Computationally Efficient Safety Falsification of Adaptive Cruise Control Systems

Markus Koschi*, Christian Pek*, Sebastian Maierhofer*, and Matthias Althoff

***Abstract*— Falsification aims to disprove the safety of systems by providing counter-examples that lead to a violation of safety properties. In this work, we present two novel falsification methods to reveal safety flaws in adaptive cruise control (ACC) systems of automated vehicles. Our methods use rapidly-exploring random trees to generate motions for a leading vehicle such that the ACC under test causes a rear-end collision. By considering unsafe states and searching backward in time, we are able to drastically improve computation times and falsify even sophisticated ACC systems. The obtained collision scenarios reveal safety flaws of the ACC under test and can be directly used to improve the system's design. We demonstrate the benefits of our methods by successfully falsifying the safety of state-of-the-art ACC systems and comparing the results to that of existing approaches.**

I. Introduction

Safety is a mandatory requirement for the ever increasing automation of vehicles. However, ensuring safety is a challenging task; even in supposedly simple scenarios like vehicle following (cf. adaptive cruise control (ACC) system in Fig. 1), the variety of stop-and-go behaviors of other vehicles may impose safety-critical situations.

Verifying that motion planning algorithms ensure certain standards is often done by testing the system in a multitude of simulations using large databases of test cases and traffic scenarios [1]–[3]. However, simulations have the significant disadvantage that they may miss testing certain scenarios that inevitably lead to unsafe situations. In contrast, formal verification approaches are able to provide strong safety guarantees by verifying that each action of the vehicle conforms to a formal specification [4]. Nevertheless, safety only holds if the used specification appropriately models the desired safety definition and no implementation mistakes have been made.

To reveal safety-critical flaws in a system, falsification approaches try to disprove the safety instead of proving it [5]. Falsification for motion planning aims to find motions that start in a safe state but eventually enter collision states (cf. Fig. 2). The obtained motions serve as counter-examples and can be used to revise the system's design. Falsification should be an integral part in the development of automated vehicles; however, falsification approaches for automated vehicle functions still have a huge potential in terms of computational efficiency and applicability [6].

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

All authors are with the Department of Informatics, Technical University of Munich, 85748 Garching, Germany.
`markus.koschi@tum.de, christian.pek@tum.de, sebastian.maierhofer@tum.de, althoff@tum.de`

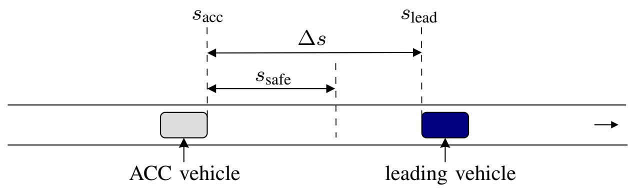


Fig. 1: To follow a leading vehicle, ACC systems adjust the velocity of the ACC-equipped vehicle so that the headway $\Delta s = s_{\text{lead}} - s_{\text{acc}}$ is larger than a safe distance s_{safe}.

A. Related work

In the following paragraphs, we *a)* provide a brief overview of safety mechanisms in state-of-the-art ACC systems and *b)* review existing techniques for generating safety-critical scenarios for automated vehicles.

a) Adaptive cruise control systems: ACC systems automatically adjust the velocity of the controlled vehicle to maintain a certain headway to a leading vehicle [7], [8]. Reviews about major ACC developments can be found in [9], [10]. Proportional integral ACCs (PI-ACCs) use PI controllers to adjust the headway and are still widely used because of their simplicity [11], [12]. Their safety relies mainly on the chosen gains to react to sudden changes in the behavior of the leading vehicle. The Intelligent Driver Model ACC (IDM-ACC) implements a more complex control scheme by switching between different driving modes [13]. As a result, the IDM-ACC can switch between comfortable or rather safe parameterizations. Nevertheless, these ACC systems do not incorporate dedicated collision avoidance mechanisms.

Collision avoidance ACCs (CA-ACCs) explicitly consider collision avoidance by quickly adjusting their response behavior if the desired headway cannot be maintained [14]. Recently proposed ACCs make use of formal methods (FM-ACC), e.g., set invariance theory or formalized traffic rules, to provide safety guarantees during operation [15]–[18].

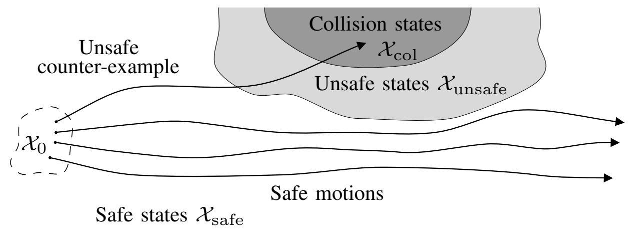

Collision states $\mathcal{X}_{\text{col}}$
Unsafe counter-example
Unsafe states $\mathcal{X}_{\text{unsafe}}$
$\mathcal{X}_0$
Safe motions
Safe states $\mathcal{X}_{\text{safe}}$

Fig. 2: Falsification of motion planning aims to find counter-examples that transition the system under test from an initial safe state $x_0 \in \mathcal{X}_0 \subset \mathcal{X}_{\text{safe}}$ to a collision state $x \in \mathcal{X}_{\text{col}}$.

b) Generating safety-critical scenarios: To test ACC systems, some approaches synthesize safety-critical scenarios [5], [19]–[22], e.g., by making use of Monte Carlo simulation (MCS) to automatically generate a variety of random scenarios. Although MCS approaches quickly generate various scenarios, they are not designed to specifically find scenarios leading to collisions.

The authors of [23] propose a systematic approach to test collision avoidance systems by primarily simulating scenarios in which leading vehicles suddenly perform emergency braking maneuvers. More sophisticated methods make use of reachability analysis, neural networks, performance metrics, or evolutionary algorithms to automatically generate safety-critical scenarios for fully automated vehicles [24]–[29]. Rapidly-exploring random trees (RRTs) [30] are used in [31]–[33] to falsify the safety of a given system, since RRTs are well suited to efficiently explore large search spaces.

B. Contributions

This paper proposes two approaches based on RRTs to falsify the safety of ACC systems. Since the aforementioned falsification methods are often computationally expensive and do not exploit domain knowledge to efficiently generate counter-examples, our contributions tackle these issues by:

1) drastically improving the performance of forward searches by integrating unsafe states, which eventually lead to collisions and are much easier to reach than collision states (cf. Fig. 2);
2) presenting a novel falsification approach that employs a backward search scheme and can successfully falsify sophisticated ACC systems in significantly less time than that of forward searches; and
3) demonstrating that our approaches successfully falsify state-of-the-art ACC systems from the literature in reasonable time and outperform classical forward search and MCS.

The remainder of this paper is organized as follows. Sec. II introduces necessary mathematical definitions and the problem statement. Next, our falsification algorithms to generate safety-critical situations are presented in Sec. III. In Sec. IV, the proposed approaches are used to falsify the safety of state-of-the-art ACC systems in numerical experiments, and the results are compared to that of Monte Carlo falsification. Conclusions are presented in Sec. V.

II. Definitions and Problem Statement

A. Vehicle configuration

Let us introduce $\mathcal{X} \subset \mathbb{R}^2$ as the set of feasible states x of a vehicle. The state vector $x = [s, v]^T$ consists of the position s and the velocity v, each in the longitudinal direction. Acceleration and jerk are denoted by a and j, respectively. We assume discrete-time systems with a time step size of $\Delta t > 0$. We further introduce $\mathcal{U} \subset \mathbb{R}$ as the set of admissible control inputs $u = a$ of the state transition function f_{motion}, which describes the longitudinal dynamics of a vehicle:

$$\underbrace{\begin{bmatrix} s(t_{k+1}) \\ v(t_{k+1}) \end{bmatrix}}_{x(t_{k+1})} = \underbrace{\begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} s(t_k) \\ v(t_k) \end{bmatrix} + \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} u}_{f_{\text{motion}}(x(t_k), u)}, \qquad (1)$$

with bounded velocity, acceleration, and jerk: $0 \leq v \leq v_{\max}$, $a_{\min} \leq u \leq a_{\max}$, $j_{\min} \leq j \leq j_{\max}$, where $a_{\min}, j_{\min} \in \mathbb{R}_{<0}$ and $j(t_k) = {}^{(u(t_k) - u(t_{k-1}))}\!/_{\Delta t}$. We adhere to the notation $x\big([t_0, t_n]\big)$ to describe a trajectory of states $x(t_i) \in \mathcal{X}$ for $t_i \in \{t_0, t_1, \ldots, t_n\}$ that satisfy (1) and its constraints, and we use $u\big([t_0, t_n]\big)$ analogously to describe an input trajectory.

As shown in Fig. 1, we consider situations in which an ACC-equipped vehicle is following another vehicle. The variables of the leading and ACC vehicle are denoted by the subscript $\square_{\text{lead}}$ and $\square_{\text{acc}}$, respectively. By defining the reference point for the position of the leading vehicle at its rear end and of the ACC vehicle at its front, the relative distance between both vehicles is $\Delta s := s_{\text{lead}} - s_{\text{acc}}$ (cf. Fig. 1). Their relative velocity is defined as $\Delta v := v_{\text{lead}} - v_{\text{acc}}$.

We treat the ACC control law under test as a black box:

$$u_{\text{acc}} = f_{\text{ACC}}\big(x_{\text{acc}}(t_k), x_{\text{lead}}(t_k), \delta\big), \qquad (2)$$

where f_{ACC} is unknown and δ is the reaction delay of the system, e.g., processing time of sensors and actuator delays. For brevity, we combine f_{ACC} and f_{motion} in the function $f_{\text{ACC-motion}}\big(x_{\text{acc}}(t_k), x_{\text{lead}}(t_k), \delta\big)$, which returns $x_{\text{acc}}(t_{k+1})$.

B. Safety definition

To define safe states, we use the established safety definition that the ACC vehicle must remain collision-free at all times [34]. This must hold even if the leading vehicle suddenly performs emergency braking, i.e., $u^{\text{brake}}_{\text{lead}}(t_i) := \max\big(a_{\text{lead}}(t_{i-1}) + j_{\text{min,lead}}\Delta t, a_{\text{min,lead}}\big)$. In response, an ACC vehicle that conforms with [34] will fully brake; during its reaction delay, we allow arbitrary acceleration, which can be over-approximated by full acceleration. Thus, for our safety analysis, we assume that the ACC vehicle applies the control inputs

$$u^{\text{brake}}_{\text{acc}}(t_i) := \begin{cases} \min\big(a_{\text{acc}}(t_{i-1}) + j_{\text{max,acc}}\Delta t, \\ a_{\text{max,acc}}\big), & t_i - t_k < \delta, \\ \max\big(a_{\text{acc}}(t_{i-1}) + j_{\text{min,acc}}\Delta t, \\ a_{\text{min,acc}}\big), & \text{otherwise}, \end{cases}$$

where t_k is the point in time at which the leading vehicle starts emergency braking. Let $t^{\text{stop}}_{\text{acc}}$ denote the point in time at which the ACC vehicle is at a standstill.

Definition 1 (Safe distance)
The ACC vehicle can definitely avoid a rear-end collision, if it maintains at least the minimal safe distance s_{safe} to the leading vehicle:

$$s_{safe}(t_k) := \inf\Big(\big\{\Delta s(t_k) \,\big|\, \forall t_i \in \{t_k, t_{k+1}, \ldots, t^{stop}_{acc}\} : \Delta s(t_i) > 0\big\}\Big),$$

where $\Delta s(t_i)$ is obtained by simulating the leading and ACC vehicle according to (1) *from $v(t_k)$ with $u^{brake}_{lead}\big([t_k, t^{stop}_{acc}]\big)$ and $u^{brake}_{acc}\big([t_k, t^{stop}_{acc}]\big)$, respectively.*

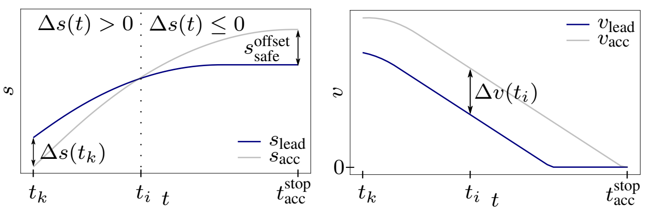
Our definition is based on [34] and [35] with the following extensions: in contrast to [34], we allow for a reaction delay of the ACC vehicle; in contrast to [35], we do not assume that the ACC vehicle maintains constant velocity during the reaction delay, but we allow for arbitrary acceleration; and in contrast to both, we consider limited jerk to allow for more realistic braking profiles.

Definition 2 (Unsafe distance)
The ACC vehicle definitely cannot avoid a rear-end collision with impact velocity of at least v_{col} (which is a parameter ≥ 0), if the leading vehicle performs emergency braking at t_k and if the ACC vehicle maintains less than or equal the maximum unsafe distance s_{unsafe} to the leading vehicle:

$$s_{unsafe}(t_k) := \sup \big(\{\Delta s(t_k) \,\big|\, \exists t_i \in \{t_{k+1}, t_{k+2}, \ldots, t_{acc}^{stop}\} : \\ \Delta s(t_{i-1}) > 0 \wedge \Delta s(t_i) \leq 0 \wedge |\Delta v(t_i)| \geq v_{col}\}\big),$$

where $\Delta s(t_{i-1})$, $\Delta s(t_i)$, and $\Delta v(t_i)$ are obtained by simulation as in Def. 1 except that we set $\delta = 0$ so that the acceleration during the reaction delay is under-approximated.

Algorithmically, Def. 1 and 2 can be evaluated by simulating both vehicles from their given current states at t_k with $u_{\text{lead}}^{\text{brake}}\big([t_k, t_{\text{acc}}^{\text{stop}}]\big)$ and $u_{\text{acc}}^{\text{brake}}\big([t_k, t_{\text{acc}}^{\text{stop}}]\big)$. The safe distance is obtained by adding $\Delta s(t_k)$ to the minimal required offset of the relative position so that both position profiles do not intersect (cf. $s_{\text{safe}}^{\text{offset}}$ in Fig. 3a). The unsafe distance is obtained by adding $\Delta s(t_k)$ to the maximal possible offset of the relative position so that both position profiles intersect and that the absolute value of the relative velocity at this point in time is at least v_{col} (cf. $\Delta v(t_i)$ in Fig. 3b).



(a) Simulated position profiles. (b) Simulated velocity profiles.

Fig. 3: State plots of the leading and ACC vehicle to determine the safe and unsafe distance.

Definition 3 (Sets of safe, unsafe, and collision states)
Using Def. 1 and 2, we define the set of all safe states of the ACC vehicle at time t_k as

$$\mathcal{X}_{safe}(t_k) := \{x_{acc}(t_k) \in \mathcal{X} \,|\, \Delta s(t_k) \geq s_{safe}(t_k)\},$$

the set of unsafe states as

$$\mathcal{X}_{unsafe}(t_k) := \{x_{acc}(t_k) \in \mathcal{X} \,|\, \Delta s(t_k) \leq s_{unsafe}(t_k)\},$$

and the set of collision states as

$$\mathcal{X}_{col}(t_k) := \{x_{acc}(t_k) \in \mathcal{X} \,|\, \Delta s(t_k) \leq 0 \wedge |\Delta v(t_k)| \geq v_{col}\}.$$

Note that $\mathcal{X} = \mathcal{X}_{\text{safe}}(t_k) \cup \mathcal{X}_{\text{unsafe}}(t_k)$ only holds if $\delta = 0$ and $v_{\text{col}} = 0$.

We use Def. 3 in our approach to detect if we have already generated an unsafe situation for the ACC system. Therefore, we model the search space of our RRT by combining the state spaces of both vehicles. Thus, a node $z(t_k)$ of the search tree $\mathcal{T}$ is defined as state tuple: $z(t_k) := \big(x_{\text{acc}}(t_k), x_{\text{lead}}(t_k)\big)$. We denote a node as safe if $x_{\text{acc}}(t_k) \in \mathcal{X}_{\text{safe}}(t_k)$, as unsafe if $x_{\text{acc}}(t_k) \in \mathcal{X}_{\text{unsafe}}(t_k)$, and as colliding if $x_{\text{acc}}(t_k) \in \mathcal{X}_{\text{col}}(t_k)$.

C. *Problem statement*

In order to falsify an ACC system (cf. Fig. 2), we aim to find a time series of inputs for the leading vehicle $u_{\text{lead}}(t_i)$, $t_i \in \{t_0, t_1, \ldots, t_{\text{col}}\}$, so that when starting in a safe state $x_{\text{acc}}(t_0) \in \mathcal{X}_{\text{safe}}(t_0)$, the ACC vehicle will eventually collide with the leading vehicle at $t_{\text{col}} > t_0 : x_{\text{acc}}(t_{\text{col}}) \in \mathcal{X}_{\text{col}}(t_{\text{col}})$.

III. Falsification Approaches

To quickly find counter-examples bridging the sets $\mathcal{X}_{\text{safe}}(t_k)$ and $\mathcal{X}_{\text{unsafe}}(t_k)$, we efficiently explore the search space using RRTs (see Sec. III-A). We have developed two methods to build the RRT: 1) We search forward in time by creating random behaviors of the leading vehicle and by evaluating the reactions of the ACC vehicle (see Sec. III-B). However, this strategy may require many simulation runs if the set of unsafe states is very small, which is the case for many ACC systems (e.g., an ACC with advanced collision avoidance). 2) Thus, our second approach starts from unsafe states and searches backward in time (see Sec. III-C). As a result, we can find counter-examples in fewer simulation runs.

A. *Rapidly-exploring random trees*

RRTs are a popular approach for motion planning and have already been used for falsification (cf. Sec. I-A). The standard approach (e.g., [30], [36]) starts at an initial set of nodes and generates new nodes from time step t_k to t_j as follows. After drawing a random sample in the search space, the node $z_{\text{near}}(t_k)$ that is closest to the sample according to a distance measure is selected as its parent. Then, the optimal input u is applied to drive the system from the parent node as close as possible to the sample, resulting in the new node $z_{\text{new}}(t_j)$. This procedure can be repeated such that the same number of nodes, denoted by the parameter z_{num}, is generated for each time step. We have made modifications to this standard approach, which are introduced next and are combined with the standard approach in Alg. 1.

Our sampling process is performed in relative coordinates $\Delta z := [\Delta s, \Delta v]^T$, since only these are relevant for the criticality. To avoid generating behavior that mostly uses the minimal or maximal possible input, we do not sample in the complete search space, but restrict the sampling range depending on the states of already existing nodes. Therefore, we first compute the minimum and maximum differences, $\Delta z_{\min}(t_k)$ and $\Delta z_{\max}(t_k)$, between the states of all nodes at the current time step t_k. To favor an increase of the relative coordinates, we add a bias to obtain the sampling range $\mathcal{Z}(t_k) := [\Delta z_{\min}(t_k) - \Delta z_{\text{add}}^{\min}, \Delta z_{\max}(t_k) + \Delta z_{\text{add}}^{\max}]$ (cf. line 1 of Alg. 1).

Algorithm 1 EXPLORE($\mathcal{T}$, t_k, t_j, $\mathcal{X}_{\text{acc}}(t_j)$)

```
Input: search tree $\mathcal{T}$, current time $t_k$, desired time $t_j$, set of states $\mathcal{X}_{\text{acc}}(t_j)$
Output: generated node $z(t_j)$
1: $\mathcal{Z}(t_k) \leftarrow \mathcal{T}$.GETSAMPLINGRANGE($t_k$)
2: $\Delta z(t_j) \leftarrow$ DRAWSAMPLE($\mathcal{Z}(t_k)$)
3: $z_{\text{near}}(t_k) \leftarrow \mathcal{T}$.GETNEARESTNODE($\Delta z(t_j)$)
4: $x_{\text{acc}}(t_j) \leftarrow$ GETSUCCESSORSTATE($z_{\text{near}}(t_k)$, $\mathcal{X}_{\text{acc}}(t_j)$)
5: $u_{\text{lead}} \leftarrow$ CALCINPUT($z_{\text{near}}(t_k)$, $\Delta z(t_j) + x_{\text{acc}}(t_j)$)
6: $x_{\text{lead}}(t_j) \leftarrow f_{\text{MOTION}}(z_{\text{near}}(t_k), u_{\text{lead}})$
7: return $z_{\text{new}}(t_j) \leftarrow \big(x_{\text{acc}}(t_j), x_{\text{lead}}(t_j)\big)$
```

As a distance measure for the selection of the nearest node $z_{\text{near}}(t_k)$ (cf. line 3), we use the L^2 norm with normalized state values. The normalization is done using the mean and standard deviation of all nodes $z(t_k)$ to avoid the preference of low numerical values in the position and velocity when selecting the closest node.

As an additional input, our algorithm requires the set of states of the ACC vehicle at the next time step, denoted by $\mathcal{X}_{\text{acc}}(t_j)$. From $\mathcal{X}_{\text{acc}}(t_j)$, we select the state $x_{\text{acc}}(t_j)$ that is the successor of the state of the ACC vehicle in $z_{\text{near}}(t_k)$ (cf. line 4). This allows us to compute the input u_{lead} that drives the leading vehicle as close as possible to the sampled configuration in global coordinates $\Delta z(t_j) + x_{\text{acc}}(t_j)$ (cf. line 5).

B. Forward search

First, we present the standard forward search approach known from the literature and then our novel extensions; the complete approach is summarized in Alg. 2.

We initialize the search tree $\mathcal{T}$ with z_{num} randomly selected safe nodes. In each time step t_k, we apply the ACC control law (2) and the motion equation (1) to all state tuples $\big(x_{\text{acc}}(t_k), x_{\text{lead}}(t_k)\big)$ such that we obtain the set of states of the ACC vehicle $\mathcal{X}_{\text{acc}}(t_{k+1})$ at the next time step (cf. line 6 to 8 of Alg. 2). Then, we randomly generate the future behavior of the leading vehicle by exploring the search space using the RRT of Alg. 1 (cf. line 10 of Alg. 2). Before advancing to the next time step, we can optionally remove childless nodes to reduce the memory consumption.

Our extension, illustrated in Fig. 4, allows us to find the state trajectory leading to a collision more quickly. At each time step during the forward search, we check if we have already generated an unsafe node (cf. line 4 of Alg. 2). Since an unsafe state implies that the ACC vehicle will eventually collide if the leading vehicle fully brakes (cf. Def. 2), we let the leading vehicle perform emergency braking (cf. line 14 of Alg. 2) until we have generated a collision node (cf. Fig. 4).

C. Backward search

Searching backward in time is especially difficult, since we cannot compute the inverse of the ACC control law, which would be required to simulate the ACC vehicle backward in time (unless the ACC system is flat [37]). Thus, we generate random inputs for the ACC vehicle to obtain states of the

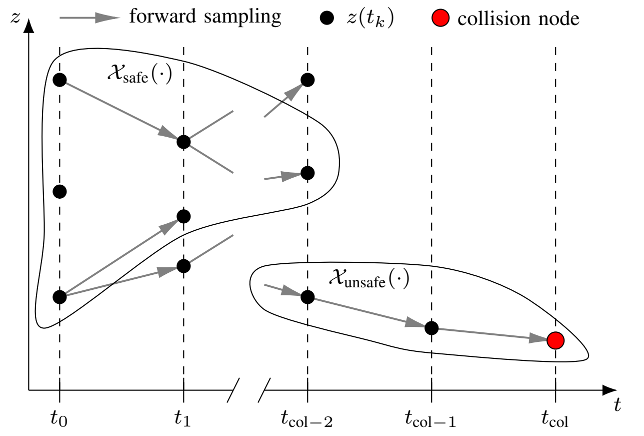


Fig. 4: Falsification by searching forward in time. Starting in safe states at t_0, the RRT is built until an unsafe node is found (i.e., $x_{\text{acc}}(t_k) \in \mathcal{X}_{\text{unsafe}}(t_k)$). Then, the leading vehicle performs emergency braking, which will result in a collision.

Algorithm 2 Falsification by forward search

```
1: $\mathcal{T}$.INITIALIZE($\mathcal{X}_{\text{safe}}(t_0)$, $z_{\text{num}}$)
2: $k \leftarrow 0$
3: while not $\mathcal{T}$.HASCOLLISIONNODE($t_k$) do
4:     $z_{\text{unsafe}}(t_k) \leftarrow \mathcal{T}$.FINDUNSAFENODE($t_k$)
5:     if $z_{\text{unsafe}}(t_k)$ is null then
6:         for all $z(t_k)$ in $\mathcal{T}$ do
7:             $\mathcal{X}_{\text{acc}}(t_{k+1})$.ADD($f_{\text{ACC-MOTION}}(z(t_k), \delta)$)
8:         end for
9:         while $\mathcal{T}$.NUMNODES($t_{k+1}$) $< z_{\text{num}}$ do
10:            $z_{\text{new}}(t_{k+1}) \leftarrow \mathcal{T}$.EXPLORE($t_k$, $t_{k+1}$, $\mathcal{X}_{\text{acc}}(t_{k+1})$)
11:            $\mathcal{T}$.ADDNEWNODE($z_{\text{new}}(t_{k+1})$)
12:        end while
13:    else
14:        $x_{\text{lead}}(t_{k+1}) \leftarrow f_{\text{MOTION}}(z_{\text{unsafe}}(t_k), u_{\text{lead}}^{\text{brake}}(t_k))$
15:        $x_{\text{acc}}(t_{k+1}) \leftarrow f_{\text{ACC-MOTION}}(z_{\text{unsafe}}(t_k), \delta)$
16:        $\mathcal{T}$.ADDNEWNODE($x_{\text{acc}}(t_{k+1}), x_{\text{lead}}(t_{k+1})$)
17:    end if
18:    $k \leftarrow k + 1$
19: end while
20: return $\mathcal{T}$.COLLISIONTRACE
```

ACC vehicle at previous time steps. However, we need to ensure that the ACC system drives the ACC vehicle into unsafe states again, since we are only interested in generating behaviors that lead to a collision.

Fig. 5 and Alg. 3 illustrate our solution. First, we initialize the search tree $\mathcal{T}$ at an arbitrary time t_h with z_{num} randomly selected unsafe nodes. The falsification successfully terminates if we have generated a safe node (with an optional larger headway distance); otherwise, we continue searching backward in time.

To obtain the states of the ACC vehicle at the next time step t_{k-1} (backward in time), we create random inputs for all $x_{\text{acc}}(t_k)$ in $\mathcal{T}$ (cf. lines 4 to 6 of Alg. 3). The new states for the leading vehicle at t_{k-1} are generated by the RRT of Alg. 1 as in the forward search (cf. line 8 of Alg. 3).

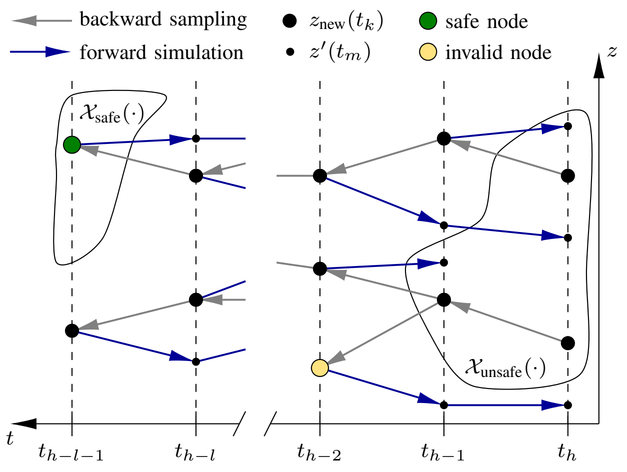


Fig. 5: Falsification by searching backward in time. Starting in unsafe states at t_h, the RRT is built by sampling backward in time until a safe node is found. Yet, a node $z_{\text{new}}(t_k)$ generated by the backward search is discarded as invalid, if applying the ACC control in a forward simulation does not result in a node $z'(t_m)$, $t_k \leq t_m \leq t_h$, in which $x_{\text{acc}}(t_m) \in \mathcal{X}_{\text{unsafe}}(t_m)$.

Algorithm 3 Falsification by backward search

```
1: $\mathcal{T}$.INITIALIZE($\mathcal{X}_{\text{unsafe}}(t_h)$, $z_{\text{num}}$)
2: $k \leftarrow h$
3: while not $\mathcal{T}$.HASSAFENODE($t_k$) do
4:     for all $z(t_k)$ in $\mathcal{T}$ do
5:         $\mathcal{X}_{\text{acc}}(t_{k-1})$.ADD(RANDOMINPUT($z(t_k)$))
6:     end for
7:     while $\mathcal{T}$.NUMNODES($t_{k-1}$) $< z_{\text{num}}$ do
8:         $z_{\text{new}}(t_{k-1}) \leftarrow \mathcal{T}$.EXPLORE($t_k$, $t_{k-1}$, $\mathcal{X}_{\text{acc}}(t_{k-1})$)
9:         for $m \leftarrow k-1$; $m \leq h$; $m$++ do
10:            $z'(t_m) \leftarrow \mathcal{T}$.FORWARDSIM($z_{\text{new}}(t_{k-1})$, $t_m$, $\delta$)
11:            if ISUNSAFENODE($z'(t_m)$) then
12:                $\mathcal{T}$.ADDNEWNODE($z_{\text{new}}(t_{k-1})$)
13:                break
14:            end if
15:        end for
16:    end while
17:    $k \leftarrow k-1$
18: end while
19: return $\mathcal{T}$.COLLISIONTRACE
```

Next, we require to check whether the ACC system will still cause a collision when starting at the new generated node $z_{\text{new}}(t_{k-1})$ (cf. lines 9 to 15 of Alg. 3). Therefore, we iteratively simulate forward in time to obtain $z'(t_m)$ for $t_k \leq t_m \leq t_h$ by applying the ACC control law (2), which uses as input the states of the leading vehicle saved in $\mathcal{T}$ (cf. forward simulation in Fig. 5). Note that $z'(t_m)$ is generally different than $z_{\text{new}}(t_m)$. Only if $z_{\text{new}}(t_{k-1})$ itself or any $z'(t_m)$ is an unsafe node, the new node $z_{\text{new}}(t_{k-1})$ is added to the tree; otherwise, we discard the new node as invalid (cf. invalid node in Fig. 5). The final collision trace is obtained by continuing the forward simulation from the unsafe node with emergency braking of the leading vehicle until the guaranteed collision.

IV. Numerical Experiments

We evaluate our forward and backward search by falsifying different ACC systems. Our implementation in Python is available at gitlab.lrz.de/tum-cps/safety-falsification-acc. All simulations were executed on a machine with an Intel Xeon Gold 6136 3.00 GHz processor and 128 GB of DDR4 2666 MHz memory. The safety-critical scenarios presented next are included in the CommonRoad benchmark suite[1] [3] and are visualized in the video attachment of this paper. Tab. I lists the parameters of the numerical experiments, where n_{forward} and n_{backward} are the user-defined maximum number of iterations of the forward and backward search, respectively.

A. Introduction to tested ACC systems

The four ACC control laws chosen for falsification vary in their safety integrity, as discussed in Sec. I-A. Their control laws are briefly introduced in this section, and their parameters are listed in Tab. II, where v_{des} denotes the desired velocity of the ACC vehicle, t_{des} the desired time gap between the leading and ACC vehicle, and $\Delta s_{\min}$ the desired minimum relative distance. We assume a reaction delay of 0 s to provide a best case situation for the ACC systems.

1) Proportional integral ACC (PI-ACC): The PI-ACC [11] uses a PI controller:

$$u = k_{\text{p}}\big(\Delta v + k_{\text{q}}\Delta s_{\text{err}}^{\text{PI}}\big) + k_{\text{i}}\frac{1}{\Delta t}\big(\Delta v + k_{\text{q}}\Delta s_{\text{err}}^{\text{PI}}\big), \quad (3)$$

[1]commonroad.in.tum.de

TABLE I: User-defined parameters of the falsification.

Description	Parameter with value
Time step size	$\Delta t = 0.1\,\text{s}$
Max. number of time steps	$n_{\text{forward}} = 12000$, $n_{\text{backward}} = 600$
Number of nodes	$z_{\text{num}} = 250$
Increase of sampling range	$\Delta z_{\text{add}}^{\min} = [0\,\text{m},\ 0\,\text{m/s}]^T$,
	$\Delta z_{\text{add}}^{\max} = [1.0\,\text{m},\ 0.25\,\text{m/s}]^T$
Minimal impact velocity	$v_{\text{col}} = 0\,\text{m/s}$
Limit on jerk	$j_{\min} = -10.0\,\text{m/s}^3$, $j_{\max} = 10.0\,\text{m/s}^3$
Limit on acceleration	$a_{\min} = -8.0\,\text{m/s}^2$, $a_{\max} = 1.5\,\text{m/s}^2$
Limit on velocity	$v_{\max} = 50.8\,\text{m/s}$

TABLE II: Parameters of the ACC systems. The values for the PI-ACC, IDM-ACC, and CA-ACC are from [11], [13], and the authors of [14], respectively. The parameters $\Delta s_{\min}$, k_{p}, k_{i}, k_{q}, and b are adapted to increase the collision avoidance performance.

General	PI-ACC [11]	IDM-ACC [13]	CA-ACC [14]
$v_{\text{des}} = 30\,\text{m/s}$	$k_{\text{p}} = 0.2\,1/\text{s}$	$b = 0.02\,\text{m/s}^2$	$K_1 = 0.1\,1/\text{s}^2$
$t_{\text{des}} = 1.5\,\text{s}$	$k_{\text{i}} = 0.1$		$K_2 = 5.4\,1/\text{s}^2$
$\Delta s_{\min} = 3\,\text{m}$	$k_{\text{q}} = 0.1\,1/\text{s}$		$P = 20\,\text{m}$
$\delta = 0\,\text{s}$	$h_0 = 0.1\,\text{s}$		$Q = 1$
	$h_{\text{c}} = 0.2\,\text{s}^2/\text{m}$		

where k_p is a proportional gain, k_i an integral gain, k_q a positive constant factor, and $\Delta s_\text{err}^\text{PI}$ the spacing error $\Delta s_\text{err}^\text{PI} := \Delta s - \Delta s_\text{min} + v_\text{acc} h$. The time headway h favors a larger spacing at higher velocities:

$$h = \begin{cases} 1, & h_0 - h_\text{c}\Delta v \geq 1, \\ h_0 - h_\text{c}\Delta v, & 0 < h_0 - h_\text{c}\Delta v < 1, \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

where h_0 is a constant time headway and h_c a constant factor.

2) Intelligent Driver Model (IDM-ACC): We use the variant of the IDM for increased safety as proposed in [13]. Its control law is given by

$$u = a_\text{max}\left[1 - \left(\frac{v_\text{acc}}{v_\text{des}}\right)^4 - \left(\frac{\Delta s_\text{des}^\text{IDM}}{\Delta s}\right)^2\right], \tag{5}$$

where $\Delta s_\text{des}^\text{IDM} := \Delta s_\text{min} + v_\text{acc} t_\text{des} + \frac{v_\text{acc}\Delta v}{2\sqrt{a_\text{max} b}}$.

3) ACC with collision avoidance (CA-ACC): The CA-ACC [14] uses the control law

$$u = K_1 \Delta s_\text{err}^\text{CA} + K_2 \Delta v R(\Delta s), \tag{6}$$

where K_1 and K_2 are constant gains and $\Delta s_\text{err}^\text{CA} := \min\big(\Delta s - \Delta s_\text{min} - v_\text{acc} t_\text{des}, (v_\text{des} - v_\text{acc}) t_\text{des}\big)$. The error response function $R(\Delta s)$ is

$$R(\Delta s) = \frac{-1}{1 + Pe^{-\frac{\Delta s}{Q}}} + 1, \tag{7}$$

where Q is the aggressiveness coefficient, and P is the perception range coefficient.

4) ACC with safety guarantees (FM-ACC): The FM-ACC [15] is divided into two modes, a nominal control mode and an emergency control mode. The nominal mode uses model predictive control (MPC) and is applied as long as it can satisfy the safe distance. If it cannot find safe input values, the emergency control mode executes a pre-defined emergency deceleration profile so that the ACC vehicle is formally guaranteed to remain collision-free.

B. Forward search

The forward search is able to falsify the PI-ACC and IDM-ACC, but cannot find a counter-example for the CA-ACC and FM-ACC. The state trajectories leading to one of the obtained collisions for the PI-ACC and IDM-ACC are shown in Fig. 6 and Fig. 7, respectively. Tab. III lists the required computation times, the time t_unsafe at which the ACC vehicle first entered the set of unsafe states, and the initial and final states of both vehicles. Even though we set $v_\text{col} = 0\,\text{m}/\text{s}$ (cf. Tab. I), the generated collisions have an impact velocity of $4.6\,\text{m}/\text{s}$ and $2.2\,\text{m}/\text{s}$ for the PI-ACC and IDM-ACC, respectively. In real traffic, the obtained trajectories of the leading vehicle are likely to occur during stop-and-go traffic on a highway.

C. Backward search

The backward search is able to falsify the CA-ACC, even though the forward search finds no counter-example. Fig. 8 shows the trajectories $x_\text{lead}([t_0, t_\text{col}])$ and $x_\text{acc}([t_0, t_\text{col}])$ that are obtained from a backward search; we continued the backward search not only until a safe node is found, but until $s_\text{safe}(t_0) \geq 100\,\text{m}$ and $\Delta s(t_0) \geq 235\,\text{m}$, which significantly increased the required computation time (cf. results in Tab. IV without this addition). Tab. III lists the details of the falsification result. The generated counter-example corresponds to a traffic situation in which the leading vehicle is forced to brake due to a traffic jam. Note that the ACC vehicle could have avoided a collision by braking earlier.

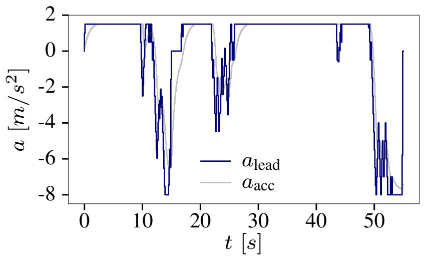
By applying the backward search to the FM-ACC, we were able to identify an error in the code generation of the FM-ACC. After correcting the code generation, the FM-ACC remains collision-free in the backward search, whereas the PI-ACC and IDM-ACC are falsified in every simulation run.

D. Comparison of the computational efficiency

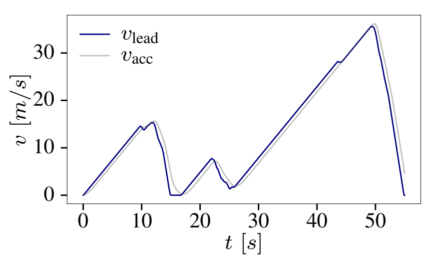
We evaluate the computational efficiency of falsifying an ACC system by comparing the forward search with and without the consideration of unsafe states, backward search, and MCS with each other. To improve the sampling of MCS so that it is more uniform over the input space (since $|a_\text{min}| \neq |a_\text{max}|$), we bias the sampling with a beta distribution $Beta(\alpha = 14,\ \beta = 2)$. All four approaches attempt to find a collision against the PI-ACC, IDM-ACC, and CA-ACC in 100 simulation runs with an iteration limit of $n = 600$, where a run is aborted as soon as a collision node is generated (the remaining parameters are set as presented in Tab. I and Tab. II). The results of the comparison are given in Tab. IV. The standard forward search (without considering unsafe states) finds 4306 transitions into unsafe states for the PI-ACC and 76 for the IDM-ACC, but it does not exploit these situations to generate a collision. As we can see, the consideration of unsafe states drastically improves

TABLE III: Falsification using the forward search for PI-ACC and IDM-ACC and the backward search for CA-ACC.

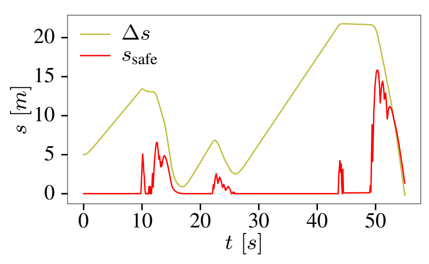
Parameter	**PI-ACC**	**IDM-ACC**	**CA-ACC**
Comp. time	3 min	2 min	16 min
t_unsafe	53.1 s	53.2 s	5.3 s
$a_\text{acc}(t_0)$	$0.0\,\text{m}/\text{s}^2$	$0.0\,\text{m}/\text{s}^2$	$-5.2\,\text{m}/\text{s}^2$
$a_\text{lead}(t_0)$	$0.0\,\text{m}/\text{s}^2$	$0.0\,\text{m}/\text{s}^2$	$-2.1\,\text{m}/\text{s}^2$
$v_\text{acc}(t_0)$	$0.0\,\text{m}/\text{s}$	$9.3\,\text{m}/\text{s}$	$42.9\,\text{m}/\text{s}$
$v_\text{lead}(t_0)$	$0.0\,\text{m}/\text{s}$	$19.5\,\text{m}/\text{s}$	$18.9\,\text{m}/\text{s}$
$\Delta s(t_0)$	5.0 m	8.0 m	237.9 m
$s_\text{safe}(t_0)$	0.0 m	0.0 m	100.0 m
$a_\text{acc}(t_\text{col})$	$-7.4\,\text{m}/\text{s}^2$	$-1.0\,\text{m}/\text{s}^2$	$-8.0\,\text{m}/\text{s}^2$
$a_\text{lead}(t_\text{col})$	$0.0\,\text{m}/\text{s}^2$	$0.0\,\text{m}/\text{s}^2$	$0.0\,\text{m}/\text{s}^2$
$v_\text{acc}(t_\text{col})$	$4.7\,\text{m}/\text{s}$	$1.4\,\text{m}/\text{s}$	$2.7\,\text{m}/\text{s}$
$v_\text{lead}(t_\text{col})$	$0.0\,\text{m}/\text{s}$	$0.0\,\text{m}/\text{s}$	$0.0\,\text{m}/\text{s}$
$s_\text{safe}(t_\text{col})$	1.4 m	0.5 m	0.4 m



(a) Acceleration profiles.

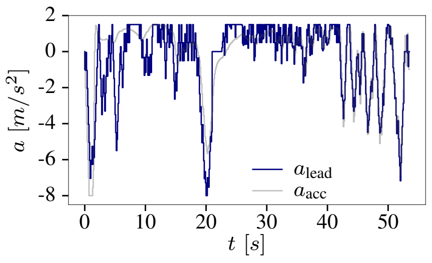


(b) Velocity profiles.

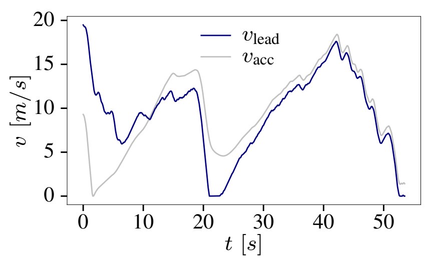


(c) Relative and safe distance profile.

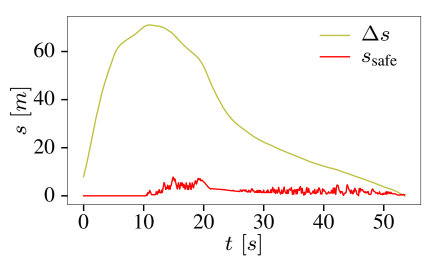
Fig. 6: The forward search finds a trajectory for the leading vehicle so that the PI-ACC causes a collision (CommonRoad ID: S=ZAM_ACC-1_1_T-1:2018b).



(a) Acceleration profiles.

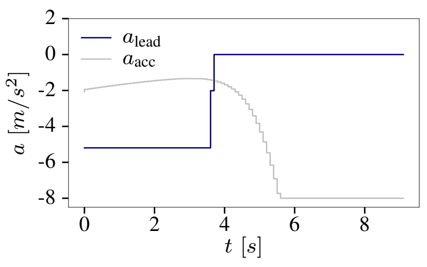


(b) Velocity profiles.

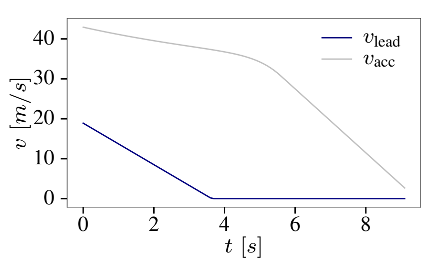


(c) Relative and safe distance profile.

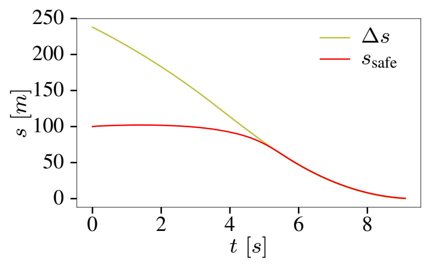
Fig. 7: The forward search finds a trajectory for the leading vehicle so that the IDM-ACC causes a collision (CommonRoad ID: S=ZAM_ACC-1_2_T-1:2018b).



(a) Acceleration profiles.



(b) Velocity profiles.



(c) Relative and safe distance profile.

Fig. 8: The backward search finds a trajectory for the leading vehicle, which is executed forward in time, so that the CA-ACC causes a collision (CommonRoad ID: S=ZAM_ACC-1_3_T-1:2018b).

the efficiency. The advantages of the forward search are its simple setup and large variance of the generated counter-examples. The advantage of the backward search is that it can falsify more ACC systems as compared to the other methods.

V. Conclusions

This paper presents two novel approaches to efficiently falsify the safety of adaptive cruise control systems. By integrating unsafe states in the standard forward search approach, we already achieve an improvement in the required computation time of up to 8 times. With this approach, however, we were not able to falsify all ACC systems in a reasonable time period. In contrast, our backward search approach is able to falsify even the sophisticated ACC system with collision avoidance in every test run. By starting the search from a set of unsafe states, our backward search algorithm is able to find counter-examples 300 times faster than standard approaches.

Our proposed methods allow developers to detect safety flaws in their system at early stages of the development with minimal effort. While the forward search can be used to generate diverse traffic scenarios, the backward search aims to quickly find an unsafe solution. In the future, we would like to add stress testing of the string stability and extend our method to falsify planning systems that combine longitudinal and lateral motions.

Acknowledgments

The authors thank Freddy Antony Mullakkal-Babu for the valuable discussion of our results and gratefully acknowledge partial financial support by the BMW Group within the CAR@TUM project and by the project *interACT* within the EU Horizon 2020 programme under grant agreement No 723395.

TABLE IV: Comparison of the standard forward search without considering unsafe states (S-FS), our proposed forward search with considering unsafe states (FS), our proposed backward search (BS), and Monte Carlo simulation (MCS) in 100 simulation runs with an iteration limit of $n = 600$.

Results for PI-ACC	**S-FS**	**FS**	**BS**	**MCS**
Number of obtained collisions	0	94	100	1
Avg. number of iterations	600.00	81.14	1.08	594.15
Avg. computation time	84.80 s	10.78 s	0.30 s	17.82 s
Results for IDM-ACC	**S-FS**	**FS**	**BS**	**MCS**
Number of obtained collisions	0	13	100	11
Avg. number of iterations	600.00	553.58	1.00	545.51
Avg. computation time	82.88 s	73.77 s	0.45 s	16.25 s
Results for CA-ACC	**S-FS**	**FS**	**BS**	**MCS**
Number of obtained collisions	0	0	100	0
Avg. number of iterations	600.00	600.00	1.08	600.00
Avg. computation time	84.70 s	81.62 s	0.29 s	8.83 s

References

[1] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, *Model-based testing of reactive systems: Advanced lectures*. Springer, 2005.

[2] L. N. Boyle and J. D. Lee, "Using driving simulators to assess driving safety," *Accident Analysis and Prevention*, vol. 42, no. 3, pp. 785 – 787, 2010.

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

[4] R. Kianfar, P. Falcone, and J. Fredriksson, "Safety verification of automated driving systems," *IEEE Intelligent Transportation Systems Magazine*, vol. 5, no. 4, pp. 73–86, 2013.

[5] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivačić, and A. Gupta, "Probabilistic temporal logic falsification of cyber-physical systems," *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 2, pp. 1–30, 2013.

[6] A. Dokhanchi, S. Yaghoubi, B. Hoxha, G. Fainekos, G. Ernst, Z. Zhang, P. Arcaini, I. Hasuo, and S. Sedwards, "ARCH-COMP18 category report: Results on the falsification benchmarks," in *Int. Workshop on Applied Verification of Continuous and Hybrid Systems*, 2018, pp. 104–109.

[7] P. Ioannou, Z. Xu, S. Eckert, D. Clemons, and T. Sieja, "Intelligent cruise control: theory and experiment," in *Proc. of the IEEE Int. Conf. on Decision and Control*, 1993, pp. 1885–1890.

[8] P. A. Ioannou and C. C. Chien, "Autonomous intelligent cruise control," *IEEE Transactions on Vehicular Technology*, vol. 42, no. 4, pp. 657–672, 1993.

[9] L. Xiao and F. Gao, "A comprehensive review of the development of adaptive cruise control systems," *Vehicle System Dynamics*, vol. 48, no. 10, pp. 1167–1192, 2010.

[10] A. Vahidi and A. Eskandarian, "Research advances in intelligent collision avoidance and adaptive cruise control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 4, no. 3, pp. 143–153, 2003.

[11] D. Yanakiev and I. Kanellakopoulos, "Nonlinear spacing policies for automated heavy-duty vehicles," *IEEE Transactions on Vehicular Technology*, vol. 47, no. 4, pp. 1365–1377, 1998.

[12] P. Shakouri, A. Ordys, D. S. Laila, and M. Askari, "Adaptive cruise control system: comparing gain-scheduling PI and LQ controllers," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 12 964–12 969, 2011.

[13] A. Kesting, M. Treiber, M. Schönhof, and D. Helbing, "Adaptive cruise control design for active congestion avoidance," *Transportation Research Part C: Emerging Technologies*, vol. 16, no. 6, pp. 668–683, 2008.

[14] F. A. Mullakkal-Babu, M. Wang, B. van Arem, and R. Happee, "Design and analysis of full range adaptive cruise control with integrated collision avoidance strategy," in *Proc. of the IEEE Int. Conf. on Intelligent Transportation Systems*, 2016, pp. 308–315.

[15] S. Magdici and M. Althoff, "Adaptive cruise control with safety guarantees for autonomous vehicles," in *Proc. of the 20th World Congress of the Int. Federation of Automatic Control*, 2017, pp. 5774–5781.

[16] S. Sadraddini, S. Sivaranjani, V. Gupta, and C. Belta, "Provably safe cruise control of vehicular platoons," *IEEE Control Systems Letters*, vol. 1, no. 2, pp. 262–267, 2017.

[17] A. D. Ames, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs with application to adaptive cruise control," in *IEEE Int. Conf. on Decision and Control*, 2014, pp. 6271–6278.

[18] J. Lunze, "Adaptive cruise control with guaranteed collision avoidance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 5, pp. 1897–1907, 2019.

[19] W. Choi and D. Swaroop, "Assessing the safety benefits due to coordination amongst vehicles during an emergency braking maneuver," in *Proc.of the IEEE American Control Conference*, 2001, pp. 2099–2104.

[20] A. Touran, M. A. Brackstone, and M. McDonald, "A collision model for safety evaluation of autonomous intelligent cruise control," *Accident Analysis and Prevention*, vol. 31, no. 5, pp. 567 – 578, 1999.

[21] A. E. Broadhurst, S. Baker, and T. Kanade, "Monte Carlo road safety reasoning," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2005, pp. 319–324.

[22] A. Eidehall and L. Petersson, "Statistical threat assessment for general road scenes using Monte Carlo sampling," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, pp. 137–147, 2008.

[23] Z. Yang, X. Wang, X. Pei, S. Feng, D. Wang, J. Wang, and S. C. Wong, "Longitudinal safety analysis for heterogeneous platoon of automated and human vehicles," in *Proc. of the IEEE Int. Conf. on Intelligent Transportation Systems*, 2018, pp. 3300–3305.

[24] M. Althoff and S. Lutz, "Automatic generation of safety-critical test scenarios for collision avoidance of road vehicles," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2018, pp. 1326–1333.

[25] I. R. Jenkins, L. O. Gee, A. Knauss, H. Yin, and J. Schroeder, "Accident scenario generation with recurrent neural networks," in *Proc. of the IEEE Int. Conf. on Intelligent Transportation Systems*, 2018, pp. 3340–3345.

[26] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski, "Simulation-based adversarial test generation for autonomous vehicles with machine learning components," in *IEEE Intelligent Vehicles Symposium*, 2018, pp. 1555–1562.

[27] G. E. Mullins, P. G. Stankiewicz, and S. K. Gupta, "Automated generation of diverse and challenging scenarios for test and evaluation of autonomous vehicles," in *IEEE Int. Conf. on Robotics and Automation*, 2017, pp. 1443–1450.

[28] M. Klischat and M. Althoff, "Generating critical test scenarios for automated vehicles with evolutionary algorithms," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2019, pp. 2095 – 2101.

[29] M. O'Kelly, A. Sinha, H. Namkoong, J. Duchi, and R. Tedrake, "Scalable end-to-end autonomous vehicle testing via rare-event simulation," in *Proc. of the Int. Conf. on Neural Information Processing Systems*, 2018, pp. 9849–9860.

[30] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int. Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[31] A. Bhatia and E. Frazzoli, "Incremental search methods for reachability analysis of continuous and hybrid systems," in *Hybrid Systems: Computation and Control*. Springer, 2004, pp. 142–156.

[32] T. Dreossi, T. Dang, A. Donzé, J. Kapinski, X. Jin, and J. V. Deshmukh, "Efficient guiding strategies for testing of temporal properties of hybrid systems," in *Proc. of the NASA Formal Methods Symposium*, 2015, pp. 127–142.

[33] T. Dreossi, A. Donzé, and S. A. Seshia, "Compositional falsification of cyber-physical systems with machine learning components," in *Proc. of the NASA Formal Methods Symposium*, 2017, pp. 357–372.

[34] A. Rizaldi, F. Immler, and M. Althoff, "A formally verified checker of the safe distance traffic rules for autonomous vehicles," in *Proc. of the NASA Formal Methods Symposium*, 2016, pp. 175–190.

[35] M. Althoff and R. Lösch, "Can automated road vehicles harmonize with traffic flow while guaranteeing a safe distance?" in *Proc. of the IEEE Int. Conf. on Intelligent Transportation Systems*, 2016, pp. 485–491.

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

[37] M. Fliess, J. Lvine, P. Martin, and P. Rouchon, "On differentially flat nonlinear systems," in *Nonlinear Control Systems Design*, 1993, pp. 159 – 163.