysis is interesting from a theoretical standpoint. The complexity of Alg. 1 depends on the time horizon $k_{\mathtt{h}}$, the number of atomic propositions $|\mathcal{AP}|$, and the number of nodes $|\varphi|$ in the syntax tree of the formula as shown below.

Proposition 1. *The theoretical worst-case runtime complexity of Alg. 1 is $\mathcal{O}(k_{\mathtt{h}}|\mathcal{AP}|^3 + k_{\mathtt{h}}^2|\varphi|)$.*

Proof. We analyze the runtime complexity in three parts.

Preprocessing: Each knowledge graph in the knowledge base has $\mathcal{O}(|\mathcal{AP}|)$ nodes and $\mathcal{O}(|\mathcal{AP}|^2)$ edges after completion. We compute the component graph of each knowledge graph using the algorithm from [62, Sec. 22.5] in $\mathcal{O}(|\mathcal{AP}| + |\mathcal{AP}|^2)$. Further, we precompute the reaching nodes in the component graph (see Sec. V-A) using the Floyd-Warshall algorithm in $\mathcal{O}(|\mathcal{AP}|^3)$ [62, Sec. 25.2].[7] Assuming that the complexity c of a literal can be determined in constant time, we precompute the augmentation (8) of every literal in $\mathcal{O}(|\mathcal{AP}|)$. As the number of distinct knowledge graphs in the knowledge

[7] For sparse knowledge graphs, Johnson's algorithm performs better than the Floyd-Warshall algorithm [62, Sec. 25.3].

base is in $\mathcal{O}(k_\mathrm{h})$ and the Floyd-Warshall algorithm dominates the preprocessing of each graph, we arrive at an overall time complexity of $\mathcal{O}(k_\mathrm{h}|\mathcal{AP}|^3)$ for this step.

Monitoring: As mentioned in [52], the Boolean operators from Appendix A are closely related to those in [66], so their evaluation is linear in k_h [66, p. 157]. The three-valued monitoring algorithm applies these operators twice for each subformula, so it runs in $\mathcal{O}(k_\mathrm{h}|\varphi|)$. Alg. 1 executes MONITOR only once as it reuses the intermediate results (see Sec. V-E).

Augmentation: In AUGMENT, we recurse over all subformulas of φ yielding a factor of $\mathcal{O}(|\varphi|)$. As the monitoring result is precomputed, it is accessible in constant time. The crucial part is the loop over $\mathcal{T}$ in l. 18; all other parts are linear in k_h when representing the potentially infinite set $\mathcal{T}$ by a finite set of intervals. The loop yields a factor of $\mathcal{O}(k_\mathrm{h})$ since $k_\mathrm{r} \leq k_\mathrm{h}$ (see Sec. V-E). It remains to analyze l. 19. Due to our preprocessing, Eq. (8) to (10) are evaluated in constant time. In (11), we form a conjunction over the interval partitioning, requiring $\mathcal{O}(k_\mathrm{h})$ operations. The only relevant difference in (12) is that we use (11) inside the disjunction, which suggests a quadratic runtime at first glance. However, only the right bound of the interval of the globally operator increases in each iteration, allowing us to reuse results from previous iterations to achieve a complexity of $\mathcal{O}(k_\mathrm{h})$. Overall, AUGMENT thus has a time complexity of $\mathcal{O}(k_\mathrm{h}^2|\varphi|)$.

We obtain $\mathcal{O}(k_\mathrm{h}|\mathcal{AP}|^3 + k_\mathrm{h}^2|\varphi|)$ as the overall runtime complexity of Alg. 1 as augmentation dominates monitoring. □

REFERENCES

[1] N. Mehdipour, M. Althoff, R. D. Tebbens, and C. Belta, "Formal methods to comply with rules of the road in autonomous driving: State of the art and grand challenges," *Automatica*, vol. 152, 2023.

[2] K. Esterle, L. Gressenbuch, and A. Knoll, "Formalizing traffic rules for machine interpretability," in *Proc. of the IEEE Connected and Automated Vehicles Symp. (CAVS)*, 2020, pp. 1–7.

[3] 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 Symp. (IV)*, 2020, pp. 752–759.

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

[5] Y. Sun, C. M. Poskitt, J. Sun, Y. Chen, and Z. Yang, "LawBreaker: An approach for specifying traffic laws and fuzzing autonomous vehicles," in *Proc. of the IEEE/ACM Int. Conf. on Automated Software Engineering (ASE)*, 2023, pp. 1–12.

[6] E. Plaku and S. Karaman, "Motion planning with temporal-logic specifications: Progress and challenges," *AI Communications*, vol. 29, no. 1, pp. 151–162, 2015.

[7] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a Frenét frame," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2010, pp. 987–993.

[8] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[9] A. Rizaldi, J. Keinholz, M. Huber, J. Feldle, F. Immler, M. Althoff, E. Hilgendorf, and T. Nipkow, "Formalising and monitoring traffic rules for autonomous vehicles in Isabelle/HOL," in *Integrated Formal Methods (iFM)*, 2017, pp. 50–66.

[10] X. Luo, Y. Kantaros, and M. M. Zavlanos, "An abstraction-free method for multirobot temporal logic optimal control synthesis," *IEEE Transactions on Robotics*, vol. 37, no. 5, pp. 1487–1507, 2021.

[11] C. I. Vasile, X. Li, and C. Belta, "Reactive sampling-based path planning with temporal logic specifications," *The International Journal of Robotics Research*, vol. 39, no. 8, pp. 1002–1028, 2020.

[12] F. Penedo, C.-I. Vasile, and C. Belta, "Language-guided sampling-based planning using temporal relaxation," in *Proc. of the Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2020, pp. 128–143.

[13] F. S. Barbosa, L. Lindemann, D. V. Dimarogonas, and J. Tumova, "Integrated motion planning and control under metric interval temporal logic specifications," in *Proc. of the European Control Conference (ECC)*, 2019, pp. 2042–2049.

[14] L. I. Reyes Castro, P. Chaudhari, J. Tůmová, S. Karaman, E. Frazzoli, and D. Rus, "Incremental sampling-based algorithm for minimum-violation motion planning," in *Proc. of the IEEE Conf. on Decision and Control (CDC)*, 2013, pp. 3217–3224.

[15] J. Karlsson, F. S. Barbosa, and J. Tumova, "Sampling-based motion planning with temporal logic missions and spatial preferences," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15 537–15 543, 2020.

[16] A. Linard, I. Torre, E. Bartoli, A. Sleat, I. Leite, and J. Tumova, "Real-time RRT* with signal temporal logic preferences," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2023, pp. 8621–8627.

[17] F. S. Barbosa, D. Duberg, P. Jensfelt, and J. Tumova, "Guiding autonomous exploration with signal temporal logic," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3332–3339, 2019.

[18] A. Rodionova, L. Lindemann, M. Morari, and G. J. Pappas, "Combined left and right temporal robustness for control under STL specifications," *IEEE Control Systems Letters*, vol. 7, pp. 619–624, 2023.

[19] E. Aasi, C. I. Vasile, and C. Belta, "A control architecture for provably-correct autonomous driving," in *Proc. of the American Control Conf. (ACC)*, 2021, pp. 2913–2918.

[20] Z. Lin and J. S. Baras, "Optimization-based motion planning and runtime monitoring for robotic agent with space and time tolerances," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 1874–1879, 2020.

[21] Y. E. Sahin, R. Quirynen, and S. D. Cairano, "Autonomous vehicle decision-making and monitoring based on signal temporal logic and mixed-integer programming," in *Proc. of the American Control Conf. (ACC)*, 2020, pp. 454–459.

[22] E. M. Wolff, U. Topcu, and R. M. Murray, "Optimization-based trajectory generation with linear temporal logic specifications," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2014, pp. 5319–5325.

[23] V. Kurtz and H. Lin, "Mixed-integer programming for signal temporal logic with fewer binary variables," *IEEE Control Systems Letters*, vol. 6, pp. 2635–2640, 2022.

[24] ——, "Temporal logic motion planning with convex optimization via graphs of convex sets," *IEEE Transactions on Robotics*, vol. 39, no. 5, pp. 3791–3804, 2023.

[25] T. Marcucci, J. Umenberger, P. Parrilo, and R. Tedrake, "Shortest paths in graphs of convex sets," *SIAM Journal on Optimization*, vol. 34, no. 1, pp. 507–532, 2024.

[26] I. Papusha, J. Fu, U. Topcu, and R. M. Murray, "Automata theory meets approximate dynamic programming: Optimal control with temporal logic constraints," in *Proc. of the IEEE Conf. on Decision and Control (CDC)*, 2016, pp. 434–440.

[27] L. Lindemann and D. V. Dimarogonas, "Efficient automata-based planning and control under spatio-temporal logic specifications," in *Proc. of the American Control Conf. (ACC)*, 2020, pp. 4707–4714.

[28] C. K. Verginis, C. Vrohidis, C. P. Bechlioulis, K. J. Kyriakopoulos, and D. V. Dimarogonas, "Reconfigurable motion planning and control in obstacle cluttered environments under timed temporal tasks," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2019, pp. 951–957.

[29] K. Esterle, V. Aravantinos, and A. Knoll, "From specifications to behavior: Maneuver verification in a semantic state space," in *Proc. of the IEEE Intelligent Vehicles Symp. (IV)*, 2019, pp. 2140–2147.

[30] K. Cho, J. Suh, C. J. Tomlin, and S. Oh, "Cost-aware path planning under co-safe temporal logic specifications," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2308–2315, 2017.

[31] M. Lahijanian, M. R. Maly, D. Fried, L. E. Kavraki, H. Kress-Gazit, and M. Y. Vardi, "Iterative temporal planning in uncertain environments with partial satisfaction guarantees," *IEEE Transactions on Robotics*, vol. 32, no. 3, pp. 583–599, 2016.

[32] Y. Zhou, D. Maity, and J. S. Baras, "Timed automata approach for motion planning using metric interval temporal logic," in *Proc. of the European Control Conference (ECC)*, 2016, pp. 690–695.

[33] E. M. Wolff, U. Topcu, and R. M. Murray, "Automaton-guided controller synthesis for nonlinear systems with temporal logic," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2013, pp. 4332–4339.

[34] E. Irani Liu and M. Althoff, "Specification-compliant driving corridors for motion planning of automated vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 9, pp. 4180–4197, 2023.
[35] F. Lercher and M. Althoff, "Specification-compliant reachability analysis for autonomous vehicles using on-the-fly model checking," in *Proc. of the IEEE Intelligent Vehicles Symp. (IV)*, 2024, pp. 1484–1491.
[36] M. Schwammberger, "An abstract model for proving safety of autonomous urban traffic," *Theoretical Computer Science*, vol. 744, pp. 143–169, 2018.
[37] M. Schwammberger and G. Vaz Alves, "Extending urban multi-lane spatial logic to formalise road junction rules," in *Proc. of the Workshop on Formal Methods for Autonomous Systems (FMAS)*, 2021, pp. 1–19.
[38] F. Somenzi and R. Bloem, "Efficient Büchi automata from LTL formulae," in *Computer Aided Verification (CAV)*, 2000, pp. 248–263.
[39] K. Etessami and G. J. Holzmann, "Optimizing Büchi automata," in *Concurrency Theory (CONCUR)*, 2000, pp. 153–168.
[40] T. Babiak, M. Křetínský, V. Řehák, and J. Strejček, "LTL to Büchi automata translation: Fast and more deterministic," in *Proc. of the Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2012, pp. 95–109.
[41] A. Duret-Lutz, "LTL translation improvements in Spot 1.0," *International Journal of Critical Computer-Based Systems*, vol. 5, no. 1-2, pp. 31–54, 2014.
[42] S. Eker, J. Meseguer, and A. Sridharanarayanan, "The Maude LTL model checker," *Electronic Notes in Theoretical Computer Science*, vol. 71, pp. 162–187, 2004.
[43] C. Johannsen, B. Kempa, P. H. Jones, K. Y. Rozier, and T. Wongpiromsarn, "Impossible made possible: Encoding intractable specifications via implied domain constraints," in *Formal Methods for Industrial Critical Systems (FMICS)*, 2023, pp. 151–169.
[44] S. Sickert and J. Esparza, "An efficient normalisation procedure for linear temporal logic and very weak alternating automata," in *Proc. of the ACM/IEEE Symp. on Logic in Computer Science (LICS)*, 2020, pp. 831–844.
[45] I. P. de Guzmán, M. Ojeda-Aciego, and A. Valverde, "Implicates and reduction techniques for temporal logics," *Annals of Mathematics and Artificial Intelligence*, vol. 27, no. 1, pp. 3–23, 1999.
[46] P. Thati and G. Roşu, "Monitoring algorithms for metric temporal logic specifications," *Electronic Notes in Theoretical Computer Science*, vol. 113, pp. 145–162, 2005.
[47] G. Roşu and K. Havelund, "Rewriting-based techniques for runtime verification," *Automated Software Engineering*, vol. 12, no. 2, pp. 151–197, 2005.
[48] L. Zhao, T. Tang, J. Wu, and T. Xu, "Runtime verification with multi-valued formula rewriting," in *Proc. of the IEEE Int. Symp. on Theoretical Aspects of Software Engineering (TASE)*, 2010, pp. 77–86.
[49] A. Pnueli, "The temporal logic of programs," in *Proc. of the Annual Symp. on Foundations of Computer Science (SFCS)*, 1977, pp. 46–57.
[50] J. Li, M. Y. Vardi, and K. Y. Rozier, "Satisfiability checking for mission-time LTL (MLTL)," *Information and Computation*, vol. 289, 2022.
[51] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.
[52] F. Lercher and M. Althoff, "Using four-valued signal temporal logic for incremental verification of hybrid systems," in *Computer Aided Verification (CAV)*, 2024, pp. 259–281.
[53] S. C. Kleene, "On notation for ordinal numbers," *The Journal of Symbolic Logic*, vol. 3, no. 4, pp. 150–155, 1938.
[54] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *Proc. of the IEEE Intelligent Vehicles Symp. (IV)*, 2014, pp. 420–425.
[55] M. Althoff and S. Magdici, "Set-based prediction of traffic participants on arbitrary road networks," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 2, pp. 187–202, 2016.
[56] E. Héry, S. Masi, P. Xu, and P. Bonnifait, "Map-based curvilinear coordinates for autonomous vehicles," in *Proc. of the IEEE Int. Intelligent Transportation Systems Conf. (ITSC)*, 2017, pp. 1–7.
[57] G. Würsching and M. Althoff, "Robust and efficient curvilinear coordinate transformation with guaranteed map coverage for motion planning," in *Proc. of the IEEE Intelligent Vehicles Symp. (IV)*, 2024, pp. 2694–2701.
[58] M. Althoff, G. Frehse, and A. Girard, "Set propagation techniques for reachability analysis," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 369–395, 2021.
[59] G. Perantoni and D. J. N. Limebeer, "Optimal control for a Formula One car with variable parameters," *Vehicle System Dynamics*, vol. 52, no. 5, pp. 653–678, 2014.
[60] G. Würsching and M. Althoff, "Sampling-based optimal trajectory generation for autonomous vehicles using reachable sets," in *Proc. of the IEEE Int. Intelligent Transportation Systems Conf. (ITSC)*, 2021, pp. 828–835.
[61] M. Althoff, S. Maierhofer, G. Würsching, Y. Lin, F. Lercher, and R. Stolz, "No more traffic tickets: A tutorial to ensure traffic-rule compliance of automated vehicles," *Proceedings of the IEEE*, pp. 1–30, 2025, Early Access.
[62] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press, 2009.
[63] T. Moers, L. Vater, R. Krajewski, J. Bock, A. Zlocki, and L. Eckstein, "The exiD dataset: A real-world trajectory dataset of highly interactive highway scenarios in Germany," in *Proc. of the IEEE Intelligent Vehicles Symp. (IV)*, 2022, pp. 958–964.
[64] J. Bock, R. Krajewski, T. Moers, S. Runde, L. Vater, and L. Eckstein, "The inD dataset: A drone dataset of naturalistic road user trajectories at German intersections," in *Proc. of the IEEE Intelligent Vehicles Symp. (IV)*, 2020, pp. 1929–1934.
[65] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *Proc. of the IEEE Intelligent Vehicles Symp. (IV)*, 2017, pp. 719–726.
[66] O. Maler and D. Ničković, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems (FORMATS/FTRTFT)*, 2004, pp. 152–166.

Florian Lercher received the B.Sc. degree in informatics from the Technical University of Munich, Munich, Germany in 2020. In 2022, he received the M.Sc. degree in software engineering from the University of Augsburg, Augsburg, Germany, the Technical University of Munich, and the Ludwig-Maximilians-Universität, Munich, Germany. He joined the Cyber Physical Systems Group at Technical University of Munich as a research assistant in 2023, where he is currently working toward the Ph.D. degree under the supervision of Prof. Dr.-Ing. Matthias Althoff. His research interests include traffic rule-compliant motion planning for autonomous vehicles and formal verification of continuous and hybrid systems.

Paul Reisenberg received the B.Sc. degree in informatics and the B.Sc. degree in information systems from the Technical University of Munich, Munich, Germany, in 2024 and is currently pursuing the M.Sc. degree in computer science. In 2024, he joined the Cyber Physical Systems Group at Technical University of Munich as a bachelor thesis student under the supervision of Prof. Dr.-Ing. Matthias Althoff, working closely with Florian Lercher. His research at the group focuses on temporal logic simplification in the context of formalized traffic rules for autonomous vehicles.

Matthias Althoff received the diploma engineering degree in mechanical engineering and the Ph.D. degree in electrical engineering from the Technical University of Munich, Germany, in 2005 and 2010, respectively. He is currently an associate professor in computer science with the Technical University of Munich. From 2010 to 2012 he was a postdoctoral researcher with Carnegie Mellon University, Pittsburgh, USA, and from 2012 to 2013 an assistant professor at Ilmenau Technical University, Germany. His research interests include formal verification of continuous and hybrid systems, reachability analysis, planning algorithms, nonlinear control, automated vehicles, and power systems.