Computer-Aided Design for Safe Autonomous Vehicles

Matthew O'Kelly, Houssam Abbas, Rahul Mangharam
Department of Electrical and Systems Engineering
University of Pennsylvania
Philadelphia, PA
Email:[mokelly, habbas, rahulm]@seas.upenn.edu

***Abstract*—This paper details the design of an autonomous vehicle CAD toolchain, which captures formal descriptions of driving scenarios in order to develop a safety case for an autonomous vehicle (AV). Rather than focus on a particular component of the AV, like adaptive cruise control, the toolchain models the end-to-end dynamics of the AV in a formal way suitable for testing and verification. First, a domain-specific language capable of describing the scenarios that occur in the day-to-day operation of an AV is defined. The language allows the description and composition of traffic participants, and the specification of formal correctness requirements. A scenario described in this language is an executable that can be processed by a specification-guided automated test generator (bug hunting), and by an exhaustive reachability tool. The toolchain allows the user to exploit and integrate the strengths of both testing and reachability, in a way not possible when each is run alone. Finally, given a particular execution of the scenario that violates the requirements, a visualization tool can display this counter-example and generate labeled sensor data. The effectiveness of the approach is demonstrated on five autonomous driving scenarios drawn from a collection of 36 scenarios that account for over 95% of accidents nationwide. These case studies demonstrate robustness-guided verification heuristics to reduce analysis time, counterexample visualization for identifying controller bugs in both the discrete decision logic and low-level analog (continuous) dynamics, and identification of modeling errors that lead to unrealistic environment behavior.**

1. Introduction

What type of evidence should we require before giving a driver's license to an autonomous vehicle? To answer this question, consider the major components which make up an AV. An AV is typically equipped with multiple sensors, like a LIDAR (a laser range finder) and several cameras. The readings of these sensors are processed by algorithms that extract the content of the current scene (who's doing what where). This information is then fused together to provide the AV with its state estimate (position, velocity, etc) and that of the other agents in the scene. The AV must then decide where to go next (a decision taken by the behavioral planner component of the control stack), what trajectory to follow to get there (a computation performed by the trajectory planner) and how to actuate steering and acceleration to follow that trajectory (performed by the trajectory tracker). Add to this the interaction with other vehicles and the respect of traffic laws, and it is clear that verifying correctness of AV control is a gargantuan task.

The Electronic Design Automation (EDA) industry has a long history of successfully managing the complexity of semiconductor development by providing tools that enable easy design entry, modular design, abstraction, formal equivalency checking between abstractions, automated test generation, formal verification, and the re-use of tests and other artifacts across abstractions. Most of these techniques are applicable to the development of AVs, by utilizing an integrated approach which deploys the appropriate tools as a coherent and traceable whole.

The novelty of the AV domain is that AVs need to operate in a variety of scenarios (e.g., highway driving vs. parking), and will execute different controllers depending on the scenario, Thus, *the AV must be verified in* ***representative scenarios***. Since each scenario can have an infinite variety of instantiations (highways with different curvatures, intersections with different traffic signage), *it is important to obtain good* ***coverage*** *of a given scenario.* The space to be covered includes road varieties and the set of initial states of all agents involved (AVs and human drivers). Safety-criticality implies that *coverage must be* ***rigorously measured*** *or bounded,* rather than let it be set by an arbitrary timeout on the duration of verification. *Thus a* ***formal description*** *of scenarios is needed,* suitable for processing by formal testing and reachability tools. The supported testing and reachability tools must be able to handle the cyber-physical nature of AVs, as they combine vehicle dynamics, constraints imposed by road geometry, traffic laws, discrete supervisory logic, and uncertainty regarding the environment.

Contributions. The toolchain proposed in this paper, *AVCAD*, addresses this need. It provides a scenario description language (SDL) to create driving scenarios with different numbers of agents and on different road topologies (Figure 1 *Panel 1*). It also enables the specification of formal correctness requiements in Metric Temporal Logic [13], a rich specification language similar to SystemVerilog Assertions. The toolchain comes pre-loaded with five driving scenarios, drawn from the 2007 NHTSA accident analysis [18]. The executable scenarios (Figure 1 *Panel 2*) are automatically passed to two verification tools: S-TaLiRo and dReach. S-TaLiRo is a specification-guided automatic test generation tool for cyber-physical systems. It can find *many* different ways in which the AV might violate the specification, thus promoting good coverage of the test space. It is also possible to obtain upper bounds on the probability of missing a *counter-example* (i.e., a requirement violation). dReach is a formal verification tool that can *exhausively* determine whether a hybrid dynamical system violates its specification (Figure 1 *Panel 4*). However, it can only handle smaller-scale scenarios than S-TaLiRo. Thus the two tools are complementary since they handle systems of different sizes and provide different guarantees. AVCAD

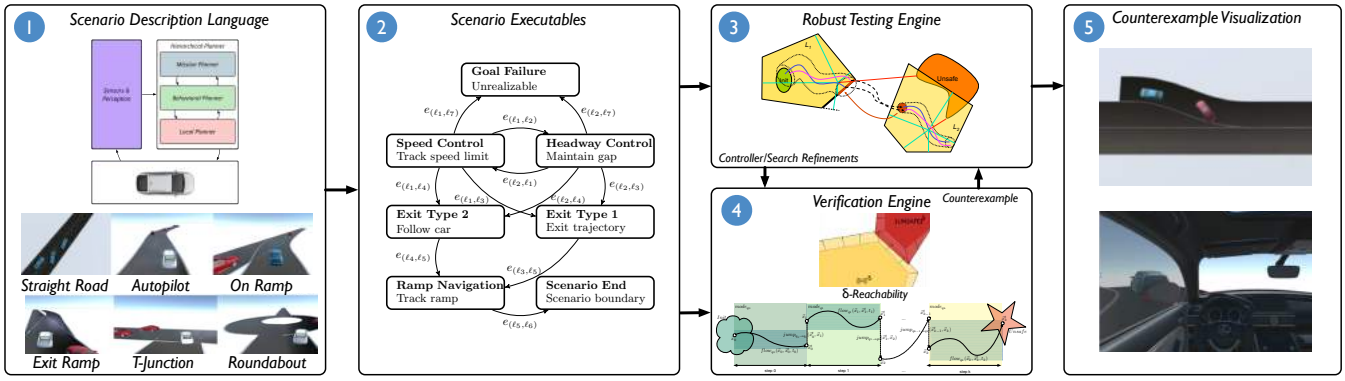


Figure 1: The *AVCAD* toolchain: (1) Scenario Description Language (2) Scenario Executables Generation (3) Robust Testing [G. Fainekos] (4) Verification Engine [S. Kong] (5) Scenario Visualization

also enables a mixed testing-reachability approach, in which testing results guide the reachability analysis and reduce its runtime. Finally, the counter-examples can be visualized in a photorealistic environment (Figure 1 *Panel 5*) aiding in the debugging process and enabling the generation of synthetic camera-based sensor outputs.

Outline. Section 2 describes the Scenario Description Language and presents the formal models of agents and scenarios. Section 3 summarizes the capabilities of the verification engines and the mixed testing-reachability approach. Section 4 presents five case studies in using AVCAD on the pre-loaded scenarios. Section 6 discusses related work and Section 7 concludes the paper.

Notation and terminology. The *ego-vehicle* is the AV under test. For a state variable x, $\dot{x} := dx/dt$. $\mathbb{R}$ is the set of real numbers.

2. Scenario Description Language

The *Scenario Description Language* (SDL) allows the user to quickly specify a driving scenario with the following components. A scenario $\mathbf{S}$ consists of a set of agents, A, that includes the ego-vehicle and other vehicles and the road, a set of traffic laws $\mathcal{L}$, a goal Φ to be achieved by the ego-vehicle in this scenario, exit conditions $\mathcal{E}$ that define when a scenario is over (otherwise the verification tools might not know when to terminate), and a set of initial states $Init$ that captures the initial states of all vehicle agents.

$$\mathbf{S} = (A, \mathcal{L}, \Phi, Init, \mathcal{E})$$

Rather than define these formally, it is best to illustrate them using the T-Junction scenario from the *AVCAD* library depicted in Fig. 2.

Agents (A): There are 3 agents: the `ego-vehicle` (a_1), `environment-vehicle` (a_2), and `road` agent (a_3). The ego-vehicle's model will be detailed in Sections 2.1-2.3. It is unrealistic to assume perfect knowledge of the environment vehicle's intentions and dynamics, the latter must be modeled non-deterministically. For example, the dynamics of the environment vehicle a_2 are given by $\dot{x} = v_x, \dot{v}_x \in [0,1]\frac{m}{s^2}$. So at every moment, a_2 can change acceleration arbitrarily. In general, roads are described as finitely-parameterized curves: e.g., straight lines (with parameter $length$), arcs (parameters $length, radius$), cubic splines (parameters length and curvatures), etc. These basic curves can be linked together using a connectivity graph. Because the elements

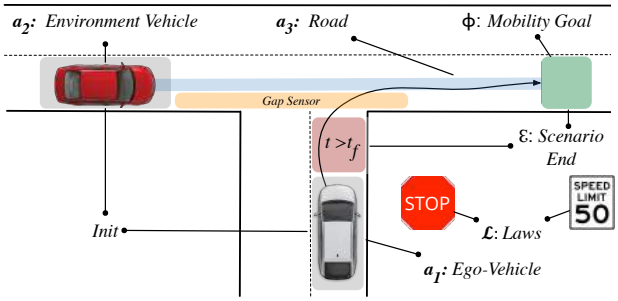


Figure 2: The T-Junction Scenario.

are parameterized they can be initialized within a set. Thus when the scenario is verified, the results are applicable to all modeled behaviors of the environment vehicles and roads, not just one arbitrarily fixed behavior or road.

Law Set ($\mathcal{L}$): The laws are fixed in a given scenario and expressed in Metric Temporal Logic (MTL) [13]. In Fig. 2, one law imposes a speed limit of 50 over for the duration T of the scenario: $l = \text{Always}_{[0,T]}(v < 50)$.

Goal (Φ): The goal Φ of the ego vehicle is always of the form "Ego must reach some region within N time units". In Fig. 2, the goal region is the green rectangle, expressing that the vehicle must exit the highway.

Initialization ($Init$): An AV estimates its state x_0 to within some bounded error (because of measurement imprecision), and so it only knows that x_0 is in some set $Init_a$. Thus, it is necessary at verification time to verify that whatever actual value x_0 has in $Init_a$, the scenario will not lead to a collision or to a requirement violation. The set $Init$ is the product of all vehicles' initial sets: $Init = \Pi_{a \in A} Init_a$. The initial sets of the 2 cars are shown in light grey in Fig. 2.

Exit Condition ($\mathcal{E}$): Finally, the scenario ends either when the ego vehicle leaves the region defined by the T-Junction (the green region in Fig. 2) and proceeds to the next navigation task, or a timeout occurs (the red region in Fig. 2).

The SDL enables the user to describe these elements of a scenario in a consistent and structured manner, and handles many low-level details like maintaining a global clock, monitoring for important events like scenario end, type and dimensionality checking, and sharing of variables

to model one AV perceiving another (e.g., the gap sensor of Fig. 2). The SDL also relieves the user from initially having to create an AV model by providing one by default, described next.

2.1. AV Control and Planning

When the user creates an ego-vehicle agent, the latter is modeled as having the hierarchical control stack [9], [29] shown in Fig. 3 (bottom). It consists of a discrete behavioral planner, a trajectory planner, and a trajectory tracker. This architecture is common, but others have also been proposed. All testing and reachability results apply to this model, briefly described here, with more details in the appendix and [20].

Behavioral Planner (BP): takes the information about the environment and the next destination point (the goal) as an input and outputs the next waypoint which should be reached by the AV. This planner contains the intelligence and decision logic needed, for example, to decide when to change lanes or come to a stop. AVCAD implements a standard 2-mode adaptive cruise controller shown in Fig. 3 [25]. The appendix provides more details. The BP is responsible for the discrete dynamics of the AV. The next planning and control layers are responsible for the continuous dynamics.

Trajectory Planner (TP): given the environment information, the current state x_{init} of the AV and a goal state x_{goal} (which includes the next waypoint produced by the BP), *the trajectory planner* computes constant-velocity reference trajectories $t \mapsto \kappa(t)$ to be followed by the vehicle to get from x_{init} to x_{goal}. Here, $\kappa(t)$ is the curvature of the reference at time t. See Section 2.3 for how AVCAD solves the problem of modeling this optimization-based planner.

Trajectory Tracker (TT): This computes acceleration and steering rate to make the vehicle follow the reference trajectory computed by the trajectory planner. The continuous dynamics of the ego-vehicle are described by a kinematic bicycle model where the components of $\dot{x}$ are: $\dot{s}_x = v\cos(\theta)$, $\dot{s}_y = v\sin(\theta)$, $\dot{v} = a$, $\dot{\theta} = \frac{v}{l}\sin(\delta)$, and $\dot{\delta} = v_{ref}$, where (s_x, s_y) is the position, θ is orientation, v is the longitudinal velocity, δ is the steering angle, and a and v_{ref} are the control inputs (acceleration and reference velocity respectively). The *AVCAD* agents are extensible to incorporate many, but not all design choices. For example, a user may extend an *AVCAD* agent by modifying properties such as the mass, incorporating alternate PID controllers, changing the set of discrete behavioral modes, or even changing the cost function of the trajectory planner.

2.2. Formal model of AV

To perform rigorous reachability analysis on the scenario, *AVCAD* uses a *formal* internal representation of the AV and the agents in the scenario, i.e. a representation with unambiguous mathematical semantics. While several AV simulators exist with much richer models than presented above, the fact that they lack this internal formal representation means that it is not possible to run any formal tools on them. Agents in *AVCAD* have both discrete switching dynamics in the Behavioral Planner and continuous dynamics in the Trajectory Planner and Tracker. Thus the appropriate formal model is the *hybrid automaton*.

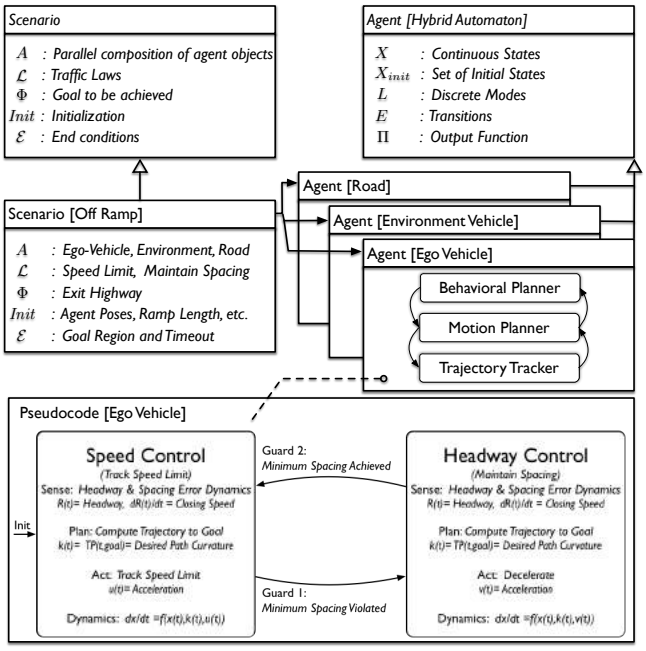


Figure 3: Scenario structure. A white arrow indicates class inheritance. The Off-ramp *scenario automaton* is the product of all *vehicle agents'* automata. The AV automaton in this scenario uses 2 modes (see Fig. 8 for its explanation).

DEFINITION 2.1 (HYBRID AUTOMATON). *A **hybrid automaton** (HA) is a tuple* $\Sigma = (X, Init_\Sigma, L, f, E, Inv, \Gamma, Re, \Pi)$

- $X \subseteq \mathbb{R}^n$ *is the continuous state space of the system and* $Init_\Sigma \subseteq \mathbb{R}^n$ *is its set of initial states (modeling state estimation errors).*
- $L \subset \mathbb{N}$ *is a finite set of control modes that the system switches through (e.g., Headway Control and Speed Control in Fig. 3).*
- *In each mode* $\ell \in L$ *the continuous state vector obeys* $\dot{x} = f(\ell, x)$*. It can stay in this mode as long as* x *is the invariant set of that mode,* $Inv(\ell) \subset \mathbb{R}^n$*. If* x *exits the invariant, it must either switch to a different mode or the simulation stops.*
- $E \subseteq L \times L$ *is the set of mode transitions (a.k.a. switches). The system will switch from mode* ℓ *to* ℓ' *when* x *is in the guard set* $\Gamma(\ell, \ell') \subset X$*. (The guards are assumed disjoint). Upon a transition, the state is reset instantaneously to* $x^+ \in Inv(\ell')$ *given by:* $x^+ = Re((\ell, \ell'), x)$*, where* $Re : E \times X \to X$*.*
- $\Pi : \mathbb{R}^n \to \mathbb{R}^p$ *is an output function. Other AVs can measure* $\Pi(x(t))$*, but don't have direct access to* $x(t)$*.*

In Fig. 3, a high-level view of the SDL's class hierarchy depicts an *AVCAD* scenario as the product of each agent's hybrid automaton.

2.3. Approximation of Trajectory Planner

AVCAD strikes a balance between accurate modeling of AV operation and *verifiability* of the models. Specifically, in the case of the Trajector Planner, AVCAD introduces an

approximation of the TP's outputs to maintain the ability to perform reachability analysis, as detailed below.

Let $x_{goal} \in \mathbb{R}^n$ be the goal state to be reached by the AV. In the Trajectory Planner, the reference trajectories connecting current state to x_{goal} are found by minimizing the following cost over all *spline* trajectories $\kappa : [0, T] \to \mathbb{R}$ that are finitely-parametrized, where $\kappa(t)$ is the curvature of the reference at time t [16].

$$\min_{\kappa} \int_{t_0}^{t_f} \phi(x(t), \kappa(t), t)dt + ||x_{goal} - x(t_f)||^2$$
$$st.\ \dot{x} = f(x(t), \kappa(t)) \quad (1)$$

Here ϕ penalizes large curvatures This optimization cannot be represented within the hybrid automaton formalism (Section 2.2). Moreover, it can't be handled by the reachability tool (dReach) used in AVCAD. Therefore, AVCAD does the following: recall that x_{init} is the current state of the AV and x_{goal} is the goal state. *Offline*, a region in front of the AV is sampled, yielding a set of M possible goals $\{x_{goal,i}\}_{i=1}^{M}$, expressed in relative coordinates. Then for each goal $x_{goal,i}$ the reference trajectory connecting them is computed by solving Eq. (1). Denote the computed reference trajectory by κ_i. Thus we now have a *training* set $\{(x_{goal,i}, \kappa_i)\}_{i=1}^{M}$. A neural network NN_{TP} is used to fit the function $x_{goal,i} \mapsto \kappa_i$. *Online*, given an actual target state x_g in relative coordinates, the AV computes $NN_{TP}(x_g)$ to obtain the parameters of the reference trajectory κ_g leading to x_g. The function NN_{TP} can be used as part of the hybrid automaton formalism. Importantly, NN_{TP} is a composition of linear functions and nonlinearities, and represented in the hybrid systems formalism used in *AVCAD* (Section 3). Further details are provided in [22].

3. Verification Engines

AVCAD enables two methods of system verification: testing and exhaustive reachability analysis. Because an AV combines continuous dynamics (i.e., differential equations) with discrete decision logic (e.g., the behavioral planner), the testing and exhaustive verification engines must be able to handle this added complexity, which is well beyond the complexity of purely digital designs.

Consider a system model H like the one presented above. H captures the dynamics of the AV and all other agents present in the scenario. H is required to satisfy a specification φ written in Metric Temporal Logic (MTL) [13]. MTL is a formal language for expressing complex reactive requirements with real-time constraints, such as "If other car approaches the AV with velocity $> v$ for more than n ms, accelerate within 3 seconds in current lane". Readers familiar with SystemVerilog Assertions will appreciate that MTL is a subset of SVA, yet is expressive enough to describe many properties of interest for AV scenarios. The system state x_0 can start anywhere in a set $X_0 \subset \mathbb{R}^n$. The initial set X_0 models the error with which the AV estimates its state. That is, online, the AV computes a state estimate with a bounded error, and so it only knows that $x_0 \in X_0$. Thus, it is necessary, at verification time, to verify that whatever actual value x_0 has, the scenario will not lead to a collision or to a violation requirement. The purpose of the following two verification tools, S-TALIRO and dReach, is precisely to answer this question: does there exist an initial state $x_0 \in X_0$ s.t. the system's behavior from x_0 violates φ?

3.1. Specification Guided Testing

AVCAD translates the executable scenario created by the user to a format that can be processed by S-TALIRO [4]. S-TALIRO is a tool for automatic test generation for Cyber-Physical Systems (CPS). S-TALIRO uses a stochastic search algorithm, like Simulated Annealing, to find initial conditions, a state x_0, such that the trajectory of H starting in x_0 violates φ. The results are then returned to the designer to debug. Note that if such a falsifying x_0 exists, then S-TALIRO will find it in the long run with probability approaching 1.

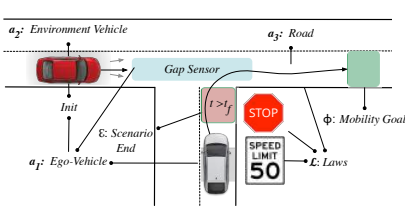
In practice, S-TALIRO can find errors quickly, as will be shown in the experiments. Furthermore, if the system does not violate its specification, S-TALIRO can still estimate how *close* the system gets to a violation. It does so by computing an upper bound on the minimum *robustness* $\rho_\varphi(x_0)$ of φ over X_0 which approaches the true minimum as the number of tests increases. The *robustness* $\rho_\varphi(x_0)$ of φ relative to x_0 [8] is a real number that measures two things: its sign tells whether $\mathbf{x}$ satisfies the spec ($\rho_\varphi(\mathbf{x}) > 0$) or violates it ($\rho_\varphi(\mathbf{x}) < 0$). Moreover, the trajectory starting at x_0 can be disturbed by any amount $|\rho_\varphi(x_0)|$ without changing its truth value (e.g., if it is correct, the disturbed trajectory is also correct). The robustness of MTL specifications has been used successfully for the verification of automotive systems [6], medical devices [27], and general CPS. The theory behind S-TALIRO can be reviewed in numerous publications, such as [7], while the test generation algorithm is presented in detail in [19] and [1].

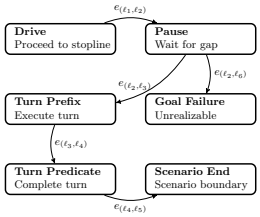
3.2. Delta Reachability Analysis

AVCAD also translates the executable scenario to a format that can be processed by dReach [12], an exhaustive δ-decidability reachability tool. Whereas the testing tool S-TALIRO might be interrupted before having found a violation that does exist in X_0, dReach answers the verification question *exhaustively*: once it terminates, its answer is decisive. Specifically: if dReach returns that the scenario is SAFE, then it is indeed safe. If it returns that the scneario is δ-*UNSAFE*, then this means that a δ-sized perturbation of the scenario's trajectories can reach a slightly expanded unsafe set. The argument then is that in practice, a system that is this close to being unsafe should be considered as unsafe, and its design made more robust.

3.3. Guiding Reachability by Testing

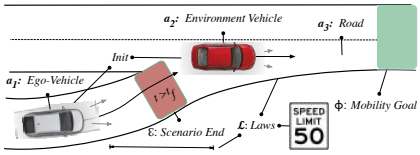
The properties of the testing and exhaustive verification methodologies described above are complementary; thus, a scalable CAD system for AVs should leverage their individual strengths where applicable. *AVCAD* enables simple heuristics to utilize the results of robust simulation from S-TALIRO to guide the reachability analysis of dReach. For instance, regions of the state space that have low robustness relative to the specification φ are good candidates for exhaustive reachability analysis. Instead of feeding the full initial set X_0 to dReach, a smaller sub-region of it, around the initial state of a low-robustness trajectory, is fed to dReach, potentially finding elusive counter-examples faster. This is illustrated in Section 4.3.

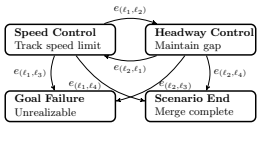




Mode	Transitions		
Drive (ℓ_1)	**Guard** $\Gamma_{(\ell_1,\ell_2)}$: $s_x \geq s_{x_{stop}}$ **Reset** $Re_{(\ell_1,\ell_2)}$: $t' = 0$ **Next State:** Pause	**Guard**: NA **Reset**: NA **Next State:** NA	
Pause (ℓ_2)	**Guard** $\Gamma_{(\ell_2,\ell_3)}$: $(t > t_{pause}) \wedge (d_{gap} > d_{min})$ **Reset** $Re_{(\ell_2,\ell_3)}$: $t' = 0$ **Next State:** Turn Prefix	**Guard**: $\Gamma_{(\ell_2,\ell_6)}$ $(t > t_f) \wedge (\ell = 2)$ **Reset**: NA **Next State:** Goal Failure	
Turn Prefix (ℓ_3)	**Guard** $\Gamma_{(\ell_3,\ell_4)}$: $s_y < s_{fy_1}$ **Reset** $Re_{(\ell_3,\ell_4)}$: $s'_{x_0} = s_x, s'_{y_0} = s_y, s'_{ego} = 0$ $s'_{x_{goal}} = wp_{x_1}, s_{y_{goal}} = wp_{y_1}$ **Next State:** Turn Predicate	**Guard**: NA **Reset**: NA **Next State:** NA	
Turn Predicate (ℓ_4)	**Guard** $\Gamma_{(\ell_4,\ell_5)}$: $s_y < s_{fy_2}$ **Reset** $Re_{(\ell_4,\ell_5)}$: $s'_{x_0} = s_x, s'_{y_0} = s_y, s'_{ego} = 0$ $s'_{x_{goal}} = wp_{x_2}, s_{y_{goal}} = wp_{y_2}$ **Next State:** Scenario Complete	**Guard**: NA **Reset**: NA **Next State:** NA	

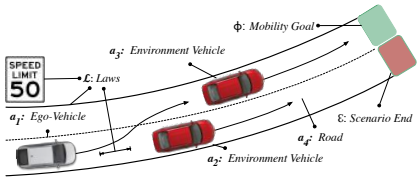
Figure 4: T-Junction: scenario instance details, scenario automaton (middle) and corresponding guards and resets. NA: not present.

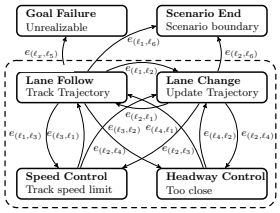




Mode	Transitions		
Speed Control (ℓ_1)	**Guard** $\Gamma_{(\ell_1,\ell_2)}$: $70R + 7.4833\dot{R} - 224.5 \geq 0$ **Reset** $Re_{(\ell_1,\ell_2)}$: NA **Next State:** Headway Control	**Guard** $\Gamma(\ell_1,\ell_3)$: $(t > t_f) \wedge (s_x < s_{x_{goal}})$ **Reset** $Re(\ell_1,\ell_3)$: NA **Next State:** Goal Failure	**Guard** $\Gamma(\ell_1,\ell_4)$: $(t \leq t_f) \wedge (s_x \geq s_{x_{goal}})$ **Reset** $Re(\ell_1,\ell_4)$: NA **Next State:** Scenario End
Headway Control (ℓ_2)	**Guard** $\Gamma(\ell_2,\ell_1)$: $70R + 7.4833\dot{R} - 225.5 \leq 0$ **Reset** $Re(\ell_2,\ell_1)$: NA **Next State:** Speed Control	**Guard** $\Gamma(\ell_2,\ell_3)$: $(t > t_f) \wedge (s_x < s_{x_{goal}})$ **Reset** $Re(\ell_2,\ell_3)$: NA **Next State:** Goal Failure	**Guard** $\Gamma(\ell_2,\ell_4)$: $(t \leq t_f) \wedge (s_x \geq s_{x_{goal}})$ **Reset** $Re(\ell_2,\ell_4)$: NA **Next State:** Scenario End

Figure 5: On Ramp: scenario instance details, scenario automaton (middle) and corresponding guards and resets. NA: not present.





Mode	Transitions			
Lane Follow (ℓ_1)	**Guard** $\Gamma_{(\ell_1,\ell_2)}$: $(R_1 < 4.2) \wedge (R_2 \geq 10)$ **Reset** $Re_{(\ell_1,\ell_2)}$: NA **Next State:** Lane Change	**Guard** $\Gamma_{(\ell_1,\ell_3)}$: $70R_1 + 7.4833\dot{R}_1 \leq 225$ **Reset** $Re_{(\ell_1,\ell_3)}$: $t'_m = 0$ **Next State:** Speed Control	**Guard** $\Gamma_{(\ell_1,\ell_4)}$: $70R_1 + 7.4833\dot{R}_1 \geq 224$ **Reset** $Re_{(\ell_1,\ell_4)}$: $t'_m = 0$ **Next State:** Headway Control	**Guard** $\Gamma_{(\ell_1,\ell_6)}$: $(t \leq t_f) \wedge (s_x \geq s_{x_{goal}})$ **Reset** $Re_{(\ell_1,\ell_6)}$: NA **Next State:** Scenario End
Lane Change (ℓ_2)	**Guard** $\Gamma_{(\ell_2,\ell_1)}$: $s_{ego} \geq s_{traj}$ **Reset** $Re_{(\ell_2,\ell_1)}$: NA **Next State:** Lane Follow	**Guard** $\Gamma_{(\ell_2,\ell_3)}$: $70R_2 + 7.4833\dot{R}_2 \leq 225$ **Reset** $Re_{(\ell_2,\ell_3)}$: $t'_m = 0$ **Next State:** Speed Control	**Guard** $\Gamma_{(\ell_2,\ell_4)}$: $70R_2 + 7.4833\dot{R}_2 \geq 224$ **Reset** $Re_{(\ell_2,\ell_4)}$: $t'_m = 0$ **Next State:** Headway Control	**Guard** $\Gamma_{(\ell_2,\ell_6)}$: $(t \leq t_f) \wedge (s_x \geq s_{x_{goal}})$ **Reset** $Re_{(\ell_2,\ell_6)}$: NA **Next State:** Scenario End
Speed Control (ℓ_3)	**Guard** $\Gamma_{(\ell_3,\ell_1)}$: $(t_m > 0.05) \wedge (\ell_{prev} = 1)$ **Reset** $Re_{(\ell_3,\ell_1)}$: NA **Next State:** Lane Follow	**Guard** $\Gamma_{(\ell_3,\ell_2)}$: $(t_m > 0.05) \wedge (\ell_{prev} = 2)$ **Reset** $Re_{(\ell_3,\ell_2)}$: NA **Next State:** Lane Change	**Guard**: NA **Reset**: NA **Next State:** NA	**Guard**: NA **Reset**: NA **Next State:** NA
Headway Control (ℓ_4)	**Guard** $\Gamma_{(\ell_4,\ell_1)}$: $(t_m > 0.05) \wedge (\ell_{prev} = 1)$ **Reset** $Re_{(\ell_4,\ell_1)}$: NA **Next State:** Lane Follow	**Guard** $\Gamma_{(\ell_4,\ell_2)}$: $(t_m > 0.05) \wedge (\ell_{prev} = 2)$ **Reset** $Re_{(\ell_4,\ell_2)}$: NA **Next State:** Lane Change	**Guard**: NA **Reset**: NA **Next State:** NA	**Guard**: NA **Reset**: NA **Next State:** NA
All Modes (ℓ_x)	**Guard** $\Gamma_{(\ell_x,\ell_5)}$: $(t > t_f) \wedge (\|\|x - g\|\|_2 > r)$ **Reset**: NA **Next State:** Goal Failure	**Guard**: NA **Reset**: NA **Next State:** NA	**Guard**: NA **Reset**: NA **Next State:** NA	**Guard**: NA **Reset**: NA **Next State:** NA

Figure 6: Highway: scenario instance details, scenario automaton (middle) and corresponding guards and resets. NA: not present.

4. Case Studies and Performance

Consider a trip taken by an AV: it begins by navigating a one way residential road, turning onto an arterial road at a T-Junction (Figure 4), then negotiates an on-ramp for a highway (Figure 5), eventually switching lanes to overtake another vehicle (Figure 6). This section details the analysis of the AV control and decision system for each scenario, and shows the complementary nature of specification-guided testing and reachability analysis. Each scenario is written in the AVCAD SDL and analyzed using the toolchain.

4.1. Under-constrained Environment Dynamics

The first example presented utilizes the T-Junction scenario. This case study illustrates the challenges of modeling the environment vehicles without being overly restrictive or overly permissive. Specifically, it reveals that the non-deterministic dynamics of the environment vehicle were under-constrained, allowing it to rear-end the ego-vehicle.

SCENARIO 1 (T-JUNCTION, FIGS. 4 & 7). *The scenario contains three agents: an ego-vehicle, an environment vehicle, and a T-junction road network with a stop sign. The ego-vehicle begins in mode* `Drive` *(ℓ_1), traveling towards the stop sign obeying relevant traffic laws. When it reaches the region where the roads connect it enters mode* `Pause` *(ℓ_2) and comes to a stop. Once there is a sufficiently large gap in the traffic the ego-vehicle begins the turn right (mode* `Turn Prefix`*, ℓ_3). Finally, when it has reached the main road it switches to mode* `Turn Predicate` *(ℓ_4) and aligns itself with the centerline.*

The top panel of Fig. 7 shows a collision, which can be traced back to the dynamics of `env`: indeed the latter could accelerate behind `ego` and rear-end it. In an accident of this nature, the liability always falls on the trailing vehicle's operator. This undesirable behavior can be excised in two ways: either `env` dynamics are constrained to disallow this. Or, the scenario automaton (which is the composition of all agents' hybrid automata) is given a new exit condition, which is triggered precisely when `env` displays such unreasonable behavior. In this case new exit conditions are defined so as not to add further discrete locations to the automaton; thus such false positives are suppressed.

Upon applying new exit conditions another failure mode is identified by S-TALIRO. In this case `ego` fails to yield to an accelerating vehicle, an example is depicted in the bottom panel of Fig. 7. While, dReach is also capable of identifying this failure mode (see Table 1 column 'Controller Error'); S-TALIRO is capable of producing many variations of the crash. In order to correct this failure, the guard in `Pause` is refined to ensure that the environment vehicle's speed is low enough, and there is a large enough gap that the ego-vehicle has time to complete the maneuver. Upon correcting this problem, dReach exhaustively verified the scenario to be SAFE, with added uncertainty in the ego-vehicle's perception in 543 seconds (Table 1 column 'Cor-

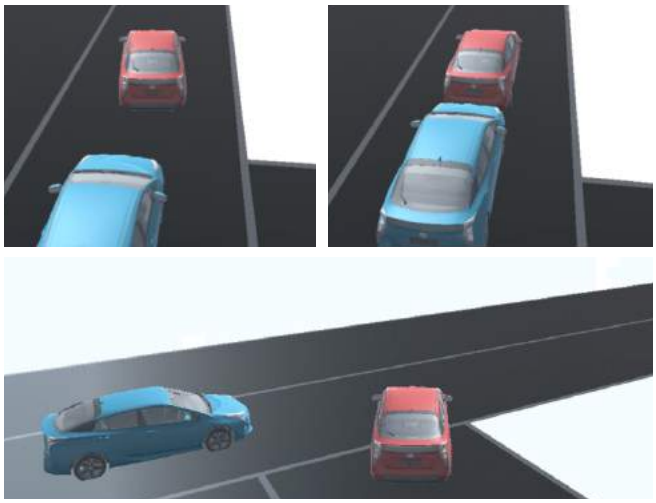


Figure 7: Scenario 1. *Top*: collision due to under-constrained environment vehicle. *Bottom*: collision due to incorrect transitions.

State Variable	Description	Controller Error	Corrected Controller	Robustly SAFE
$s_{x_{env}}$	Environment X-position m	[10,10]	[10,10]	**[9,11]**
$\dot{v}_{env} \in F(\ell)$	Environment Velocity Noise	**[0, 0]**	[-0.5,0.5]	[-0.5,0.5]
$\dot{s}_{x_{env}} \in F(\ell)$	Environment X-Position Noise	[0, 0]	[0,0]	**[-0.5,0.5]**
Runtime (s)	2.6 GHz Intel Core i7, 16 GB RAM	83.622	41.985	542.344
Result	$\delta = 0.1$	δ-**UNSAFE**	SAFE	SAFE

TABLE 1: Verification Results for the T-Junction Scenario. δ-UNSAFE means the scenario is nearly unsafe (A δ-sized perturbation in dynamics or specification leads to collisions)

rected Controller.) dReach certified the scenario as SAFE even with added uncertainty in `env`'s initial set and non-determinism in its dynamics (Table 1 last column).

4.2. Adapting Controllers Between Scenarios

The second scenario investigate is a highway merge maneuver, which adds complexity due to the nature of on-ramp road geometry. This scenario illustrates the complexities of adapting controllers that work in one scenario to a different, seemingly similar, scenario. This motivates the development of CAD toolchains, like *AVCAD*, to ease the creation of scenarios and analyze them from multiple perspectives. It also illustrates how fast semi-formal testing can help flush out many bugs quickly, thus complementing slow reachability analysis, which is used to certify it as bug-free (or to find corner case bugs that only reachability can find).

SCENARIO 2 (ON RAMP, FIG. 5). *There are three agents: the ego-vehicle, an environment vehicle, and a highway on-ramp road network. The on-ramp is a cubic spline. The shape of the on-ramp matters because the tracking performance of the ego-vehicle controllers is altered by sharp curvatures. The ego-vehicle uses a hybrid Adaptive Cruise Controller (ACC) shown in Fig. 8. It enters the* `Speed Control` *(ℓ_1) mode so as to match the speed of the highway traffic. It then transitions to the* `Headway Control` *(ℓ_2) mode if a minimum time to collision constraint is violated.*

The ACC design shown in Fig. 8 has been utilized extensively on real vehicles, but is designed for operating condi-

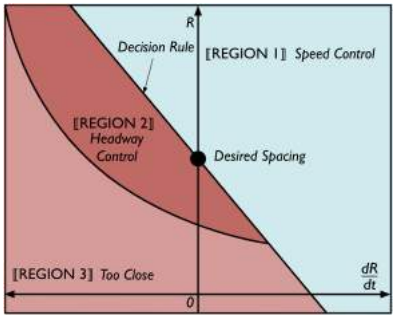


Figure 8: Hybrid Adaptive Cruise Controller. In Speed Control, AV tries to maintain a desired speed. In Headway Control, it tries to maintain a given separation from the leading vehicle. R is the spacing to lead vehicle. In Region 3, the vehicle breaks to avoid collision.

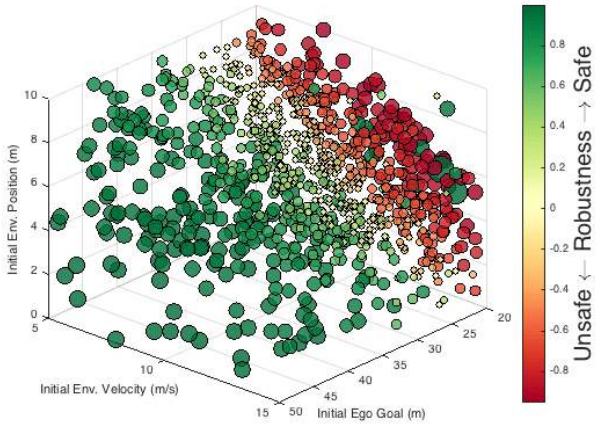


Figure 9: Robustness of On-Ramp scenario as a function of some initial state variables (1000 runs). Color in on-line version.

tions involving highway driving tasks with straight roads. Initial testing on the on-ramp indicates that it produces oscillating reference velocity profiles, because it switches frequently between Headway Control and Speed Control. This makes it difficult for the ego-vehicle to track the reference trajectory, particularly on short on-ramps where road curvature is high. Robust testing in S-TALIRO was able to quickly identify that the design was flawed within 8 seconds. In contrast, dReach also returned δ-UNSAFE, but ran for 5+ hours. This raises the general point that *when analyzing new controller designs or complex environments, robust testing produces interpretable results more quickly that reachability analysis. Once the design and controller composition issues have been addressed in simulation, then reachability analysis can be brought in to certify the scenario as error-free, or find corner case errors.*

Additionally, if a safe design for the entire initial set $Init_\Sigma$ proves to be unrealizable, robust testing can quickly identify potential safe sub-regions. Namely, consider Fig. 9. It shows the robustness of system trajetories as a function of the initial velocity of `env`, the goal region of `ego`, and the x-coordinate of `env`. In Fig. 9 sample points denoted by green spheres are safe. Fig. 9 suggests that the system

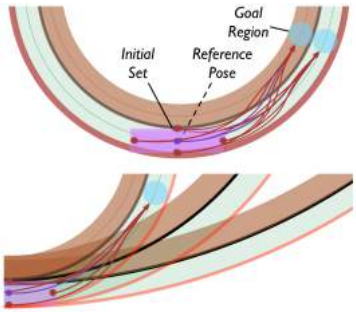


Figure 10: A search over a family of scenario instances. The top panel shows the initial set of the ego-vehicle and a sampling of possible trajectories computed by the TP. The bottom panel shows how varying the curvature changes the road geometry.

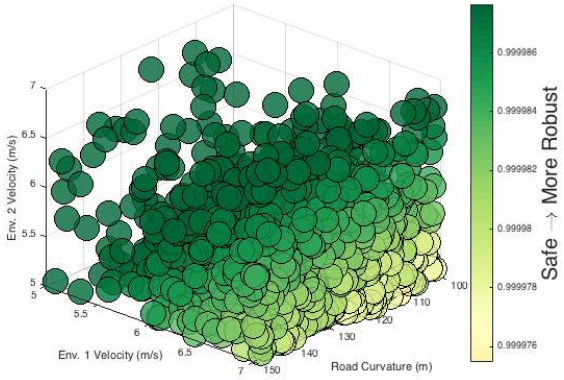


Figure 11: Analysis of samples generated by robust testing details that updating the controller gains yields no counterexamples (10,000 runs)

is robust on longer ramps (Initial ego goal between 39 and 50 meters). dReach is able to prove that this region of the scenario (with full width intervals on all other state variables) is safe in about 3 minutes. *This approach is useful because it can precisely answer regulatory questions such as: under what conditions is the system safe to operate?*

4.3. Controller Gain Error

The last scenario involves a highly parameterized driving task capable of covering most modern highway designs. This scenario demonstrates how *AVCAD* enables a search for violations over road topologies and geometries in order to identify design flaws in the continuous control of the vehicle. This capability is important since the tracking controllers in this domain generally make strong assumptions about the reference trajectory properties and do not investigate the influence of dynamic obstacles.

SCENARIO 3 (HIGHWAY, FIG. 6). *The scenario contains four agents: an ego-vehicle, two environment vehicles, and a parameterized multi-lane road. The highway's curvature is a design parameter that can be varied. In mode* `Lane Follow` *(ℓ_1) the ego-vehicle computes a trajectory to track the current lane. The scenario automaton (which is the product of all agents' automata) shown in Fig. 6 includes the hybrid controller of Fig. 8 for the AV. These aspects of the controller are represented in* `Speed Control` *(ℓ_3) mode so as to match the speed of the highway traffic and* `Headway Control` *(ℓ_4) mode if a minimum time to collision constraint is violated. Unlike the Scenario 2 these modes are briefly visited (with a minimum dwell time to simulate the sampling rate) and an indicator variable is set so that the proper controller is active. In addition, mode* `Lane Change` *(ℓ_2) allows the ego vehicle to overtake a slower moving vehicle in the current lane of travel if preconditions regarding the spacing in the alternate lane are met. When the lane change maneuver is complete the ego-vehicle switches to tracking the new lane.*

Vehicle controller gains are often tuned in the field or simulation after the structure of the controllers has been determined; the original control engineers are not always involved. Defensive code should include assertions regarding the parameter selection, but in practice such guidelines may be ignored, or a fast direct design procedure may not exist. S-TALIRO was used to search for specification violations over all possible curvatures of the highway in a pre-fixed interval. In the case of the hybrid ACC low gains on the headway controller combined with a straight line estimate of the headway R caused the ego-vehicle to undershoot the target acceleration and rear end one of the environment vehicles at a road curvature of 100 (m).

The design was adjusted, increasing the proportional gain of the headway controller and the desired headway time, then the system retested. Over the course of 10,000 runs no counterexamples were found. Fig. 11 summarizes the results (where the axes are: Environment Vehicle 1 Velocity, Road Radius, and Environment Vehicle 2 Velocity). Quantitatively, all sampled trajectories resulted in the ego-vehicle maintaining at least 5.5 m of separation from the other cars. The narrow range of the normalized robutness indicates low variance in vehicle spacing performance.

5. Counterexample Visualization

All of the counter-examples produced in AVCAD can be visualized as videos in the gaming engine UNITY. This allows automotive engineers to understand better what happened (as opposed to looking at multi-dimensional signals against time), and can, in some cases, accelerate the debugging process. Sample videos can be seen at (http://bit.ly/2oRHMG5).

6. Related Work

Other approaches such as theorem proving [14], [15], [26], verification [3], [10], [28], and synthesis [5], [17], [30] also address the safety of autonomous vehicles. The authors of [14], [15], [26] all demonstrate the use of theorem proving for AV safety. The drawback of the approach is that all such techniques require human input and abstract models, making scaling the methodology difficult. Additionally, existing work such as [15] only considers scenarios involving straight roads and utilizes very conservative lemmas (such as, vehicle spacing of at least 291 feet) [24]. In [2] the authors solve the reachability problem online in order to verify online operation of the AV. In our previous work [20] we also consider scenario-based approaches and the use of reachability analysis to verify AV controllers. Additonally,

in [21] we consider a specification-guided testing approach to falsifying safety properties of AVs. In contrast this work builds a larger library of scenarios with multiple dynamic agents, adds realistic road geometries, introduces a flexible scenario description language, and integrates specification-guide testing with exhaustive reachability analysis. Finally, other works [5], [17], [30] address the synthesis of AV controllers from formal specifications. This orthogonal attack largely addresses design at the behavior level only, and when composed with the complete system often must still be verified.

7. Conclusions

Because the operation of an AV involves components from very different domains, verifying its correctness will necessitate the deployment of tools from different disciplines. In this work we demonstrate how a combination of robust testing and reachability analysis can lead to the uncovering of bugs in both the continuous and discrete aspects of the AV controller, as well as modelling errors in the description of uncontrollable agents. Once the bugs are addressed we show several cases where reachability analysis can verify that the system meets the specification. The key is to integrate these tools in a coherent toolchain, in a manner similar to CAD toolchains in semiconductor industry. AVCAD is one such toolchain, integrating a scenario description language, executable scenarios, testing and reachability tools, and rich counter-example visualization. As current tools reveal new errors and hit their limitations, AVCAD forms the basis for an extensible CAD toolset for correctness verification of AVs.

References

[1] H. Abbas, G. E. Fainekos, S. Sankaranarayanan, F. Ivancic, and A. Gupta. Probabilistic temporal logic falsification of cyber-physical systems. *ACM Transactions on Embedded Computing Systems*, 12(s2), May 2013.

[2] M. Althoff and J. M. Dolan. Reachability computation of low-order models for the safety verification of high-order road vehicle models. In *American Control Conference (ACC), 2012*, pages 3559–3566. IEEE, 2012.

[3] M. Althoff and J. M. Dolan. Online verification of automated road vehicles using reachability analysis. *IEEE Transactions on Robotics*, 30(4):903–918, 2014.

[4] Y. S. R. Annapureddy and G. E. Fainekos. Ant colonies for temporal logic falsification of hybrid systems. In *Proc. of the 36th Annual Conference of IEEE Industrial Electronics*, pages 91–96, 2010.

[5] W. Damm, H.-J. Peter, J. Rakow, and B. Westphal. Can we build it: formal synthesis of control strategies for cooperative driver assistance systems. *Mathematical Structures in Computer Science*, 23(04):676–725, 2013.

[6] T. Dreossi, T. Dang, A. Donze, J. Kapinski, X. Jin, and J. V. Deshmukh. A trajectory splicing approach to concretizing counterexamples for hybrid systems. In *NASA Symposium on Formal Methods*, 2015.

[7] G. Fainekos and G. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, September 2009.

[8] G. E. Fainekos, A. Girard, and G. J. Pappas. *Temporal Logic Verification Using Simulation*, pages 171–186. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[9] E. Gat. On three-layer architectures. In *Artificial Intelligence and Mobile Robots*. MIT Press, 1998.

[10] D. Heß, M. Althoff, and T. Sattel. Formal verification of maneuver automata for parameterized motion primitives. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1474–1481, 2014.

[11] Y. Kanayama, Y. Kimura, F. Miyazaki, and T. Noguchi. A stable tracking control method for an autonomous mobile robot. In *ICRA 1990. Proceedings.*, pages 384–389. IEEE, 1990.

[12] S. Kong, S. Gao, W. Chen, and E. M. Clarke. dreach: Delta-reachability analysis for hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015*, pages 200–205, 2015.

[13] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[14] S. Linker and M. Hilscher. Proof theory of a multi-lane spatial logic. In *International Colloquium on Theoretical Aspects of Computing*, pages 231–248. Springer, 2013.

[15] S. M. Loos, A. Platzer, and L. Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In *International Symposium on Formal Methods*, pages 42–56. Springer, 2011.

[16] M. McNaughton. *Parallel Algorithms for Real-time Motion Planning*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2011.

[17] A. Mehra, W.-L. Ma, F. Berg, P. Tabuada, J. W. Grizzle, and A. D. Ames. Adaptive cruise control: Experimental validation of advanced controllers on scale-model cars. In *2015 American Control Conference (ACC)*, pages 1411–1418. IEEE, 2015.

[18] W. G. Najm, J. D. Smith, and M. Yanagisawa. Pre-crash scenario typology for crash avoidance research. In *DOT HS*. Citeseer, 2007.

[19] T. Nghiem, S. Sankaranarayanan, G. Fainekos, F. Ivancic, A. Gupta, and G. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Hybrid Systems: Computation and Control*, 2010.

[20] M. O'Kelly, H. Abbas, S. Gao, S. Shiraishi, S. Kato, and R. Mangharam. Apex: Autonomous vehicle plan verification and execution. *SAE World Congress*, 1, Apr 2016.

[21] M. O'Kelly, H. Abbas, R. Mangharam, S. Dai, B. Kim, J. Shum, Y. Kashiba, and S. Shiraishi. Specification-guided testing and scenario visualization for adas safety analysis (under review). In *ACM ESEC/FSE'17*, 2017.

[22] M. OKelly, V. Pacelli, S. Gao, J. Weimer, and R. Mangharam. Policy approximation with radial basis function networks. Technical report, Scholarly Commons, University of Pennsylvania, April 2017.

[23] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

[24] T. P. Pavlic, P. A. Sivilotti, A. D. Weide, and B. W. Weide. Comments on adaptive cruise control: hybrid, distributed, and now formally verified. *OSU CSE Dept TR22*, 2011.

[25] R. Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[26] A. Rizaldi and M. Althoff. Formalising traffic rules for accountability of autonomous vehicles. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 1658–1665. IEEE, 2015.

[27] S. Sankaranarayanan and G. Fainekos. Simulating insulin infusion pump risks by in-silico modeling of the insulin-glucose regulatory system. In *International Conference on Computational Methods in Systems Biology*, 2012. [To Appear].

[28] O. Stursberg, A. Fehnker, Z. Han, and B. H. Krogh. Verification of a cruise control system using counterexample-guided search. *Control Engineering Practice*, 12(10):1269–1278, 2004.

[29] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[30] T. Wongpiromsarn, U. Topcu, and R. M. Murray. Receding horizon control for temporal logic specifications. In *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, pages 101–110. ACM, 2010.

Appendix

1. Control

In *AVCAD*, a reference tracker is employed to follow paths generated by the local planner. First, we define a fixed frame at the center of the vehicle body. The pose tracking error is derived [11] as:

$$\begin{bmatrix} s_{x_e} \\ s_{y_e} \\ \theta_e \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_{x_{ref}} - s_x \\ s_{y_{ref}} - s_y \\ \theta_{ref} - \theta \end{bmatrix} \tag{2}$$

Given the pose error, we define the path tracking controller [23] $\mathbf{u}(t)$:

$$\begin{bmatrix} a \\ v_\delta \end{bmatrix} = \begin{bmatrix} k_1 s_{x_e} + v_{ref}\cos(\theta_e) \\ \delta_{ref} + v_{ref}(k_2 s_{y_e} + k_3 \sin(\theta_e)) \end{bmatrix} \tag{3}$$

Then the closed-loop error dynamics are defined by the following non-linear ODEs such that $\dot{\mathbf{q}}_{err}$ is [23]:

$$\begin{bmatrix} \dot{x}_e \\ \dot{y}_e \\ \dot{\theta}_e \end{bmatrix} = \begin{bmatrix} (\delta_{ref} + v_{ref}(k_2 y_e + k_3 \sin(\theta_e)))\, y_e - k_1 x_e \\ (-\delta_{ref} + v_{ref}(k_2 y_e + k_3 \sin(\theta_e)))\, x_e + v_{ref}\sin(\theta_e) \\ \delta_{ref} - \delta \end{bmatrix} \tag{4}$$

For these error dynamics a *control Lyapunov function*, $V = \frac{1}{2}(s_{x_e}^2 + s_{y_e}^2) + \frac{1-\cos(\theta_e)}{k_2}$ guarantees path stability for the kinematic bicycle model for $k_1, k_2, k_3 > 0$, $\dot{\delta} = 0$, $\dot{v}_{ref} = 0$ [23].

2. Local Planning

The local planner, derived from [16]. Each execution of the local planner requires as an input the current state of the vehicle and a goal state as defined by the behavioral planner. The local planner's objective is then to find a feasible path from the initial pose to a goal pose The control input $\kappa(s)$ is assumed to be a cubic spline, such that the path can be parameterized, and is defined as a function of arc length s:

$$\kappa(s) = a(\mathbf{p}) + b(\mathbf{p})s + c(\mathbf{p})s^2 + d(\mathbf{p})s^3 \tag{5}$$

Note that there are three free parameters $[b, c, s_f]$ (where a is fixed by the inital condition and s_f is the total arc length of the trajectory).

For any particular (state, goal) pair the problem is a nonlinear program. To find feasible solutions we use an initialization function to produce a parameter assignment.

$$p_0 = \kappa_0 = \kappa_i, p_1 = \kappa_1 = \frac{1}{49}(8b(s_f - s_i) - 26\kappa_0 - \kappa_3) \tag{6}$$

$$p_2 = \kappa_2 = \frac{1}{4}(\kappa_3 - 2\kappa_0 + 5\kappa_1), p_3 = \kappa_3 = \kappa_f \tag{7}$$

Then, the local planner simulates the trajectory, computes the cost, and updates the parameters via gradient descent. Further details are provided in [22].

3. Behavioral Controller

The controller and definitions presented in this section are derived from [25]. Define the hybrid ACC such that:

$$g(\nu) = \begin{cases} a = -k_p(s_x - R) - k_d(v - \dot{R}) & R < R_{ref} + \frac{\dot{R}^2}{2D} \\ a = -k_{p_{sc}}(v - v_{des}) & R > R_{ref} + \frac{\dot{R}^2}{2D} \end{cases} \tag{8}$$