Specification-Compliant Reachability Analysis for Autonomous Vehicles Using On-the-Fly Model Checking

Florian Lercher and Matthias Althoff

***Abstract*— Compliance with the rules of the road is crucial for the safe operation of autonomous vehicles. Previous work has shown that one can expedite rule-compliant motion planning by constraining the search space based on the reachable states of the vehicle. We propose an algorithm to overapproximate the states that a vehicle can reach while adhering to a linear temporal logic specification. By integrating model checking into reachability analysis, we can exclude many non-compliant states early. Moreover, we only have to semantically split the reachable set when necessary to decide the validity of the specification. This significantly reduces the computation time compared to existing approaches. We benchmark our approach in recorded real-world scenarios to demonstrate its real-time capability.**

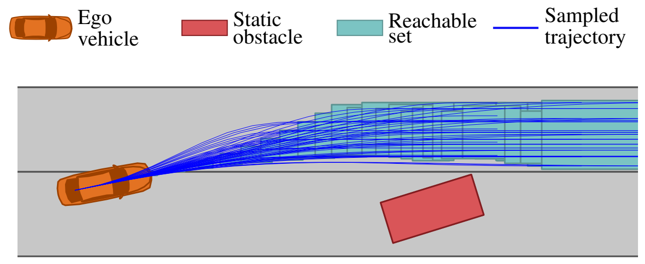


Fig. 1. Motivation: Sampling-based planning with reachable sets [6]. By sampling only within the reachable set, a feasible trajectory is found faster than with a naive approach (figure inspired by [6, Fig. 1b]).

I. Introduction

If autonomous vehicles are to be accepted by the broad public, they need to drive safely and thus comply with the rules of the road. To provide formal guarantees that an autonomous vehicle does not violate the rules of the road, they need to be formally specified. For the formalization of complex traffic rules, temporal logics have emerged as the language of choice [1]. In particular, [2], [3] formalize parts of the German traffic law in metric temporal logic (MTL); [4] uses linear temporal logic (LTL) for the same purpose.

Motion planning subject to temporal logic specifications is challenging, as it needs to respect both continuous constraints originating, e.g., from the vehicle dynamics, and discrete constraints arising, e.g., from the specifications [5]. By determining the reachable set of a vehicle, we can narrow the search space for specification-compliant trajectories, which facilitates the planning of intended [6] or fail-safe trajectories [7] (see Fig. 1). The *reachable set* comprises all states that the vehicle can reach while adhering to the specifications. The algorithm presented in [8] computes the reachable set subject to the simple specification "always avoid collisions." Irani Liu and Althoff [9] augment this algorithm to handle arbitrary LTL specifications. To this end, they split the reachable set with respect to every atomic proposition in the specification at each step of the reachability analysis. Thus, they obtain a semantically annotated reachability graph, from which they extract specification-compliant driving corridors using model checking.

Splitting the reachable set along every proposition at every step can significantly increase the number of sets to consider in subsequent steps, leading to an increased computational effort. However, some splits are unnecessary because, typically, only some propositions are relevant at a given step to decide the validity of the specification. For example, given the specification "X holds eventually", the propositions in X are only relevant until X holds for the first time. Moreover, since specification compliance is checked only after the reachability analysis, one might explore large parts of the state space that are only reachable when violating the specification. Given the specification "Y never holds", e.g., continuing to explore from states satisfying Y is pointless. To address these issues, we propose to perform model checking *on the fly*, i.e., while computing the reachable sets.

A. Related Work

We categorize existing techniques for motion planning subject to temporal logic specifications into three groups: *Sampling-based* and *optimization-based* methods directly plan specification-compliant trajectories in the continuous state space. *Multilayer* approaches, on the other hand, guide the motion planner using a discrete abstraction of the system.

Sampling-based approaches plan specification-compliant trajectories by randomly sampling the state space. Many convert the specification into an automaton and incrementally construct its product with the dynamical system using variants of the RRT* algorithm [10]–[14]. For signal temporal logic (STL) specifications, one can leverage the quantitative semantics in the cost function of RRT* to obtain maximally robust trajectories [11], [15]. Typically, the employed RRT* variants are *probabilistically complete* [16]; thus, they are only guaranteed to find a solution (assuming one exists) as the number of samples tends to infinity.

Optimization-based methods treat trajectory planning as a constrained optimization problem. To this end, many works encode the system dynamics and the specification as mixed-integer constraints [17]–[22]. The objective function then

This work was supported by the German Research Foundation (DFG) under grant numbers AL 1185/20-1 and GRK 2428.

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

captures robust satisfaction or soft constraints like favoring small control inputs. In [23], the authors formulate the planning problem as a dynamic program over the product of the dynamical system and the specification automaton.

Multilayer approaches determine a specification-compliant plan using a discrete abstraction of the dynamical system and then attempt to find a continuous trajectory obeying the discrete plan. However, the system dynamics are often not taken into account when constructing the abstraction, which leads to potentially infeasible plans [24]–[28]. The authors of [29] consider the system dynamics by providing control laws to realize the discrete transitions. Semantically annotated reachability graphs from [9] can also be viewed as discrete abstractions. As the graph is constructed using overapproximative reachability analysis, the inclusion of all dynamically feasible transitions is guaranteed, while many infeasible ones are excluded. The extracted driving corridor expedites trajectory planning by narrowing the search space.

While the authors of [30] do not directly address motion planning, they overapproximate the reachable set complying with an STL specification using a data-driven method. To this end, they constrain the reachable set at each step using *predicate functions* derived from the specification. In contrast, our approach leverages a specification automaton to determine the relevant propositions at each step.

B. Contributions

We propose a novel algorithm for overapproximating the reachable set of an autonomous vehicle subject to an LTL specification that encodes, e.g., traffic rules. Since we compute the reachable set for a finite time horizon, we interpret the specification as LTL over finite traces (LTL$_\mathrm{f}$) [31]. Our main contributions are:

- Performing model checking *on the fly*, i.e., during the reachability analysis (contrary to [9]).
- Tracking the progress toward specification satisfaction while computing the reachable sets to a) consider only the propositions relevant to decide whether the specification is satisfied, and b) exclude states that are unreachable when adhering to the specification.
- Considerably lower computation times compared to [9].

This paper is organized as follows: In Sec. II, we introduce preliminaries before presenting our problem statement. We explain our algorithm for specification-compliant reachability analysis in Sec. III. Finally, we evaluate our approach in Sec. IV before coming to a conclusion in Sec. V.

II. Preliminaries and Problem Statement

After introducing LTL$_\mathrm{f}$ and our vehicle model, we define specification-compliant reachable sets. Based on this definition, we provide a formal problem statement.

A. Linear Temporal Logic over Finite Traces

Let $\mathcal{AP}$ be a fixed set of atomic propositions. LTL$_\mathrm{f}$ shares the syntax of LTL, so an LTL$_\mathrm{f}$ *formula* φ over $\mathcal{AP}$ is constructed according to the grammar

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

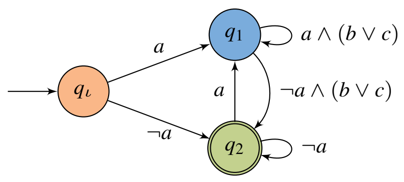


Fig. 2. Specification automaton for $\mathbf{G}(a \rightarrow \mathbf{X}(b \vee c))$ with $\mathcal{AP} = \{a, b, c\}$. We represent δ symbolically, so the edge $(q_1, \neg a \wedge (b \vee c), q_2)$ represents the transitions $(q_1, \{b\}, q_2)$, $(q_1, \{c\}, q_2)$, and $(q_1, \{b, c\}, q_2)$.

where $a \in \mathcal{AP}$ [31, Sec. 2]. LTL$_\mathrm{f}$ formulas are interpreted over *finite traces*, i.e., non-empty, finite words over the alphabet $2^{\mathcal{AP}}$. Intuitively, the *next* operator $\mathbf{X}$ requires φ to hold in the next state of the trace, while $\mathbf{U}$ requires that φ_1 holds *until* φ_2 becomes true. We use the common abbreviations $\mathbf{F}\,\varphi := \mathit{true}\,\mathbf{U}\,\varphi$ (finally) and $\mathbf{G}\,\varphi := \neg\,\mathbf{F}\,\neg\varphi$ (globally/always). Let $\mathcal{L}(\varphi)$ denote the set of all finite traces that satisfy the formula φ. We refer the reader to [31, Sec. 2] for the formal definition of the satisfaction relation, which requires in particular that all eventualities (like next and finally) are resolved before the end of the trace.

The literature provides several methods for translating an LTL$_\mathrm{f}$ formula φ into a (non-deterministic) finite automaton $\mathcal{A}_\varphi$ that accepts exactly the finite traces that satisfy φ, i.e., $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$. The *specification automaton* $\mathcal{A}_\varphi$ has the alphabet $2^{\mathcal{AP}}$ and the states $\mathcal{Q}$ with initial state $q_\iota \in \mathcal{Q}$ and final states $\mathcal{F} \subseteq \mathcal{Q}$. Its transition relation is $\delta \subseteq \mathcal{Q} \times 2^{\mathcal{AP}} \times \mathcal{Q}$. We opt for an indirect translation via regular LTL and Büchi automata (see, e.g., [32, Sec. V-A]) since with Spot [33] there is a mature tool to perform the conversion.[1] Our approach works best if we ensure that $\mathcal{A}_\varphi$ only has states that are part of some accepting run. Fig. 2 depicts an example automaton for $\varphi = \mathbf{G}(a \rightarrow \mathbf{X}(b \vee c))$ with the above property. As shown in Fig. 2, the transition relation induces a propositional formula $\psi_{q,q'}$ for each pair of states $q, q' \in \mathcal{Q}$ so that the automaton can transition from q to q' if and only if $\psi_{q,q'}$ is satisfied.

Besides requiring an explicit representation of the state space, our approach is independent of how the specification is converted into an automaton. Moreover, it can also handle specifications in other logics expressible as finite automata, e.g., syntactically co-safe LTL [34].

B. Vehicle Model

We introduce the index $k \in \mathbb{N}_0$ to refer to the discrete time step corresponding to the continuous time $t_k = k\Delta t$, where $\Delta t \in \mathbb{R}_{>0}$ is a fixed time increment. Without loss of generality, the initial time step is 0, and we denote the fixed final time step as the *planning horizon* k_h.

Following previous work [8], [9], we adopt a simple point-mass model of the ego vehicle with its center as the reference point, described in a curvilinear coordinate system [35] with path length s and lateral deviation d. The system state $\mathbf{x}_k = (s_k, \dot{s}_k, d_k, \dot{d}_k)^\mathsf{T} \in \mathcal{X}_k$ at time step k consists of position and velocity in s- and d-direction; the input $\mathbf{u}_k = (\ddot{s}_k, \ddot{d}_k)^\mathsf{T} \in \mathcal{U}_k$

[1]See `https://spot.lre.epita.fr/tut12.html` for details.

controls the respective accelerations. Here, $\mathcal{X}_k \subseteq \mathbb{R}^4$ and $\mathcal{U}_k \subseteq \mathbb{R}^2$ denote the set of admissible states and inputs at time step k, respectively. As stated in [9, Eq. (6)], the discrete-time system dynamics is

$$\mathbf{x}_{k+1} := \underbrace{\begin{pmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{pmatrix} \mathbf{x}_k + \begin{pmatrix} \frac{1}{2}\Delta t^2 & 0 \\ \Delta t & 0 \\ 0 & \frac{1}{2}\Delta t^2 \\ 0 & \Delta t \end{pmatrix} \mathbf{u}_k}_{f(\mathbf{x}_k, \mathbf{u}_k)} . \quad (1)$$

We conservatively bound the velocities and accelerations in both directions to account for the kinematic limitations of the vehicle and the effects of transforming the vehicle dynamics to the curvilinear coordinate system (see, e.g., [36]).

C. Specification-Compliant Reachable Sets

Let $\mathcal{X}_0$ be the measured set of initial states of the ego vehicle. Moreover, let $[n]$ denote the set $\{0, 1, \dots, n\}$ for $n \in \mathbb{N}_0$. A state trajectory $\mathbf{x}_{[k_\mathrm{h}]} := \langle \mathbf{x}_0, \dots, \mathbf{x}_{k_\mathrm{h}} \rangle$ is *drivable*, if it is a solution to the system dynamics (1) with initial state in $\mathcal{X}_0$. We define the set of all drivable trajectories as

$$\mathcal{X} := \{\mathbf{x}_{[k_\mathrm{h}]} \mid \mathbf{x}_0 \in \mathcal{X}_0 \wedge \forall k \in [k_\mathrm{h} - 1]\, \exists \mathbf{u}_k \in \mathcal{U}_k : \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) \in \mathcal{X}_{k+1}\}.$$

To define the specification-compliant reachable sets, we need to interpret a given LTL$_\mathrm{f}$ specification φ over a trajectory $\mathbf{x}_{[k_\mathrm{h}]}$. Each atomic proposition $a \in \mathcal{AP}$ is typically associated with a *predicate* p_a [2]–[4]. The predicate $p_a(\mathbf{x}; \dots)$ depends on the state $\mathbf{x}$ of the ego vehicle, and possibly also on its environment (denoted by "…" above), which includes, e.g., other traffic participants and traffic signs. To access information about the environment when evaluating predicates, we assume that an environment model is provided. The model includes a behavior prediction of the dynamic obstacles, the kind of which depends on the use case: For determining motion planning constraints, one could use a most likely trajectory, while a set-based prediction is more appropriate for computing fail-safe trajectories. A predicate p_a then induces the set $[\![p_a]\!]_k \subseteq \mathcal{X}_k$ containing the states that satisfy p_a at step $k \in [k_\mathrm{h}]$. We write $\mathbf{x} \models_k a$ for $\mathbf{x} \in [\![p_a]\!]_k$ and extend this notation to propositional formulas ψ over $\mathcal{AP}$. Using the predicates, we can label each state in $\mathbf{x}_{[k_\mathrm{h}]}$ with the propositions it satisfies. This yields the *proposition trace* $\tau(\mathbf{x}_{[k_\mathrm{h}]})$ of $\mathbf{x}_{[k_\mathrm{h}]}$, whose k-th element is $\{a \in \mathcal{AP} \mid \mathbf{x}_k \models_k a\}$. A trajectory is *specification-compliant*, if its proposition trace $\tau(\mathbf{x}_{[k_\mathrm{h}]})$ satisfies φ, i.e., $\tau(\mathbf{x}_{[k_\mathrm{h}]}) \in \mathcal{L}(\varphi)$.

We can now define the exact *specification-compliant reachable set* $\mathcal{R}_k^\mathrm{e}$ at step $k \in [k_\mathrm{h}]$ as the set of all states reachable by following a drivable and specification-compliant trajectory for k steps. Formally, we have

$$\mathcal{R}_k^\mathrm{e} := \{\mathbf{x}_k \mid \mathbf{x}_{[k_\mathrm{h}]} \in \mathcal{X} \wedge \tau(\mathbf{x}_{[k_\mathrm{h}]}) \in \mathcal{L}(\varphi)\}. \quad (2)$$

D. Problem Statement

Even when considering collision avoidance as the only specification, it is impossible to efficiently determine the exact reachable sets [8]. Therefore, we aim to compute a tight overapproximation $\mathcal{R}_k^+ \supseteq \mathcal{R}_k^\mathrm{e}$ for all $k \in [k_\mathrm{h}]$. Because the point-mass model with conservative bounds is reachset conformant [37, Sec. 3.5], our overapproximation encloses all drivable and specification-compliant trajectories of the real vehicle.

We emphasize that computing the specification-compliant reachable set does not replace specification-compliant motion planning. Rather, it is a preparatory step to facilitate planning by restricting the search space. Since some driving situations have a very narrow solution space, we need to ensure that the restricted search space still contains all admissible trajectories, while the planner takes care of guaranteeing that the planned trajectories adhere to the specification. If no specification-compliant trajectory is found in time, the autonomous vehicle can fall back to a previously computed fail-safe trajectory, e.g., as described in [7].

III. Specification-Compliant Reachability Analysis

In the case of a discrete system model, one would construct the product of the specification automaton $\mathcal{A}_\varphi$ and the system model to determine satisfying executions [38, Sec. 5.2]. However, our system model has a continuous state space, so this is not readily feasible. Instead, we construct a *reachability graph* G_R as an overapproximative discretization of the system model as described in Sec. III-A. We store the reachable states of the specification automaton in addition to the reachable system states, to track the progress in satisfying the specification. Finally, we extract the desired overapproximations $\mathcal{R}_k^+$ from G_R as shown in Sec. III-B.

A. Computing the Reachability Graph

Fig. 3 shows an example reachability graph. In line with previous work [8], [9], each graph node is a base set. A *base set* $\mathcal{R}_k^{(i)} \subseteq \mathcal{X}_k$ represents a set of states reachable at time step k, with the index $i \in \mathbb{N}_0$ distinguishing base sets of the same time step. It is given as the Cartesian product of two convex polytopes, one of which represents reachable positions and velocities in s-direction, and the other the same in d-direction. Moreover, we tag each $\mathcal{R}_k^{(i)}$ with a non-empty set of automaton states $\mathcal{Q}_k^{(i)} \subseteq \mathcal{Q}$ to track the states that the specification automaton $\mathcal{A}_\varphi$ can reach. The directed edges in G_R indicate the reachability relation between base sets of successive time steps.

Alg. 1 constructs the reachability graph iteratively for the time steps $k \in [k_\mathrm{h}]$. For ease of presentation, we abbreviate $\{\mathcal{R}_k^{(0)}, \dots, \mathcal{R}_k^{(n)}\}$ as $\mathcal{R}_k$; $\mathcal{Q}_k$, $\hat{\mathcal{R}}_k$, etc. have an analogous meaning. In the k-th iteration, the algorithm computes the successors of $\mathcal{R}_{k-1}$ in G_R. Fig. 4 shows this exemplarily for the reachability graph from Fig. 3. The forward propagation step generates all dynamically reachable states for step k, while the semantic splitting step removes states that violate the specification. Before updating G_R, we merge and repartition the base sets representing the successors to reduce their number. We detail these steps below.

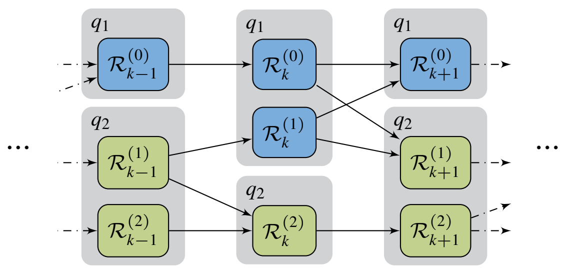

q_1
$\mathcal{R}_{k-1}^{(0)}$
q_1
$\mathcal{R}_{k}^{(0)}$
q_1
$\mathcal{R}_{k+1}^{(0)}$
q_2
$\mathcal{R}_{k-1}^{(1)}$
$\mathcal{R}_{k}^{(1)}$
q_2
$\mathcal{R}_{k+1}^{(1)}$
$\mathcal{R}_{k-1}^{(2)}$
q_2
$\mathcal{R}_{k}^{(2)}$
$\mathcal{R}_{k+1}^{(2)}$
...
...

Fig. 3. Example reachability graph resulting from the automaton in Fig. 2 (shaded areas indicate associated automaton states, e.g., $\mathcal{Q}_k^{(0)} = \{q_1\}$)

Algorithm 1 Computing the Reachability Graph

Input: Planning horizon $k_\mathtt{h}$, initial states $\mathcal{X}_0$, automaton $\mathcal{A}_\varphi = (2^{\mathcal{AP}}, \mathcal{Q}, \delta, q_\iota, \mathcal{F})$, environment model $\mathfrak{E}$
Output: Reachability graph $G_\mathtt{R}$

```
1: for $k = 0$ to $k_\mathtt{h}$ do
2:     if $k = 0$ then                                  ▷ Initialization
3:         $\hat{\mathcal{R}}_k \leftarrow \{\mathcal{X}_0\}$
4:         $\hat{\mathcal{Q}}_k \leftarrow \{\{q_\iota\}\}$
5:     else
6:         $\hat{\mathcal{R}}_k \leftarrow$ PROPAGATE($\mathcal{R}_{k-1}$)
7:         $\hat{\mathcal{Q}}_k \leftarrow \mathcal{Q}_{k-1}$
8:     end if
9:     $\tilde{\mathcal{R}}_k, \tilde{\mathcal{Q}}_k \leftarrow$ SEMANTICSPLIT($\hat{\mathcal{R}}_k$, $\hat{\mathcal{Q}}_k$, $\mathcal{A}_\varphi$, $\mathfrak{E}$)
10:    $\mathcal{R}_k, \mathcal{Q}_k \leftarrow$ REPARTITION($\tilde{\mathcal{R}}_k$, $\tilde{\mathcal{Q}}_k$, $\hat{\mathcal{Q}}_k$)
11:    $G_\mathtt{R}$.UPDATE($\mathcal{R}_k$, $\mathcal{Q}_k$)
12: end for
13: return $G_\mathtt{R}$
```

1) Forward Propagation: We propagate the base sets $\mathcal{R}_{k-1}$ of the preceding time step according to the system dynamics (1), obtaining the base sets $\hat{\mathcal{R}}_k$. Details on the propagation are provided in [8, Sec. IV-A]. Let $\hat{\mathcal{Q}}_k := \mathcal{Q}_{k-1}$ denote the tags of the propagation sources. In the initial step ($k = 0$), there is nothing to propagate. Instead, we set $\hat{\mathcal{R}}_0^{(0)}$ to enclose the initial states $\mathcal{X}_0$; $\hat{\mathcal{Q}}_0^{(0)}$ contains just the initial state q_ι of $\mathcal{A}_\varphi$.

2) Semantic Splitting: We consider every $\hat{\mathcal{R}}_k^{(i)} \in \hat{\mathcal{R}}_k$ individually. We want to remove all system states from $\hat{\mathcal{R}}_k^{(i)}$ whose satisfied atomic propositions cause the specification automaton to get stuck, thereby violating the specification. Thus, we restrict $\hat{\mathcal{R}}_k^{(i)}$ so that we only keep system states that allow $\mathcal{A}_\varphi$ to transition from at least one state in $\hat{\mathcal{Q}}_k^{(i)}$.

Recall from Sec. II-A that $\mathcal{A}_\varphi$ can transition from state q to q' if and only if $\psi_{q,q'}$ is satisfied. Thus, the formula

$$\psi_q := \bigvee\nolimits_{q' \in \hat{\mathcal{Q}}_k^{(i)}} \psi_{q',q}$$

characterizes the system states that enable $\mathcal{A}_\varphi$ to reach the state $q \in \mathcal{Q}$ from $\hat{\mathcal{Q}}_k^{(i)}$. To restrict $\hat{\mathcal{R}}_k^{(i)}$ as described above, we determine a disjunctive normal form (DNF) of each ψ_q. The fixed structure of the DNF facilitates the computation of the restriction: We can implement disjunctions using set unions and conjunctions using set intersections. Let Ψ_q denote the set of conjunctive subformulas of the DNF of ψ_q, which we refer to as *product terms*. For each product term $\pi \in \bigcup_{q \in \mathcal{Q}} \Psi_q$, we restrict $\hat{\mathcal{R}}_k^{(i)}$ to states satisfying π by successively intersecting it with $[\![\lambda]\!]_k$ for each literal λ occurring in π, where

$$[\![\lambda]\!]_k := \begin{cases} [\![p_a]\!]_k & \text{if } \lambda = a \quad \text{with } a \in \mathcal{AP} \\ \mathcal{X}_k \setminus [\![p_a]\!]_k & \text{if } \lambda = \neg a \text{ with } a \in \mathcal{AP} \end{cases}.$$

Depending on the predicate, we can only overapproximate the intersection using the union of multiple base sets. We assume that an implementation of the overapproximative intersection is provided for each predicate; in Sec. IV-A, we give an example for a concrete predicate. Collecting the base sets created for each product term, we obtain the *non-empty* base sets $\tilde{\mathcal{R}}_k^{(j)}$; empty base sets are omitted. If $\tilde{\mathcal{R}}_k^{(j)}$ was created for the product term π, we tag it with $\tilde{\mathcal{Q}}_k^{(j)} := \{q \in \mathcal{Q} \mid \pi \in \Psi_q\}$. Due to non-determinism in $\mathcal{A}_\varphi$, a single product term may occur in multiple Ψ_q. We show that the union of the $\tilde{\mathcal{R}}_k^{(j)}$ subsumes the set of states that allow $\mathcal{A}_\varphi$ to transition out of $\hat{\mathcal{R}}_k^{(i)}$; in particular, we prove:

Proposition 1. *For all $q \in \mathcal{Q}$, the union of all $\tilde{\mathcal{R}}_k^{(j)}$ that are tagged with q cover the states that enable transitioning to q. Formally, this means*

$$\bigcup\nolimits_{j \text{ s.t. } q \in \tilde{\mathcal{Q}}_k^{(j)}} \tilde{\mathcal{R}}_k^{(j)} \supseteq \{\mathbf{x} \in \hat{\mathcal{R}}_k^{(i)} \mid \mathbf{x} \models_k \psi_q\}.$$

Proof. Let $\mathbf{x} \in \hat{\mathcal{R}}_k^{(i)}$ with $\mathbf{x} \models_k \psi_q$ be arbitrary. As $\mathbf{x} \models_k \psi_q$, there exists $\pi \in \Psi_q$ such that $\mathbf{x} \models_k \pi$ by definition of the DNF. Hence, $\mathbf{x} \in [\![\lambda]\!]_k$ for all literals λ in π. We create at least one $\tilde{\mathcal{R}}_k^{(j)}$ for π that contains $\mathbf{x}$, as the intersections of $\hat{\mathcal{R}}_k^{(i)}$ with the $[\![\lambda]\!]_k$ are overapproximative. Finally, $q \in \tilde{\mathcal{Q}}_k^{(j)}$ holds, as $\pi \in \Psi_q$. □

Since the number of new base sets increases with the number of product terms, using a DNF with few product terms is crucial for computational efficiency. As computing the shortest DNF of a formula is NP-hard (cf. [39, Thm. 4]), we settle for an *irredundant* DNF [40], which can be computed efficiently (see, e.g., [41]). An irredundant DNF is minimal in that we cannot delete a product term, nor a literal in any product term. It is necessary to compute the DNF on the fly to deal with non-deterministic specification automata. For deterministic automata, one could precompute all necessary DNFs. Although any non-deterministic automaton can be made deterministic using the powerset construction [42, Def. 11], this may lead to an exponential growth of its state space. When working with a fixed rule set, this trade-off should be investigated.

Let us consider the example in Fig. 4. When processing $\hat{\mathcal{R}}_k^{(0)}$, we find that Ψ_{q_1} contains the product terms $a \wedge b$ and $a \wedge c$, according to the specification automaton from Fig. 2. As indicated by the arrow labels, $\tilde{\mathcal{R}}_k^{(0)}$ is the result of restricting $\hat{\mathcal{R}}_k^{(0)}$ to $a \wedge b$, i.e., it overapproximates $\hat{\mathcal{R}}_k^{(0)} \cap [\![a]\!]_k \cap [\![b]\!]_k$. In this example, no state in $\hat{\mathcal{R}}_k^{(2)}$ satisfies a, which means that

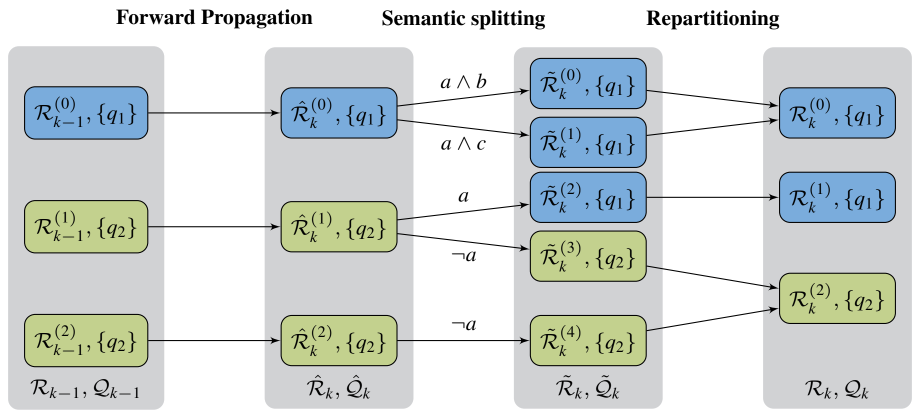

Forward Propagation
Semantic splitting
Repartitioning
$\mathcal{R}_{k-1}^{(0)}, \{q_1\}$
$\mathcal{R}_{k-1}^{(1)}, \{q_2\}$
$\mathcal{R}_{k-1}^{(2)}, \{q_2\}$
$\mathcal{R}_{k-1}, \mathcal{Q}_{k-1}$
$\hat{\mathcal{R}}_k^{(0)}, \{q_1\}$
$\hat{\mathcal{R}}_k^{(1)}, \{q_2\}$
$\hat{\mathcal{R}}_k^{(2)}, \{q_2\}$
$\hat{\mathcal{R}}_k, \hat{\mathcal{Q}}_k$
$a \wedge b$
$a \wedge c$
a
$\neg a$
$\neg a$
$\tilde{\mathcal{R}}_k^{(0)}, \{q_1\}$
$\tilde{\mathcal{R}}_k^{(1)}, \{q_1\}$
$\tilde{\mathcal{R}}_k^{(2)}, \{q_1\}$
$\tilde{\mathcal{R}}_k^{(3)}, \{q_2\}$
$\tilde{\mathcal{R}}_k^{(4)}, \{q_2\}$
$\tilde{\mathcal{R}}_k, \tilde{\mathcal{Q}}_k$
$\mathcal{R}_k^{(0)}, \{q_1\}$
$\mathcal{R}_k^{(1)}, \{q_1\}$
$\mathcal{R}_k^{(2)}, \{q_2\}$
$\mathcal{R}_k, \mathcal{Q}_k$

Fig. 4. Constructing the reachability graph from Fig. 3 using Alg. 1: Base sets and their associated automaton states during the k-th iteration. The arrow labels in the semantic splitting step indicate the product terms used for restricting the propagated base sets.

the discrete transition from q_2 to q_1 is dynamically infeasible at this point. Therefore, the intersection of $\hat{\mathcal{R}}_k^{(2)}$ with $[\![a]\!]_k$ is empty, and the resulting base sets are omitted. Similarly, no state in $\hat{\mathcal{R}}_k^{(0)}$ enables transitioning to q_2.

3) Repartitioning: To reduce the number of base sets to consider in the next iteration, we merge and repartition the base sets in $\tilde{\mathcal{R}}_k$. For this, we apply the procedure from [8, Sec. IV-B], skipping the collision detection step. To ensure that we only merge semantically equivalent base sets, we process them grouped by their discrete source and target states ($\hat{\mathcal{Q}}_k$ and $\tilde{\mathcal{Q}}_k$). In Fig. 4, this allows us to merge $\tilde{\mathcal{R}}_k^{(0)}$ and $\tilde{\mathcal{R}}_k^{(1)}$ even though the contained states satisfy different propositions. We store the repartitioned base sets $\mathcal{R}_k$ as nodes in G_{R} and create an edge from $\mathcal{R}_{k-1}^{(i)}$ to $\mathcal{R}_k^{(l)} \in \mathcal{R}_k$, if $\mathcal{R}_k^{(l)}$ is reachable from $\mathcal{R}_{k-1}^{(i)}$. For each $\mathcal{R}_k^{(l)}$, we set $\mathcal{Q}_k^{(l)}$ to the discrete target states of the corresponding group.

B. Extracting the Overapproximations from the Graph

We want to assemble our desired overapproximations $\mathcal{R}_k^+$ of the specification-compliant reachable set by unifying base sets from the reachability graph. For the following theorem, we define an *accepted path* as a path $\mathcal{R}_0^{(i_0)}, \dots, \mathcal{R}_{k_{\mathrm{h}}}^{(i_{k_{\mathrm{h}}})}$ in G_{R}, where $\mathcal{Q}_{k_{\mathrm{h}}}^{(i_{k_{\mathrm{h}}})} \cap \mathcal{F} \neq \emptyset$, i.e., $\mathcal{R}_{k_{\mathrm{h}}}^{(i_{k_{\mathrm{h}}})}$ is tagged with at least one final state.

Theorem 1. *Suppose $\mathbf{x}_{[k_{\mathrm{h}}]}$ is a drivable and specification-compliant trajectory. Then, there exists an accepted path $\mathcal{R}_0^{(i_0)}, \dots, \mathcal{R}_{k_{\mathrm{h}}}^{(i_{k_{\mathrm{h}}})}$ in G_{R} with $\mathbf{x}_k \in \mathcal{R}_k^{(i_k)}$ for all $k \in [k_{\mathrm{h}}]$.*

Sketch of proof. As $\mathbf{x}_{[k_{\mathrm{h}}]}$ is specification-compliant, there exists an accepting run $\langle q_\iota, q_0, \dots, q_{k_{\mathrm{h}}}\rangle$ of $\mathcal{A}_\varphi$ for $\tau(\mathbf{x}_{[k_{\mathrm{h}}]})$, where $q_{k_{\mathrm{h}}} \in \mathcal{F}$. We proceed by induction on the time step k and rely on the fact that all operations in our algorithm are overapproximative. Suppose we have already shown $\mathbf{x}_k \in \mathcal{R}_k^{(i_k)}$ and $q_k \in \mathcal{Q}_k^{(i_k)}$. As the trajectory is drivable, the states generated by propagating $\mathcal{R}_k^{(i_k)}$ must include $\mathbf{x}_{k+1}$. Moreover, the atomic propositions satisfied by $\mathbf{x}_{k+1}$ allow $\mathcal{A}_\varphi$ to transition from q_k to q_{k+1}. Thus, $\mathbf{x}_{k+1}$ is included in a base set tagged with q_{k+1} after the semantic splitting step (cf. Prop. 1). As the repartitioning is overapproximative and preserves the tags of the base sets, it yields the desired $\mathcal{R}_{k+1}^{(i_{k+1})}$ that contains $\mathbf{x}_{k+1}$ and is tagged with q_{k+1}. We can reason similarly for the base case of the induction. $\square$

We choose our overapproximations $\mathcal{R}_k^+$ as the union of all base sets $\mathcal{R}_k^{(i)}$ lying on any accepted path. Using Thm. 1, we show that these subsume the exact specification-compliant reachable sets $\mathcal{R}_k^{\mathrm{e}}$ as required.

Corollary 1. *For all $k \in [k_{\mathrm{h}}]$, we have $\mathcal{R}_k^+ \supseteq \mathcal{R}_k^{\mathrm{e}}$.*

Proof. We show that $\mathcal{R}_k^+ \supseteq \mathcal{R}_k^{\mathrm{e}}$ by proving that every state in the exact specification-compliant reachable set is included in our overapproximation. Thus, let $\mathbf{x}_k \in \mathcal{R}_k^{\mathrm{e}}$ be arbitrary. By definition of the exact specification-compliant reachable set (2), $\mathbf{x}_k$ must be part of a drivable and specification-compliant trajectory. Therefore, Thm. 1 yields an accepted path $\mathcal{R}_0^{(i_0)}, \dots, \mathcal{R}_{k_{\mathrm{h}}}^{(i_{k_{\mathrm{h}}})}$ with $\mathbf{x}_k \in \mathcal{R}_k^{(i_k)}$. As $\mathcal{R}_k^{(i_k)}$ lies on an accepted path, we have $\mathcal{R}_k^{(i_k)} \subseteq \mathcal{R}_k^+$ and thus $\mathbf{x}_k \in \mathcal{R}_k^+$. $\square$

Inspired by [8, Sec. V], we prune the reachability graph to remove all base sets not contributing to the overapproximations. To this end, we first delete all $\mathcal{R}_{k_{\mathrm{h}}}^{(i)}$ where $\mathcal{Q}_{k_{\mathrm{h}}}^{(i)} \cap \mathcal{F} = \emptyset$. Then, we iteratively eliminate the $\mathcal{R}_k^{(i)}$ with $k \in [k_{\mathrm{h}} - 1]$ that no longer have successors in the graph.

IV. Evaluation

We evaluate our approach using intersection and interstate traffic rules from [2] and [3]. Since these are given in MTL with past connectives, we rewrite them to LTL$_{\mathrm{f}}$ as described in [9, Sec. VI]. We compare our approach with that from [9], since, to the best of our knowledge, it is the only other approach for computing the reachable set of an autonomous vehicle adhering to an LTL$_{\mathrm{f}}$ specification. To this end, we use simple hand-crafted scenarios. Then, we benchmark our prototype on scenarios from the exiD dataset [43] to evaluate its performance in real-world situations.

A. Implementation Details

Our prototype builds on CommonRoad-Reach [44] and is implemented in Python and C++. It expects the input scenarios to be in the CommonRoad[2] [45] format. In our prototype, the environment model provides a most likely trajectory as behavior prediction for each dynamic obstacle. We performed our experiments on a laptop with an Intel Core i7-12700H 4.7 GHz processor. Considering the physical limits of a typical vehicle, we choose the following bounds for the velocities and accelerations of the vehicle model:

$$-13.9\,\mathrm{m/s} \le \dot{s} \le 50.8\,\mathrm{m/s} \qquad -4.0\,\mathrm{m/s} \le \dot{d} \le 4.0\,\mathrm{m/s}$$
$$-11.5\,\mathrm{m/s^2} \le \ddot{s} \le 11.5\,\mathrm{m/s^2} \qquad -2.0\,\mathrm{m/s^2} \le \ddot{d} \le 2.0\,\mathrm{m/s^2}$$

All computation times are averaged over five executions.

For our experiments, we set collision avoidance as a default specification. The corresponding LTL$_\mathrm{f}$ formula is $\mathbf{G}\,\texttt{c\_free}$, where the predicate $\texttt{c\_free}(\mathbf{x}_k; \mathfrak{E})$ is true, if and only if the occupied region of the ego vehicle in state $\mathbf{x}_k$ is disjoint from all obstacles in the environment model $\mathfrak{E}$. To overapproximate the intersection of a base set with $[\![\texttt{c\_free}]\!]_k$, we underapproximate the shape of the ego vehicle with its inscribed circle. Then, we can filter out the violating states by recursively splitting the base sets that collide with an obstacle as described in [8, Sec. IV-B]. The intersection with $[\![\neg\texttt{c\_free}]\!]_k$ can be implemented similarly.

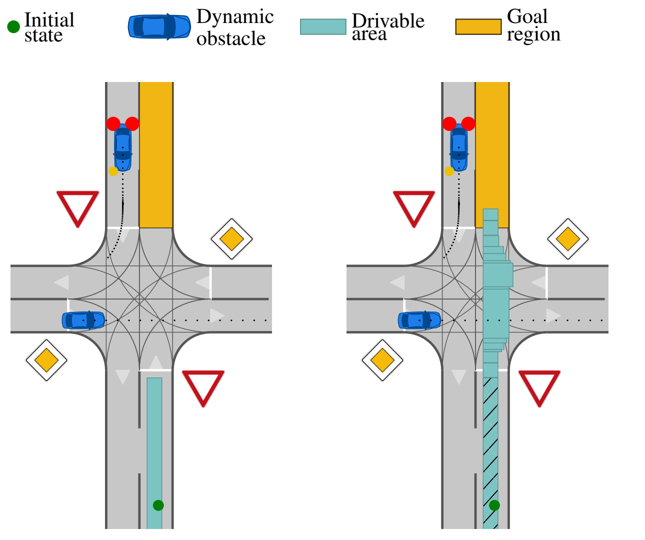
B. Comparison on Simple Scenarios

Fig. 5 shows an intersection in which the horizontal road is a priority road (scenario ID: `ZAM_Yield-1_1_T-1`). According to the priority rule R-IN4 [2, Tab. VI], the ego vehicle, which is approaching from the bottom, must not endanger the left vehicle crossing the intersection on the priority road. Consequently, the ego vehicle should stop before entering the intersection and give way to the crossing vehicle. We set $k_\mathrm{h} := 15$ and $\Delta t := 0.2\,\mathrm{s}$. Using rather coarse time steps is sufficient here, as our goal is to narrow the search space of a motion planner, which then checks the specification compliance of the planned trajectories.

Figs. 5a and 5b show the drivable area of the ego vehicle at time step 9 computed with our on-the-fly approach and the offline model checking approach in [9], respectively. The *drivable area* at step k is defined as the projection of the base sets $\mathcal{R}_k$ to the position domain. As the offline approach disregards the specification during the reachability analysis, the resulting reachability graph indicates that the ego vehicle could enter the intersection. In the subsequent model checking step, the algorithm determines that stopping before the intersection is the only rule-compliant maneuver. At step 9, the model-checked reachability graph thus contains only the base set corresponding to the drivable area hatched in Fig. 5b. With our approach, this is the only base set generated for step 9, even before pruning the reachability graph. However, pruning is generally required to achieve results comparable to the offline approach.

Tab. I compares the number of base sets in the reachability graph (before pruning) and the computation time for the reachability analysis (including pruning). In addition to the scenario introduced above, the table considers the

[2]https://commonroad.in.tum.de



(a) Our approach (b) Approach in [9]

Fig. 5. Drivable area for the give-way scenario at time step 9 (hatching in (b) indicates the drivable area that remains after model checking).

TABLE I
COMPARISON WITH THE OFFLINE APPROACH IN [9]

ID (`ZAM_` ...)	#Base Sets		Comp. time	
	Ours	Theirs	Ours	Theirs
`Yield-1_1_T-1`	23	163	12 ms	185 ms
`TIV-1_1_T-1`	29	187	11 ms	37 ms
`TIV-2_1_T-1`	58	181	22 ms	204 ms

interstate and intersection scenarios used in [9, Sec. VIII] (IDs: `ZAM_TIV-1_1_T-1` and `ZAM_TIV-2_1_T-1`). Our approach creates considerably fewer base sets in all scenarios because we can discard many non-compliant system states early. Besides, our repartitioning strategy is more liberal since, contrary to the offline approach, we do not require all states in a base set to satisfy the same atomic propositions. As a result, our approach is faster by a factor of up to 15. Computing the specification automaton took about 1 ms for the interstate scenario and 10 ms for the intersection scenarios. The separate model checking step in the offline approach required 8 ms on average. However, this step runs in Python, while the other computations run mainly in C++.

C. Benchmarking in Real-World Scenarios

To assess the performance of our approach in real-world scenarios, we extracted CommonRoad scenarios from the exiD dataset [43] (location: Merzenich-Rather). The scenarios feature an interstate with a three-lane main carriageway and one on-ramp, as well as several dynamic obstacles. Since we want to consider the “respect entering vehicles” rule R_I5 [3, Tab. I] for our benchmark, we selected 125 scenarios in which vehicles enter the main carriageway via the on-ramp. Thus, we include only scenarios in which R_I5 cannot be deemed irrelevant a priori due to the absence of entering vehicles. Fig. 6 depicts an example scenario

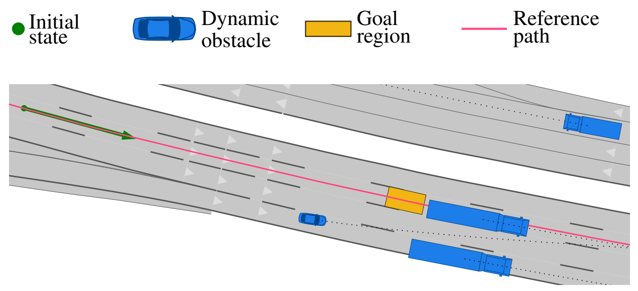


Fig. 6. A scenario from the exiD dataset with one car and two trucks going in the same direction as the ego vehicle. Due to the car entering the interstate, the ego vehicle must not change to the right lane.

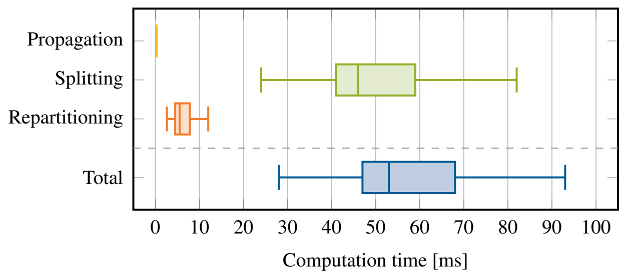


Fig. 7. Benchmark results on selected scenarios of the exiD dataset (outliers are not shown; the maximum total computation time is 332 ms).

at the initial time step, including the reference path for the curvilinear coordinate system and the trajectories of the dynamic obstacles. As backward driving is prohibited on interstates, we additionally impose the rule $\mathbf{G}\,\neg\texttt{reverses}$ (see [3, Sec. IV-B] for the predicate definition).

Fig. 7 shows the computation time of our algorithm broken down by the steps described in Sec. III-A. With $k_{\mathrm{h}} := 30$ and $\Delta t := 0.08\,\mathrm{s}$, the algorithm terminated within 70 ms for 75 % of the scenarios. This is well below the real-time planning horizon of $k_{\mathrm{h}} \cdot \Delta t = 2.4\,\mathrm{s}$, suggesting that our approach is real-time capable. Eleven scenarios exceeded a computation time of 100 ms, with the maximum observed time at 332 ms for a scenario with three entering vehicles. Compared to the total execution time, the time required for creating the specification automaton and pruning the reachability graph is negligible (on average 1 ms and 0.03 ms, respectively).

V. Conclusion

We proposed integrating LTL$_{\mathrm{f}}$ model checking into set-based reachability analysis to overapproximate the specification-compliant reachable set of an autonomous vehicle. By performing model checking on the fly, we can exclude many dynamically reachable but non-compliant system states early on during the reachability analysis. We demonstrated that our algorithm is considerably faster than the offline approach in [9] for both intersection and interstate scenarios. Our evaluation using recorded real-world scenarios suggests that the approach is real-time capable.

In the future, the order in which the predicates are applied during semantic splitting could be optimized. For example, it might be helpful to apply the most restrictive predicates first in order to quickly decide whether the formula can be satisfied at all. Moreover, more predicates need to be implemented in our prototype to support additional traffic rules. We will integrate our implementation into an upcoming release of CommonRoad.

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] 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.

[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] 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.

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

[6] 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.

[7] C. Pek, S. Manzinger, M. Koschi, and M. Althoff, "Using online verification to prevent autonomous vehicles from causing accidents," *Nature Machine Intelligence*, vol. 2, no. 9, pp. 518–528, 2020.

[8] S. Söntges and M. Althoff, "Computing the drivable area of autonomous road vehicles in dynamic road scenes," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 6, pp. 1855–1866, 2018.

[9] 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.

[10] 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.

[11] 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.

[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 Conf. (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] 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.

[16] 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.

[17] 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.

[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] 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.

[24] 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.

[25] 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.

[26] 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 Conf. (ECC)*, 2016, pp. 690–695.

[27] 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.

[28] 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.

[29] 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.

[30] A. Alanwar, F. J. Jiang, M. Sharifi, D. V. Dimarogonas, and K. H. Johansson, "Enhancing data-driven reachability analysis using temporal logic side information," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2022, pp. 6793–6799.

[31] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)*, 2013, pp. 854–860.

[32] S. Dutta and M. Y. Vardi, "Assertion-based flow monitoring of SystemC models," in *Proc. of the ACM/IEEE Conf. on Formal Methods and Models for Codesign (MEMOCODE)*, 2014, pp. 145–154.

[33] A. Duret-Lutz, E. Renault, M. Colange, F. Renkin, A. Gbaguidi Aisse *et al.*, "From Spot 2.0 to Spot 2.10: What's new?" in *Proc. of the Int. Conf. on Computer Aided Verification (CAV)*, S. Shoham and Y. Vizel, Eds., 2022, pp. 174–187.

[34] O. Kupferman and M. Y. Vardi, "Model checking of safety properties," *Formal Methods in System Design*, vol. 19, no. 3, pp. 291–314, 2001.

[35] 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.

[36] J. Eilbrecht and O. Stursberg, "Challenges of trajectory planning with integrator models on curved roads," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15 588–15 595, 2020.

[37] H. Roehm, J. Oehlerking, M. Woehrle, and M. Althoff, "Model conformance for cyber-physical systems: A survey," *ACM Transactions on Cyber-Physical Systems*, vol. 3, no. 3, pp. 30:1–30:26, 2019.

[38] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[39] C. Umans, "The minimum equivalent DNF problem and shortest implicants," *Journal of Computer and System Sciences*, vol. 63, no. 4, pp. 597–611, 2001.

[40] W. V. Quine, "The problem of simplifying truth functions," *The American Mathematical Monthly*, vol. 59, no. 8, pp. 521–531, 1952.

[41] S.-i. Minato, "Fast generation of prime-irredundant covers from binary decision diagrams," *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, vol. E76-A, no. 6, pp. 967–973, 1993.

[42] M. O. Rabin and D. Scott, "Finite automata and their decision problems," *IBM Journal of Research and Development*, vol. 3, no. 2, pp. 114–125, 1959.

[43] 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.

[44] E. Irani Liu, G. Würsching, M. Klischat, and M. Althoff, "CommonRoad-Reach: A toolbox for reachability analysis of automated vehicles," in *Proc. of the IEEE Int. Intelligent Transportation Systems Conf. (ITSC)*, 2022, pp. 2313–2320.

[45] 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.