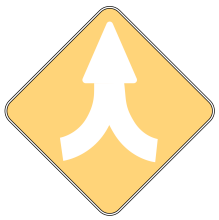

Autonomous Vehicle Plan Execution and Verification: 2015-2016 Report

Matthew O'Kelly, Houssam Abbas, Aditya Pinapala, Rahul Mangharam
University of Pennsylvania, Philadelphia, PA, U.S.A.

February 23, 2016

Executive Summary

Part I: Understanding algorithms and models for AVs

- Creation of realistic planning stack
- Creation of simulation environment for planning stack
- Validation of planning stack on real vehicle

Part II: Verifying scenarios involving lane changes on periodic road structures

- Formalization of vehicle model and environment as a hybrid system
- Interface between planning stack and verification instances
- Proof of chaining for verified decisions
- Infinite time property for periodic roads
- Verification of a lane change scenario

Part III: Alternate formal representations of scenarios via geometric methods, computation modes, and differential inclusions

- Implementation and simulation of a new planning mechanism: pure pursuit
- Addition of differential inclusions to dReach language
- New hybrid systems representation of AV which does not require chaining
- New hybrid systems representation of traffic participant behaviors
- Ongoing work to understand scalability

Part IV: Using our tools, lessons from implementations and installation

- Understanding the limitations of dReach environment in the design process
- New recommended workflow for development of AV algorithms

Part V: Conclusions and Future Work:

- Must build simulation tools
- Likely will need to find clever ways to identify portions of scenarios such that verification is scalable
- Finish building bottom up components such that users can work in top down manner to specify scenarios

Key Results:

- Build and Test Planning Stack
- Formalize planning stack and verify lane change scenario
- Understand alternative stack representations which relax requirements for verification
- Tool available for download

Contents

Part I

Background

1 Introduction

1.1 Software agents are now drivers...

The NHSTA describes Level 4 autonomous vehicles as designed:

> *To perform all safety-critical driving functions and monitor roadway conditions for an entire trip. Such a design anticipates that the driver will provide destination or navigation input, but is not expected to be available for control at any time during the trip. This includes both occupied and unoccupied vehicles. By design, safe operation rests solely on the automated vehicle system.*

In a letter to *Google Inc.* regarding their Level 4 autonomous vehicles, the NHSTA states:

> *Once the AV is determined to be the driver for purposes of a particular standard or test, the next question whether and how Google could certify that the SDS meets a standard developed and designed to apply to a human driver. In order for the NHSTA to interpret a standard as allowing certification of compliance by a vehicle manufacturer, NHSTA must first have a test procedure or other means of verifying such compliance.*

Given this interpretation of the latest AV technology, and the ever-growing gap between the capabilities of current testing frameworks and regulations several pressing questions *must* be answered:

- The NHTSA states AV software is considered a driver. How does one bound the risk posed to society by an autonomous software agent?

- A driver's license is a set of tests, we attempt to generalize human behavior based on tests. AV's don't necessarily fail in predictable ways. Tests cover an infinitesimal portion of the state space. How do we gain confidence that algorithms are sound, free of bugs, or even ethical?

Our perspective encompasses an integrated approach that captures the manifestation of errors in the physical world. At its heart is the use of formal models of system and precise mathematical specifications of desired behavior. Specifically in APEX, we are interested in developing the appropriate formal models, a set of specifications, and a battery of testable scenarios. As part of this work we must investigate how such methods scale, or fail to scale, and provide new solutions and interfaces to the underlying tools.

1.2 How do you give a self-driving car a driver's license?

Each year there are an average of 1.24 million traffic fatalities around the world [19]. Estimates indicate that more than 90 percent of all accidents are due to driver error [19]. Competent autonomous vehicles (AVs) could drastically reduce the occurrence of such incidents, but a major question first needs be answered: how can we judge when an AV is ready to graduate from research laboratories to public roads? One thing is clear: the public will want significant evidence that AVs are indeed safe [**?**]. This raises significant ethical and legal questions about how the AV should behave, and technical questions about how to *verify* that it will always behave the way its designers intended.

Prototype AVs have driven millions of miles and are even being approved for *testing* on public roads in some states [**?**]; however, manufacturers cannot *verify* (i.e., guarantee) the safety of even the simplest of scenarios in the presence of other dynamic traffic participants. Compounding the difficultly of vehicle certification, vehicle manufacturers such as Tesla are transitioning to frequent over-the-air software updates. Such practice eschews conventional vehicle development technique and greatly increases pressure on developers to deliver correct software at a rapid pace.

The net result is a wide gap between current regulations and our technological capabilities. Human drivers are not 'certified' to act safely in

all situations, in fact, we know they don't; however, they assume liability for their actions. Who is liable for the behavior of an AV? The manufacturer or the vehicle owner? Currently, it appears that manufacturers will take one of two approaches: (1) assume liability for the actions of the vehicle and self-insure [1] or (2) force the human occupants of an AV to make all critical decisions and shift liability to the pilot [**?**]. In either case, even if AVs reduce accidents by 99 percent, it is likely that the 1 percent of remaining accidents will invariably spawn a myriad of legal actions against *both* manufacturers [**?**] and vehicle owners. If the legal question is not answered, these risks could stifle the development of the AV market.

Thus, regardless of where legal liability falls it is clear that we first need new, practical methods for verification and validation of the decision engines of each AV. Furthermore, verification must be automatic, exhaustive, and expedient for clearly defined scenarios. Secondly, ethical considerations still underlie both the design and verification of AVs: how does the safety of the car's passengers weigh against that of people in its environment? Whose morals are embedded in the decision engines of an AV?

1.3 New and Old Challenges

Some of the problems facing would-be AV manufacturers in 2016 are very similar to those outlined by the teams in the DARPA Urban Challenge [**?**]. One problem highlighted by the first ever crash between AVs at the Urban Challenge [7] is that there are no known "formal methods that would allow definitive statements about the completeness or correctness of a vehicle interacting with a static environment, much less a dynamic one" [18]. Today one must still consider the massive configuration space of each individual snapshot of the day-to-day life of an AV if verification is to be attempted. For example, in order to test the interaction between two 7 DOF AVs requires 10^{14} simulations for only 10 samples from each state. If each test takes 10 seconds, the resulting set of simulations will take *30 million years* to complete. Furthermore, within a given scenario, errors in localization, sensing, and actuation imply we never know the state of the world exactly. Small state estimation errors can befuddle motion planning algorithms and mean a difference between collision and safety in tight spots. As many teams in the Urban Challenge noted, a means of verifying

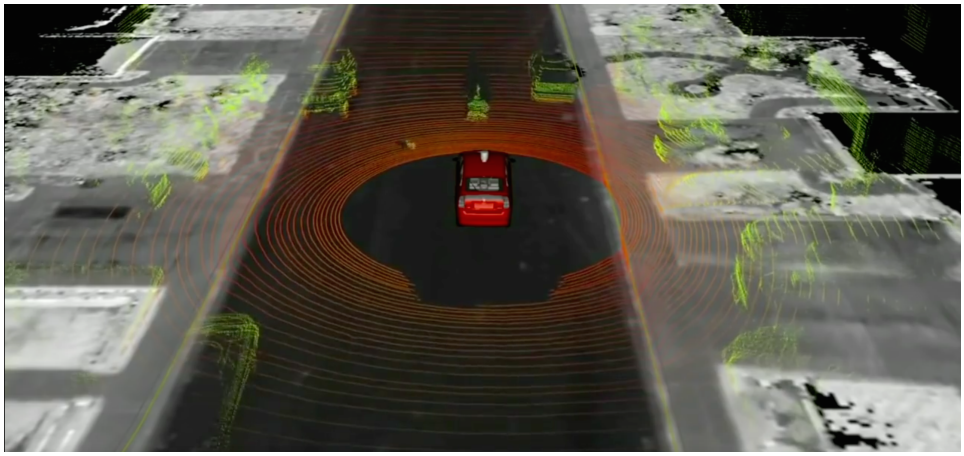

Figure 1: New Challenges: A woman in a wheelchair chasing a duck; autonomous vehicles are driving in real traffic engineers face an extremely long tail of events for which they must provide solutions

the safety of autonomous driving is paramount to putting self-driving cars in the hands of the public [18].

In contrast to the Urban Challenge in 2007, in which only 6 teams out of an original 89 applicants were able to finish, many research labs find themselves in the position of being able to construct a convincing AV in a matter of months; such a vehicle might be relatively competent in 75 percent of the situations it faces, but the long tail of special cases beckons. In fact Sebastian Thrun, a veteran of Google's Self Driving Car project and the Urban Challenge, notes that *there were many more of unusual situations than we believed in the beginning.* One possible solution to handling rare events and scenarios is to record such occurrences using consumer vehicles driving in real traffic; with each new scenario existing behavioral planners can be adjusted via a reinforcement learning scheme [**?**, **?**, 19]. Just as in the more vanilla scenarios, timely updates to controllers provided on the basis of a single example will need to be verified offline, before shipping, and without the thousands of miles of testing necessary for a typical safety feature.

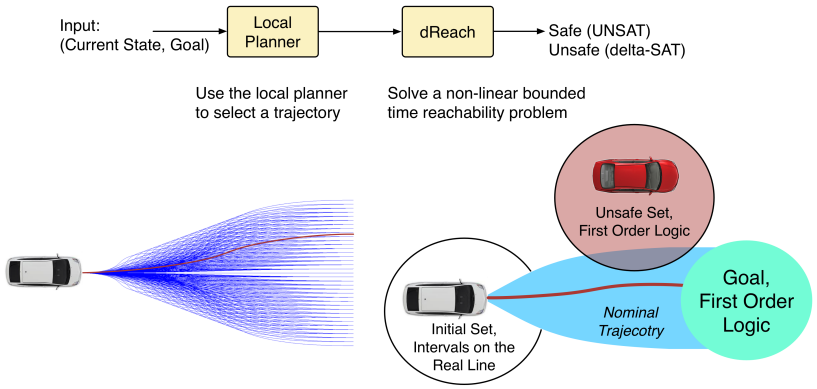

Figure 2: A single offline execution of the APEX tool calls the local planner associated with the AV in order to generate a reachability problem

1.4 The APEX approach

The APEX tool represents a new approach to solving both the problems of the Urban Challenge and investigating rare events encountered only through on road driving and testing. Unlike other tools capable of verifying controllers for hybrid systems we require almost no abstraction. Most current approaches look at only the behavioral layer and assume perfect implementation of plans at the motion planning layer. Instead, we propose that the behavioral layer is used to generate sequences of problems to be investigated at the level of motion planning and trajectory tracking. Thus, APEX addresses the safety verification issue by leveraging new results in hybrid systems and reachability analysis [9] to convert a brute force search over real intervals (which is intractable) into a set of finite sequences of bounded reachability problems.

Using APEX we capture the output of trajectory generation and optimization based methods outside of the formal model of the system. Using such information, we generate a sequence of verification problems by running the through realistic vehicle dynamics and low level controls. The software which runs on the vehicle is used directly for verification. If a rare

event is encountered and recorded by a real vehicle and a new rule or controller is added to the AV it is imperative that the manufacturer have high confidence that the modification will not induce new errors which are not evident in a single trace. APEX can make the most of rare events and scenarios; given a template we could find every possible instantiation and prove it is impossible for solution to make a decision leading to crash. The key features of our approach are:

- Use of realistic planning software which runs on actual vehicles.
- Modular vehicle model construction which can easily be replaced when new algorithms or alternate vehicle dynamics are necessary.
- Non-conservative evaluation of safety at the level of the ego-vehicles actual spatial-temporal evolution.

Thus, verification of realistic scenarios under all possible configurations over a length of 1-7 seconds is potentially feasible. Using such an approach a manufacturer, regulator, or insurer can begin to build a library of scenarios on which to test new vehicle software or updates made to the behavioral layer made through a reinforcement learner.

1.5 Contributions

Our main contribution is a design-time approach to *formally* verifying the trajectory planning and trajectory tracking stacks of an ADAS/AV as they interact with potentially dynamic participants in a *variety* of driving scenarios. This approach is implemented in a software tool, APEX, and illustrated with examples of a lane change maneuver. The verification approach has two characteristics:

- It is formal: we are *guaranteed* that if APEX determines a scenario to be safe, then it is safe. No amount of simulation can find an unsafe behavior in a scenario verified as correct by APEX.
- It allows the use of an arbitrary trajectory planner, for example, it could be code or an abstraction. That is, there is no need to model the trajectory planner, which is often very complex software. Moreover, the same trajectory planner can then be run on a real

vehicle. In the case study presented in this paper, APEX uses a trajectory planner that has been tested on a real vehicle.

In APEX, the verification engineer can

- Specify the low-level dynamics of the vehicle, including the trajectory tracker. Unlike other approaches and existing tools the dynamics can be nonlinear. The default model in APEX is a 7D bicycle model.
- Provide a motion planner that takes in a starting position and end position and returns a trajectory that links the two points. The motion planner can be *any piece of software*: there are no restrictions on it. The default planner in APEX is a state lattice planner incorporated in ROS and tested on a real vehicle. Figure 3 shows the planner GUI available as part of Autoware [11].
- Specify a sequence of goal positions (or *waypoints*) that the vehicle must visit, or a behavioral planner that computes these waypoints in a reactive manner. The default behavioral planner in APEX is a simple 2-state automaton that decides whether to execute lane following or lane changing. However, we expect that designers will implement many other more complex behavioral planners.
- Specify the uncertainty sets for the ego vehicle and the other agents in the scenario.
- Specify the unsafe conditions to be avoided by the vehicle. APEX supports a rich specification language, Metric Interval Temporal Logic (MITL) for the description of unsafe behaviors [4].

APEX will then verify, in an exhaustive fashion, that the ego vehicle can complete the scenario under the specified uncertainty, or return a specific case where it fails. The engineers can then use this *counter-example* in order to debug the controllers, and better understand how to avoid this failure at design-time.

One real-world example of an AV software bug related to plan execution was highlighted by the first ever crash between AVs at the Urban Challenge [7]. At the time of the accident, participants noted that there are no known "formal methods that would allow definitive statements about the completeness or correctness of a vehicle interacting with a static

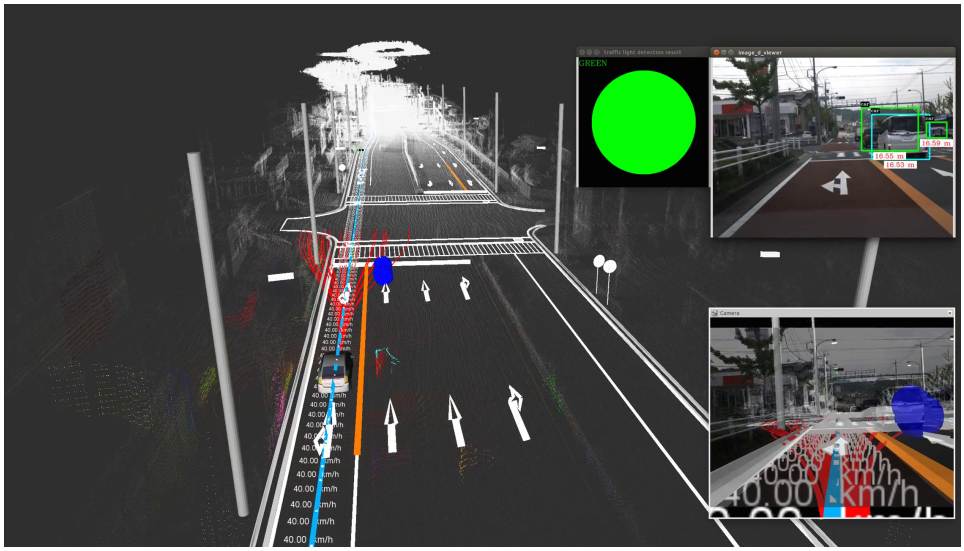

Figure 3: ROS APEX planning implementation GUI.

environment, much less a dynamic one" [18]. It is beyond the scope of this paper to review the numerous developments in verification and synthesis technology; we note attempts exist to reason about the safety of autonomous vehicles in static environments via synthesis [20], but such methods cannot currently scale to realistic systems and are extremely conservative. In response the authors of [20] propose a receding horizon framework, but still rely on coarse grid-based abstractions. Others have sought to verify Adaptive Cruise Control Algorithms (ACC) which severely restrict scenarios in which the car may operate (no lane changes) [15]. Finally, some research which eschews discretization in favor of continuous linearized dynamics focuses on moving the verification task online [3].

Part II

Vehicle Modeling

2 Autonomous Vehicle Software Architecture

In order to motivate the need for the APEX approach, we first outline the architecture of a typical ADAS/AV control system. It is *not* necessary that a vehicle use this *particular architecture* in order to be verified under APEX, but it motivates the key issues involved in obtaining a proof of safety. In the three-layer architecture paradigm [10] which we demonstrate, the planning and control of the vehicle is hierarchical in nature. Each successive layer performs a task over a shorter time horizon. Fig. 4 details this approach to AV architecture.

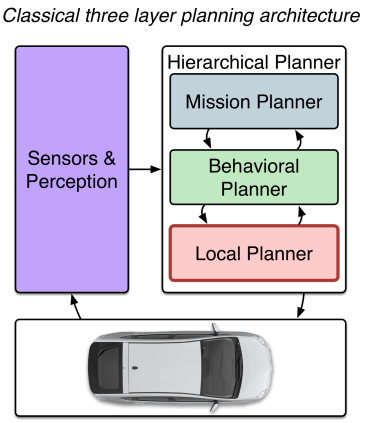

Figure 4: The three layer architecture presented by Gat is widely accepted as a standard means of implementing planning and control for an autonomous vehicle.

At the top level a mission planner is given a mobility goal. Such a goal is typically expressed as a (location, destination) pair. Given this pair the mission planner finds an optimal (or feasible) route through the road network.

In the next layer, the behavioral planner makes local decisions about how to navigate the road network. For example, if the mission planner informs

the behavioral planner that at the next intersection it will need to turn left, the behavioral planner will use a set of rules to determine that the ego vehicle must be in the left lane. It then provides a sequence of waypoints, or intermediary destinations, to the lower-level local planner.

Finally, the local planner, or *trajectory planner*, produces a trajectory that connects the vehicle's current pose to the target pose at the next waypoint. Here 'pose' refers to the combined position, heading and velocity of the vehicle. Specifically, given a goal pose relative to the vehicle's current pose, the local planner computes a set of candidate smooth trajectories that can lead to the goal pose or near it, then selects a single trajectory and sends it to the vehicle. The vehicle itself includes a PID controller (or some other controller) that makes it track the selected trajectory.

3 Building an Autonomous Vehicle Agent

To run APEX, we need to capture the AV dynamics, the low level tracking controller, and the planning stack which generates the trajectories for the vehicle to follow.

3.1 Modeling

The first step towards verification is a model of the AV. APEX uses the formalism of nonlinear hybrid systems to describe the AV and other vehicles. The trajectory tracking controller and AV can be described using ordinary differential equations. The discrete nature of the behavioral control layer dictates that we much capture a system with mixed continuous-discrete dynamics. We provide a list of symbols used in Table 1.

3.1.1 Ego Vehicle Model

APEX uses a non-linear 7 degree of freedom bicycle model [16] in order to describe the ego-vehicle. Higher order models can be supported in the future, and of course the parameters of the base model can be customized in order to match specific vehicles. See Fig. 5. The input to such a model

Table 1: Symbols for Vehicle Model

Symbol List		
Symbol	Units	Description
x_v	-	Verification State Vector
x_{sl}	-	Lattice Planning State Vector
x_p	-	Vehicle Pose
x_g	-	Goal Pose
x_f	-	Predicted Vehicle Pose
p	-	Cubic Spline Parameter Vector
t_f	s	Prediction Horizon
m	kg	Vehicle Mass
l_r	m	Rear Wheelbase
l_f	m	Front Wheelbase
I_z	kg m^2	Moment of Inertia
C_f	N/rad	Front Cornering Stiffness
C_r	N/rad	Rear Cornering Stiffness
β	rad	Slip Angle
Ψ	rad	Heading Angle
v	m/s	Velocity
s_x	m	Position, x
s_y	m	Position, y
δ	rad	Steering Angle
ϵ_x	m	Tracking Error, x
ϵ_y	m	Tracking Error, y
v_w	rad/s	Steering Angle Velocity
a_x	m/s^2	Longitudinal Acceleration
κ	rad/m	Curvature
s_f	m	Arc Length

is steering angle velocity and linear velocity, the output is vehicle state as a function of time.

The state vector describing the vehicle is described in equations (1)-(7). The variable β is the slip angle at the center of mass, ψ is the heading angle, $\dot{\psi}$ is the yaw rate, v is the velocity, s_x and s_y are the x and y positions, and δ is the angle of the front wheel. In the formulation of [6], the inputs to the system are a_x, the longitudinal acceleration, and v_w the rotational speed of the steering angle.

$$x_v = (\beta, \Psi, \dot{\Psi}, v, s_x, s_y, \delta) \tag{1}$$

The state equations for the system as described in [2] are:

$$\dot{\beta} = \left(\frac{C_r l_r - C_f l_f}{m v^2}\right)\dot{\psi} + \left(\frac{C_f}{mv}\right)\delta - \left(\frac{C_f + C_r}{mv}\right)\beta \tag{2}$$

$$\ddot{\psi} = \left(\frac{C_r l_r - C_f l_f}{I_z}\right)\beta - \left(\frac{C_f l_f^2 - C_r l_r^2}{I_z}\right)\left(\frac{\dot{\psi}}{v}\right) + \left(\frac{C_f l_f}{I_z}\right)\delta \tag{3}$$

$$\dot{v} = a_x \tag{4}$$

$$\dot{s_x} = v\cos(\beta + \psi) \tag{5}$$

$$\dot{s_y} = v\sin(\beta + \psi) \tag{6}$$

$$\dot{\delta} = v_w \tag{7}$$

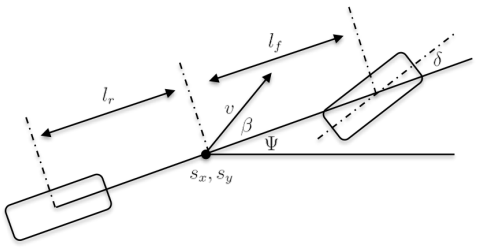

Figure 5: Nonlinear bicycle model describing the statespace for the APEX approach to vehicle dynamics

3.1.2 Vehicle Parameters

The parameters C_f, C_r and l_f, l_r describe respectively the cornering stiffness and distances from the center of gravity to the axles respectively; the subscripts f, r denote whether the parameter is defined for the front or rear of the vehicle. The moment of inertia, I_z and the vehicle mass, m are experimentally determined constants [17]. The kinematic bicycle model considers the two front wheels and two rear wheels of the vehicle to move in unison, with steering provided by the front wheels only. Furthermore, Each abstracted wheel is located along the center of the vehicle's body. Table 2 contains the validated vehicle parameters as given in [2]. It is possible to obtain such parameters and replace these constants in order to investigate specific vehicle characteristics.

Table 2: Parameters of Example Ego Vehicle [2]

Vehicle Parameters					
$m(kg)$	$I_z(kg*m^2)$	$C_f(N/rad)$	$C_r(N/rad)$	$l_f(m)$	$l_r(m)$
2273	4423	10.8e4	10.8e4	1.292	1.515

3.1.3 Tracking Controller

A simple trajectory tracking controller is included with the APEX vehicle model. Trajecotry tracking controllers guide a vehicle along a geometrically defined cubic spline by apply steering and longitudinal acceleration inputs. A successful path tracking algorithm maintains vehicle stability and attempts to minimize the error between the desired trajectory and actual trajectory. The parameters computed for this controller when implemented and validated on a typical crossover SUV [2] are presented in Table 3.

Table 3: Controller Parameters [2]

Controller Parameters					
k_1	k_2	k_3	k_4	k_5	k_6
2	12	4	2	1	1.515

Using the approach in [17] and [2] the control inputs for longitudinal acceleration (pressing the accelerator) and steering angle velocity (turning

the steering wheel) can be computed as v_w and a_x respectively.

$$\begin{aligned} v_w = k_1(cos(\Psi_d)(s_{y,d} - s_y - w_y) - sin(\Psi_d)(s_{x,d} - s_x - w_x)) \\ +k_2(\Psi_d - \Psi - w_\Psi) \\ +k_3(\dot{\Psi}_d - \dot{\Psi} - w_\psi) - k_4(\delta - w_\delta) \end{aligned} \tag{8}$$

$$\begin{aligned} a_x = k_5(cos(\Psi_d)(s_{x,d} - s_x - w_x) + sin(\Psi_d)(s_{y,d} - s_y - w_y)) \\ +k_6(v_d - v - w_v) \end{aligned} \tag{9}$$

We note that we cannot use traditional linear systems techniques or sum of squares optimizations to directly find a Lyapunov function for this system because of the obvious non-linearity and non-polynomial form of the governing ordinary differential equations. Instead we will seek to show stability and safety properties using reachability and model checking analysis.

3.1.4 Planning

In APEX we provide a validated planning stack which can be run on a real vehicle. The planning strategy is hierarchical and includes: mission planning, behavioral planning, and local planning. In this section we will focus on the local planner because it is the layer which connects directly to the tracking controller for the vehicle. The local planner is used to generate smooth trajectories which a non-holonomic dynamically constrained vehicle is capable of following. Our planning stack utilizes the methods outlined in [13] commonly known as state-lattice planning with cubic spline trajectory generation.

Each execution of the planner requires as an input the current state of the vehicle and a goal state as defined by the behavioral planner. We note that we will call the vehicle state x_{sl} because it does not necessarily have to be the same as the model used for verification (although it can be); because the planner must run online, in real-time, lower order models are often substituted here. In this implementation we define x_{sl} as:

$$x_{sl} = (s_x, s_y, v, \Psi, \kappa) \tag{10}$$

Where s_x and s_y are the x and y positions of the center of mass, v is the velocity, Ψ is the heading angle, and κ is the curvature. We note that the

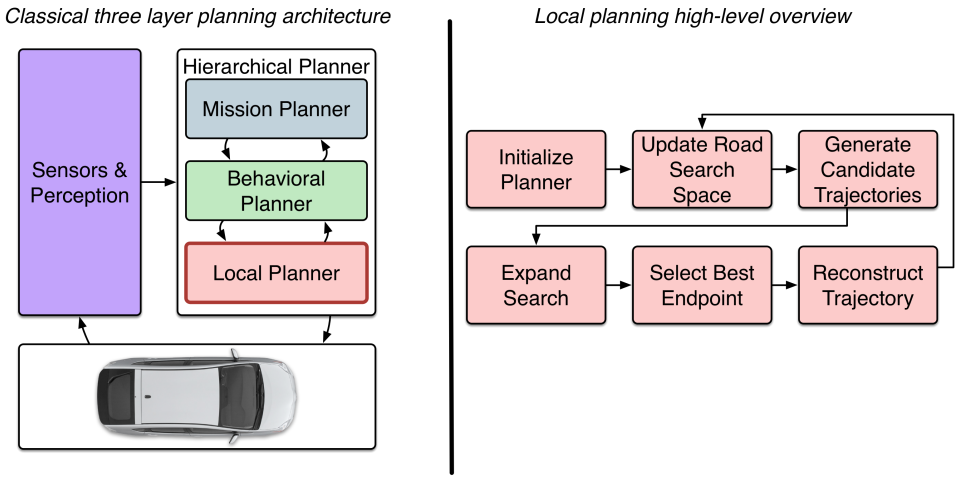

Figure 6: Details of a local planning algorithm and used by AVs employing state lattice planning

state equations involve an additional constant, L which is the wheelbase of the vehicle. Where the state equations are described as:

$$\dot{x} = v * cos(\Psi) \tag{11}$$

$$\dot{y} = v * sin(\Psi) \tag{12}$$

$$\dot{\theta} = \kappa * v \tag{13}$$

$$\dot{\kappa} = \frac{\dot{\Psi}}{L} \tag{14}$$

The local planner's objective is then to find a feasible trajectory from the initial state defined by the tuple x_{sl} to a goal pose x_p defined as:

$$x_p = (s_x, s_y, \Psi) \tag{15}$$

In this formulation we limit trajectories to a specific class of parameterized curves known as cubic splines. A cubic spline is defined as a function of arc length:

$$\kappa(s) = \kappa_0 + a\kappa_1 s + b\kappa_2 s^2 + c\kappa_3 s^3 \tag{16}$$

Note that there are four free parameters (a, b, c, s_f) and our goal posture has four state variables. Thus, a cubic spline is a minimal polynomial that

can be assured to produce a trajectory from the current position to the goal position (if it is kinematically feasible). For any particular state, goal pair there are two steps necessary to compute the parameters. First, it is necessary to produce an initial guess. There are several approaches available such as using a neural network, lookup table, or a simple heuristic. In this case we adapt a heuristic from Nagy and Kelly [14] such that it is compatible with a stable parameter formulation presented by McNaughton [13]. The stable reparameterization is defined as:

$$\kappa(0) = p_0 \tag{17}$$

$$\kappa(s_f/3) = p_1 \tag{18}$$

$$\kappa(2s_f/3) = p_2 \tag{19}$$

$$\kappa(s_f) = p_3 \tag{20}$$

Where the parameters (a, b, c, s_f) can now be expressed as:

$$a(p) = p_0 \tag{21}$$

$$b(p) = -\frac{11p_0 - 18p_1 + 9p_2 - 2p_3}{2s_f} \tag{22}$$

$$c(p) = \frac{9 * (2p_0 - 5p_1 + 4p_2 - p_3)}{2s_f^2} \tag{23}$$

$$d(p) = -\frac{9(p_0 - 3p_1 + 3p_2 - p_3)}{2s_f^3} \tag{24}$$

Which results in the following initialization heuristic:

$$p_0 = \kappa_0 = \kappa_i \tag{25}$$

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

$$p_2 = \kappa_2 = \frac{1}{4}(\kappa_3 - 2\kappa_0 + 5\kappa_1) \tag{27}$$

$$p_3 = \kappa_3 = \kappa_f \tag{28}$$

Finally, with an initial guess in hand, and a stable re-parameterization the local planner can solve a simple gradient descent problem to drive the vehicle to the goal posture.

Thus, we can now compute a set of parameterized trajectories which may each be evaluated to test for safety and optimality. A description of these

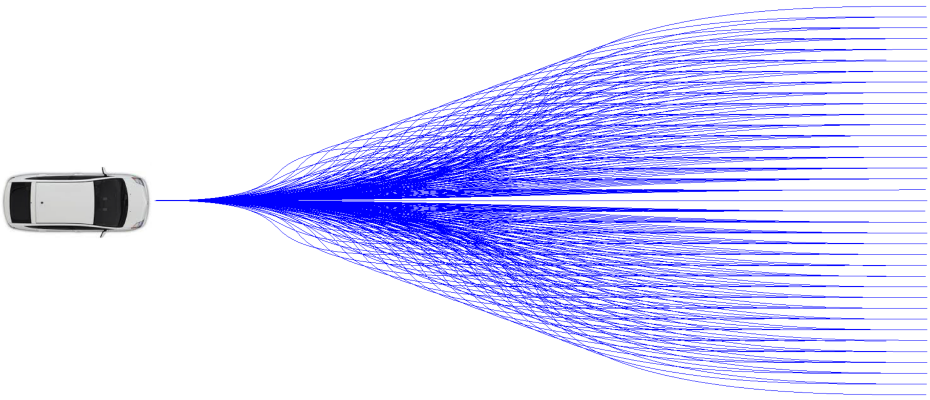

Figure 7: Output of an execution (10 Hz) of the trajectory generator, a single trajectory will be chosen from this set.

aspects of the planner may be found in [13] and such a cost function can obviously be modified based on the goals of the design team. We note that our algorithm implementation is parallelized using OpenMP such that multiple trajectories (with goals regularly sampled around the initial goal) may be evaluated simultaneously. Furthermore, with small changes we can also support quintic splines which expand the variety of possible maneuvers and are more suitable for high speed driving. Figure 7 shows an example of a trajectory generation instance.

Part III

Verification of Lane Change Scenario

4 Specification

Formal verification requires both a system model and a specification. This means that the project stakeholders must provide an exact definition of the desirable system properties. Furthermore, it is often the case that such properties are expressed as occurring only under certain conditions. For convenience we provide the symbols used to describe the vehicle specification in Table 4.

Table 4: Symbols for Specifications

Symbol List		
Symbol	Units	Description
k	-	Search Depth
ϕ	-	Ego Vehicle Spec
ξ	-	Environment Spec
LC	Boolean	Lane Change Request
LO	Boolean	Lane Occupied
v_{ego}	m/s	Velocity of Ego Vehicle
$s_{x_{ego}}$	m	Position of Ego Vehicle, x
$s_{y_{ego}}$	m	Position of Ego Vehicle, y
v_{limit}	m/s	Speed Limit
$s_{x_{ref}}$	m	Centerline Reference, x
$s_{y_{ref}}$	m	Centerline Reference, y
$w(s_{x_{ref}}, s_{y_{ref}})$	m	Lane Width
B	m	Buffer
r	m	Collision Radius
t	s	Current Timestep
t_{max}	s	Max Timestep
$\Box$	-	Always
$\rightarrow$	-	Implies
$\neg$	-	Not
$\wedge$	-	And
$\vee$	-	Or

An example specification follows: the ego vehicle should drive in the selected lane at the speed limit *unless* a stop sign is encountered. We note that the traffic laws of a given region provide a partial, but informal definition of many of the high level specifications which the ego vehicle should adhere to.

4.1 Ego Vehicle Specifcation

The specification for the ego vehicle has two components: safety properties and liveness properties. A specification for the ego vehicle in the case study follows:

- The ego vehicle travels at a velocity less than or equal to the speed limit

$$\square\left(v_{ego} \leq v_{limit}\right) \tag{29}$$

- The ego vehicle does not drive backwards

$$\square\left(v_{ego} \geq 0\right) \tag{30}$$

- The ego vehicle does not collide with any of the n other objects in the environment

$$\square\left(\sqrt{(s_{x_{ego}} - s_{x_{env_i}})^2 + (s_{y_{ego}} - s_{y_{env_i}})^2} \geq r\right)$$
$$\forall i = 1...n \tag{31}$$

- If a timed lane change request is invoked, the ego vehicle completes the lane change on time.

$$\square\left(LC \rightarrow \left(s_{y_{ego}} > w\right) \wedge \left(t \leq t_{max}\right)\right) \tag{32}$$

4.2 Environment Specification

The other vehicles operating within a scenario present both an interesting challenge and a primary motivation for formal verification. It is clear that it is impossible to know the intentions of the agents operating such vehicles; their execution represents a significant source of non-determinism. In fact, a more complex model of such agents which includes details such as steering angle or tire friction will not enable less conservative results, for it is the control input not the plant that remains the largest unknown. Thus, we conclude that: *for verifying the autonomous agent, only the perceptible behavior of other agents is important, not their internal structure.*

Still it remains clear that *the behavior of other agents must be part of the scenario description.* As such we present a safety case which assumes that other agents will follow a certain minimal set of driving rules. For brevity we will reference the following specification as ξ in the case studies.

- Acceleration ceases when some maximum velocity is reached.

$$\square\left(v_{env} \geq v_{max} \rightarrow a = 0\right) \tag{33}$$

- Other agents must drive in the proper direction according to their lane.

$$\Box\left(v_{env} \geq 0\right) \tag{34}$$

- The accelerations of other agents are within those rates achievable by maximum engine power

$$\Box\left(a_{env} \leq a_{max}\right) \tag{35}$$

- Other agents maintain their lanes unless explicitly specified not to.

$$\Box\left(\neg LC \rightarrow \left(y_{min} \leq s_{y_{env}}\right) \wedge \left(y_{max} \geq s_{y_{env}}\right)\right) \tag{36}$$

- Lane changes by other agents are only permitted if the alternate lane is unoccupied or unless a degenerate scenario is being modeled.

$$\Box\left(LO \rightarrow \neg LC\right) \tag{37}$$

5 APEX internals and theory

APEX maintains an internal representation of the scenario as a *hybrid system*. The components of this hybrid system are:

- The behavioral planners of all vehicles involved, $\mathcal{B}_1, \ldots, \mathcal{B}_m$. Fig. 8 shows the behavioral planner we used in the case study for a lane change. A behavioral planner is a finite state system. We will refer to each state of a behavioral planner as a *mode*.

- For every vehicle, the continuous dynamics involved in each of the modes of its behavioral planner. In general, different modes may require different dynamics: e.g. a Collision Avoidance mode which is invoked when a collision is imminent requires more stability control than a turn at a low speed. The continuous dynamics are given in terms of Ordinary Differential Equations (ODEs) $\dot{x}_i = f_i(x_i)$, where $x_i \in R^n$ is the continuous state of the i^{th} agent.

- For each vehicle, transition conditions between the modes of the behavioral planner $\mathcal{B}_i$ are expressed in terms of the state vector x_i. The planner transitions between two modes q and q' only if a *guard*

condition $G_{q,q'}$ is satisfied. In general, the guard condition for $\mathcal{B}_i$ is expressed as a set in the state space of *all the agents*, since transitions will occur based on, for example, how close two vehicles are to each other. Specifically, let $x = (x_1, \ldots, x_n)$ combine the states x_i of the individual vehicles. So $x \in R^{n \cdot m}$. Then there's a transition between two states q and q' of $\mathcal{B}_i$ only if $x \in G_{qq'} \subset R^{n \cdot m}$. For example, there's a LF-to-LC transition only if the two cars are closer than $10m$ and the following car is faster than the leading car. In this case $G_{LF,LC} = \{x \mid ||x_1 - x_2|| \leq 10 \wedge v_2 > v_1\}$.

Together, these make up a hybrid system, so-called because it combines discrete dynamics in the behavioral planner with continuous dynamics in each mode. We will refer to the n hybrid systems of the n agents in the scenario as $H_1, \ldots, H_m$. The *state of the scenario* x is simply $x = (x_1, \ldots, x_n)$.

APEX does not keep an internal representation of the motion planner. Rather, as explained in earlier sections, APEX issues calls to the motion planner in the course of the verification, and obtains a trajectory from it.

APEX also needs to maintain a description of the scenario specification. This specification is provided by the user and can be any formula in first-order logic over the set of modes and states of all agents. See the *Case Study*.For example the following is a possible specification:

$$Mode_1 = LC \rightarrow |\dot{\psi}| \leq b$$

The following sections describe how APEX verifies a property of the scenario using this internal representation.

5.1 Execution tree and formal model

Let $\mathcal{B}$ be a behavioral planner of a given vehicle. The formal model of the behavioral planner is a finite transition system $\mathcal{B} = (Q, q_0, \Sigma, \rightarrow)$ where Q is the finite set of modes, q_0 is the initial mode, Σ is a set of output labels, and $\rightarrow \subset Q \times \Sigma \times Q$ is the labeled transition relation of the system. We write $q \xrightarrow{\sigma} q'$ for $(q, \sigma, q') \in \rightarrow$. Fig. 8 shows the behavioral planner that is used by APEX by default for modeling a lane change controller. It can be described as $\mathcal{B} = (\{LC, LF\}, LF, R^n, \{(LF, LF), (LF, LC), (LC, LF)\})$. In

mode LF, the vehicle's goal is to follow the current lane. In mode LC, the vehicle's goal is to change lanes. In general, a mode represents a decision by the controller, a *behavior* that the vehicle should follow. With every transition between modes, the behavioral planner outputs a vector x_B in R^n: this is the destination that the vehicle must reach. The planner transitions between modes when certain *guard conditions* are satisfied.

The behavioral planner advances in discrete time. The discrete time advances, for example, with every update of the vehicle's sensors. Thus $\mathcal{B}$ makes a decision on what to do everytime its information about the environment is updated. The planner may decide to maintain the current decision, i.e., stay in the same mode, if that mode has a self-loop. Mode LF has a self-loop in Fig. 8. Let $\Delta t > 0$ be the update period. Since every scenario is time-limited, and every transition takes fixed non-zero time Δt, there is a natural limit D on the number of decisions, or transitions, that can be taken in any given scenario.

In the first step of the verification process, APEX builds an execution tree: the root of the tree is the initial mode q_0, and every branch of the tree represents one possible sequence of decisions, i.e. one possible execution of $\mathcal{B}$. See Fig. 10 for the execution tree of the behavioral planner of Fig. 8. Since the number of transitions is bounded by D in a given scenario, this tree has a depth at most D.

With the execution tree built, APEX must next verify that the sequence of decisions taken by the behavioral planner can be implemented by the low-level controllers. E.g., let (LF, LF, LC) be a sequence of decisions of depth 3. In every occurrence of LF, APEX must check that the vehicle can indeed follow the lane, and in every occurrence of LC, APEX must verify that the vehicle can indeed change lanes. In the next section, we define what it means to 'follow the lane' and 'change lanes' via the motion planner.

5.2 Calling the motion planner

After building the execution tree, APEX starts executing every branch, starting at the root, which is the initial mode q_0. The initial set of continuous states is X_0. A transition is taken if the initial set intersects its guard. Since X_0 may intersect more than one guard, then more than one

transition are possible. APEX explores all transitions (all branches) in the execution tree. In each mode APEX enters, $\mathcal{B}$ will output a destination x_B. Formally, x_B is a scenario state, but in what remains, it is simpler to think of it as the position that the ego vehicle must reach.

APEX then calls the motion planner to obtain the trajectory that the vehicle will follow. Since the current state is only known as a set X_A, APEX sets the starting point of the trajectory to be the center x_A of X_A. The motion planner then returns a trajectory starting at x_A and ending in a neighborhood of x_B. The neighborhood shape and size are known to APEX and are part of the motion planner's description. Let that neighborhood be X_B. Note that APEX does not place any restrictions on the motion planner's operation and calls it as a black box. Therefore, the actual motion planner that is used on the real car can be used in the verification of the system. In this way the verification results are directly applicable to the actual deployed software.

5.3 Verifying each trajectory

Once a trajectory is generated connecting $x_A \in X_A$ to the neighborhood X_B of x_B, it remains to verify that the ego vehicle will always reach X_B within a specified amount of time T, regardless of where it starts in X_A. To verify that the specification is satisfied, APEX builds a *reachability problem*. This reachability problem is characterized by the following:

- The system: in this case, the system consists of the scenario hybrid system.
- The target set: this is the set that the system should reach. In this case the state of the ego vehicle x_1 should reach X_B, and there are no target sets for the other agents.
- The unsafe set: this is the set that the scenario hybrid system must *not* reach at any point in time. In this case, the ego vehicle must not get closer than d_{min} to any other agent in the scenario.
- A time bound: the target set must be reached within a certain amount of time T.

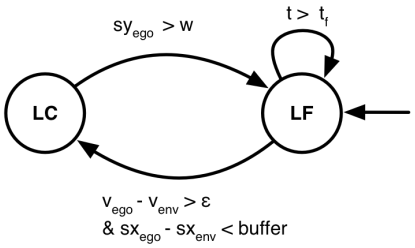

Figure 8: An automaton describing a simplistic behavior planner for lane changes

We call the above a bounded reachability problem. To solve this problem, APEX passes it to dReach [12], a reachability analysis tool for nonlinear hybrid systems. dReach answers the question: is there a trajectory of the vehicle starting in X_A that will violate the constraints? (e.g. will not reach the target set X_B or will get too close to another vehicle). dReach returns one of two answers. If the answer dReach returns is SAFE, then it is *guaranteed* that *no* behavior of the ego vehicle will violate the constraints. It should be stressed that this is a mathematical guarantee: no amount of simulation in this case will reveal a violation, because dReach guarantees that no such violation exists. If dReach answers δ-UNSAFE, then this means that there exists a behavior of the ego vehicle which, when perturbed by an amount $\delta > 0$, violates the constraints. See Fig. **??**. The parameter δ can be set by the user. It suffices to choose δ small enough so δ-SAT means the system is not robust since a small perturbation of size δ could cause it to violate the constraints.

6 Case Study

We briefly introduce and expand on the concept of driving scenarios to help reason about inherently diverse situations and requirements which an autonomous vehicle might face.

6.1 An unsafe lane change scenario

The following example describes a lane change scenario in the context of a mission and mobility goals. In this description we imply a valid local planning solution, and seek to verify that all possible individual trajectories which are selected in the *execution* of the plan are safe. First, in *Scenario 1* we will demonstrate a dangerous condition that could have been missed under testing or simulation. Next, in *Scenario 2* we will show how a refinement in the requirements on the perception system or a refinement in the behavioral controller can lead to a provably safe maneuver. Finally, in *Scenario 3* we demonstrate how a change in manufacturer specification can be accurately assessed for safety. To perform verification, we employ dReach version 3.15.10.02 on a Mac OSX laptop with Intel(R) Core i7(R) 2.60GHz CPU and 16 GB memory, and the results are provided in Table 5.

Scenario 1 (A simple lane change and goal) *As shown in Fig. 9, the ego vehicle is driving in the right lane of a uni-directional two lane road network. Another car is driving in front of the ego vehicle at a lower speed. We include the extreme case where the environmental vehicle stops. We highlight that when there is significant uncertainty regarding the ego vehicles orientation and that it may deviate (initially) from the reference trajectory (dashed line) while the tracking controller recovers. We note that the specification of the environment and the ego-vehicle in this scenario are defined as ξ and ϕ respectively.*

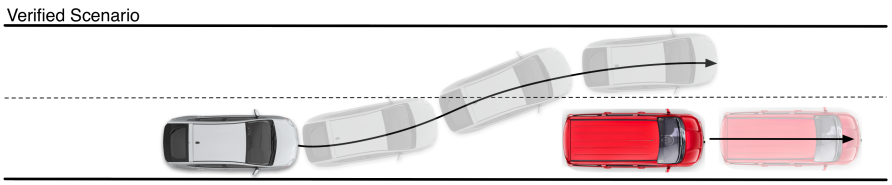

Figure 9: A lane change scenario that could have been missed in testing due to nonintuitive and uncountably infinite set of initial conditions. This scenario is unsafe for certain inter-vehicle buffer spacing and reachability analysis determines the minimum spacing to achieve a safe lance change.

6.1.1 Behavioral controller

We associate a behavioral controller $\mathcal{B}_1$ with Scenario 1. Figure 8 details the controller, where LC means "Lane Change" and LF means "Lane Follow". Table 5 records the parameters. It is a simple finite state deterministic automaton. We note, that this particular behavior controller is almost surely too simplistic to cover all of the scenarios faced by an actual vehicle, nevertheless it illustrates how we may formally represent a set of rules which instantiate certain behavior classes on an autonomous vehicle. Similar examples have been published by Darpa Urban Challenge participants []. Both controllers generated via reinforcement learning and reactive behavior controllers created via synthesis may be represented as deterministic finite automatons. As our current goal is to demonstrate that verification is possible, rather than the richness of the scenarios that the behavioral controller can handle, we find this controller suitable.

Given any deterministic finite automaton it is possible to express as a *computational logic tree*. Such a tree is rooted in a single state, is infinite in size, and represents a branching notion of time; that is each state (moment in time) may split into multiple possible future worlds. As we will explain in the following sections, such a representation is at the heart of the APEX approach and verification occurs over a bounded search depth on such a computation tree.

We present the initialization of the scenario and the results of the verification. Table 5 contains the initialization of each parameter.

6.1.2 Verification and Result

Finally, for the lane change case, we define an additional constraint set R_{unsafe} as well as a goal set representing the maximum allowable deviation from the goal state. R_{unsafe} expresses that the system fails if it still hasn't changed lanes within 2 sec or it collides with the car ahead of it.

$$((s_y < w) \wedge (t > 2)) \vee ((s_y < w) \wedge (sx - \epsilon > s_{x_{env}})) \tag{38}$$

Then, using APEX we *attempt* to show that there is no execution of the system which can enter R_{unsafe}. However, because the system is incorrectly designed dReach returns δ-UNSAFE.

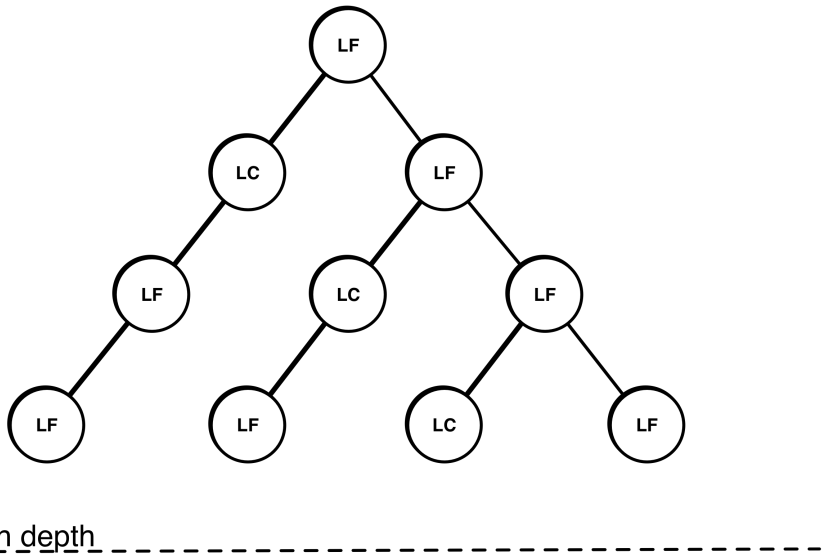

Figure 10: An automaton describing a simplistic behavior planner for Lane Following (LF) and Lane Changes (LC)

6.2 A safe lane change scenario

Using the information and counterexample from the previous scenario it is easy to see that the behavior controller must be corrected in order to guarantee safety of the lane change scenario.

Scenario 2 (A more conservative behavioral controller) *We begin with Scenario 1. In order to ensure the forward safety of the vehicle we propose a small modification to the behavioral controller of the vehicle, and furthermore require that the ego vehicle's localization system return estimates with less uncertainty. Namely, we first increase the size of variable buffer, so that the ego vehicle is forced to initiate a lane change maneuver earlier. Secondly, we decrease the size of the initial sets. Speed v now starts anywhere in* $[10.9, 11]$ *and* s_y *starts in [0.0,0.05].*

With these changes, dReal returns SAFE, meaning that no trajectory of the system violates the constraints.

Table 5: Verification Results

Symbol	Scenario 1	Scenario 2	Scenario 3
w	3.7	3.7	3.7
B	15	20	20
δ	0.1	0.1	0.1
v_{ego}	[10.8, 11.1]	[10.9, 11]	[10.9, 11]
$s_{x_{ego}}$	[0.0, 0.5]	[0.0, 0.5]	[0.0, 0.5]
$s_{y_{ego}}$	[0.0, 0.1]	[0.0, 0.05]	[0.0, 0.05]
Ψ	[0.0, 0.1]	[0.0, 0.1]	[0.0, 0.2]
Search Depth	2	2	2
Verification Time (s)	30.821	373.924	36.166
Result	δ-UNSAFE	SAFE	δ-UNSAFE

6.3 A supplier issues a specification change

Given that a safe controller has been found a supplier wishes to know if they may reduce the accuracy of several key sensors associated with localization of the ego vehicle. Such a specification change is known to add significant uncertainty to the estimate of the ego vehicle's heading angle during the planning phase.

Scenario 3 (Large perception errors) *We begin with Scenario 2. In order to reflect the change in supplier specification we update the localization system return estimates to reflect greater uncertainty. Namely, we increase the size of the intial set for ego vehicle heading such that* Ψ *starts in [0.0,0.2].*

The result of this modification is again δ-UNSAFE, because the ego vehicle clips the rear bumper of the environmental vehicle while executing the lane change maneuver. Again the engineer in charge of the project may use the new information to refine the controller design or reject the suppliers specification change. In this way formal verification efforts can be a useful tool in determining the *requirements* which sensors and perception systems must meet given a particular control algorithm.

Part IV

Composable Hybrid Agents

6.4 Motivation

Having successfully verified a lane change scenario using the APEX framework we shift our focus to scalability and usability. Scalability in this sense revolves around three key issues:

- As the complexity of scenarios grows (ie the number of agents increase) how can we ensure that verification instances are built quickly and accurately?
- As the realism of the controllers used in the scnearios is increased how can we create tractable representations of such systems?
- As the language describing the other agents behaviors becomes more expressive can naieve application of verification technology meet the challenge?

In order to begin to provide a structured answer to such questions, we look to VLSI (*very large scale integration*) methodologies for inspiration. VLSI design problems encompass area minimization, speed maximization, power dissipation minimization, design time minimization, and testability maximization. Design decisions made to improve one metric often have follow on results which may affect the other metrics which make up a VLSI cost function. The main concepts popularized within VLSI methodologies are abstraction and hierarchy. Utilizing abstraction we *hide lower level details* from the designer. *Hierarchy* allows the design's effectiveness and implementation to be viewed at different levels of abstraction.

In APEX we view one end user as an integrator who combines groups of perception, planning, and vehicle components into agents. In Section 3.1 we explored both the underlying algorithms and how some algorithms may be abstracted such that only their output is visible to the verification process. This type of system operator would be focused on bottom up construction of new vehicle or environmental agents.

At each step along the way a system operator may verify or simulate agent components in order to gain an intuitive understanding of their behaviors; however, often nothing formal can be said without moving to an alternate system view in which the agent is instantiated in an environment and checked against a specification. This system operator would be primarily concerned with top down construction of scenarios in order to investigate the interplay of the various components of the ego-vehicle agent.

In the preceding sections we presented a view of bottom up construction of an ego-vehicle agent. By verifying the agent in a limited scenario such as lane change on a straight road, we established certain expected, even intuitive, safety properties. However, the counterexamples we generated in the design process were rather simple and displayed monotonic relationships between manipulation of threshold variables and safety. Furthermore, composing more complex scenarios and the tree of verification instances quickly becomes an arduous task. We comment on a recommend work flow (design pattern) in Section **??**

7 Vehicle Model

7.1 Representing Local Planning within a Hybrid System

We use the vehicle model presented in Section 3.1 with some minor modifications, namely that δ is specified directly by a control law rather than an ODE.

$$\dot{\beta} = \left(\frac{C_r l_r - C_f l_f}{m v^2}\right)\dot{\psi} + \left(\frac{C_f}{mv}\right)\delta - \left(\frac{C_f + C_r}{mv}\right)\beta \tag{39}$$

$$\begin{aligned}\ddot{\psi} = \left(\frac{C_r l_r - C_f l_f}{I_z}\right)\beta - \left(\frac{C_f l_f^2 - C_r l_r^2}{I_z}\right)\left(\frac{\dot{\psi}}{v}\right) \\ + \left(\frac{C_f l_f}{I_z}\right)\delta\end{aligned} \tag{40}$$

$$\dot{v} = a_x \tag{41}$$

$$\dot{s_x} = v\cos\left(\beta + \psi\right) \tag{42}$$

$$\dot{s_y} = v\sin\left(\beta + \psi\right) \tag{43}$$

The dynamics of the vehicle remain the same, but we create a new controller that allows the vehicle to follow and switch between reference trajectories which describe aspects of the road geometry. The purpose of this modification is to increase the number of "motion primatives" at the vehicles disposal from two to a continuum of potential trajectories. The mechanism we use to achieve this goal is a geometric path tracking (and generating) method known as *pure pursuit*

7.1.1 Pure Pursuit

In this section we describe a popular and simple method from the class of geometric path tracking algorithms.

We provide a brief informal sketch of the algorithm, and then detail its relation to the inputs of the velocity and steering tracking controllers.

Algorithm:

- Update vehicle state
- Find nearest path point
- Find the goal point
- Transform goal to vehicle coordinates
- Calculate desired curvature
- Set steering to desired curvature
- Update position

While the pure pursuit algorithm is simple and effective, in practice it has the downside that the requested steering angles are not continuous. This results in unnatural steering which occupants may perceive to be jerky. We utilize this method because it provides a simple closed form solution which can be easily modeled in current hybrid systems verification tools. In Section **??** we discuss alternatives which may be tractable within the framework developed in these benchmarks.

We note that while it is possible to compute the appropriate derivatives to use the more complex controller presented earlier in this work, it is cumbersome, numerically unwise, and computationally intractable to use such relations in the verification process. Thus we begin an alternate derivation of a control law for the pure pursuit algorithm by noting a simple relationship describing steering angle as a function of the vehicle wheel base and the computed arc, R:

$$tan(\delta) = \frac{l_r + l_f}{R} \tag{44}$$

In Snider's implementation [17] the following relations are derived via the law of sines.

$$\frac{l_d}{sin(2\alpha)} = \frac{R}{sin(\frac{\pi}{2} - \alpha)} \tag{45}$$

$$\frac{l_d}{sin(2\alpha)} = \frac{R}{cos(\alpha)} \tag{46}$$

$$\frac{l_d}{sin(\alpha)} = 2R \tag{47}$$

Thus the curvature of the arc is:

$$\kappa = \frac{2sin(\alpha)}{l_d} \tag{48}$$

Thus the steering angle is:

$$\delta = tan^{-1}(\kappa(l_f + l_r)) \tag{49}$$

Finally, simplifying and adding a proportional gain k_{pp}

$$\delta(t) = tan^{-1}\left(\frac{2(l_f + l_r)sin(\alpha(t))}{k_{pp}l_d}\right) \tag{50}$$

Thus, the new vehicle controllers can be specified as follows:

$$\delta(t) = tan^{-1}\left(\frac{2(l_f + l_r)sin(\alpha(t))}{k_{pp}l_d}\right) \tag{51}$$

$$a_x(t) = k_6(v_d(t) - v(t)) \tag{52}$$

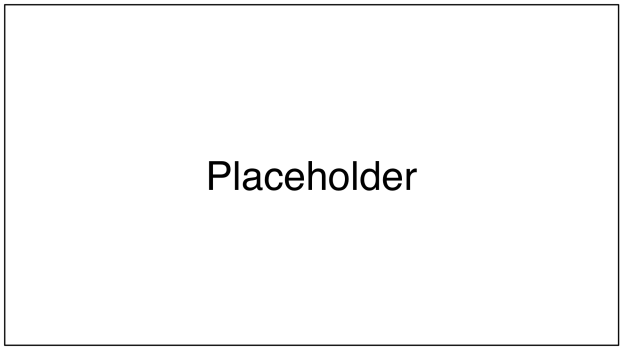

Figure 11: Our Pure Pursuit Figure

7.2 Hybrid Model of Ego-Vehicle

7.3 Traffic Participants

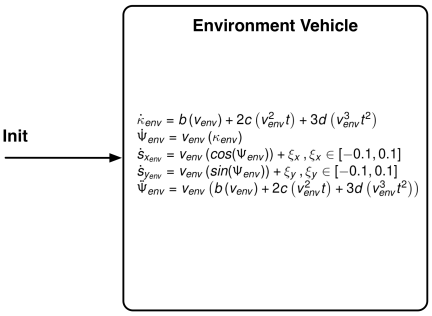

Figure 12: Environment Vehicle Automaton

The other vehicles operating within a scenario present both an interesting challenge and a primary motivation for formal verification. It is clear that it is impossible to know the intentions of the agents operating such vehicles; their execution represents a significant source of non-determinism. In fact, a more complex model of such agents which includes details such as steering angle or tire friction will not enable less conservative results, for it is the control input not the plant that remains the largest unknown. Thus, we conclude that: *for verifying the autonomous agent, only the perceptible behavior of other agents is important, not their internal structure.*

Still it remains clear that *the behavior of other agents must be part of the*

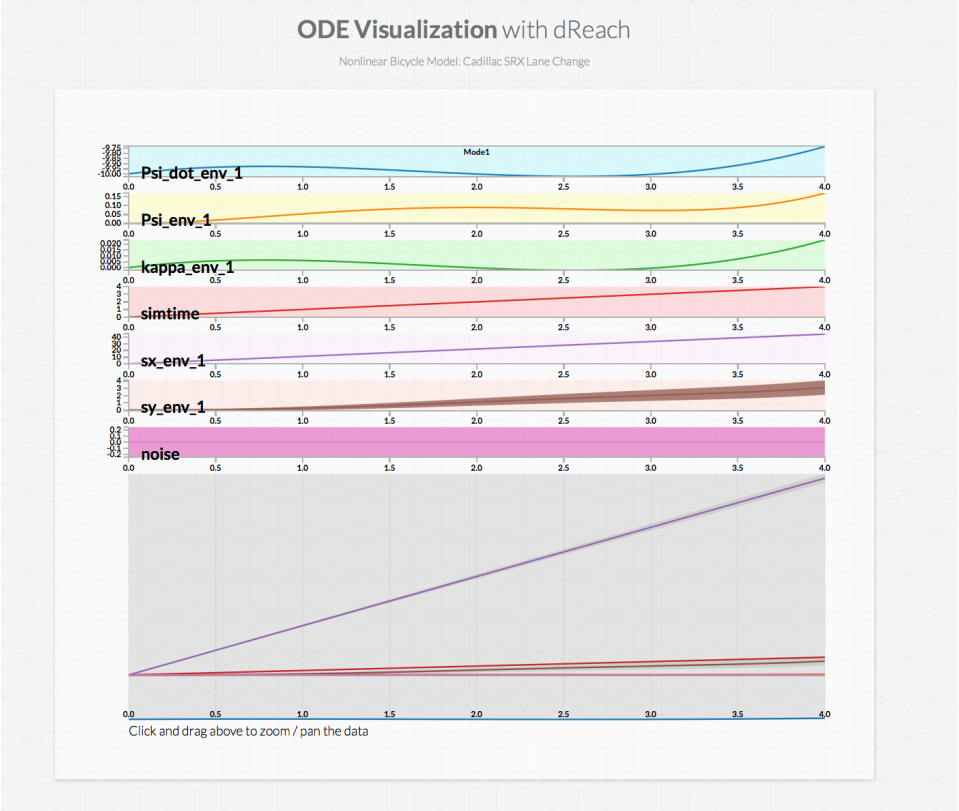

Figure 13: Environment Vehicle Behavior Visualization

scenario description. As such we present a safety case which assumes that other agents will follow a certain minimal set of driving rules.

For brevity we will reference the following specification as ξ in the case studies.

8 Case Study 2

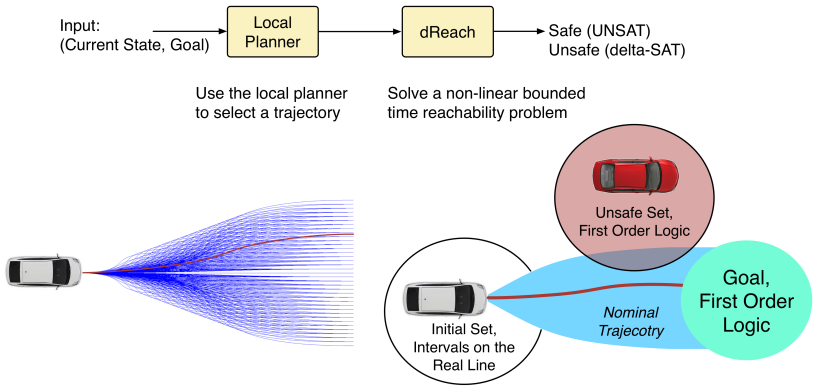

Figure 14: One step in the APEX tool: the local planner generates a trajectory, which is automatically input into the mission description file and verified using dReach

Part V

Implementation

9 APEX Tool

9.1 The APEX Approach

APEX answers the following question by re-formulating it as a reachability problem: given some initial uncertainty about the state of the AV, constraints on the configuration of the environment, and a desired behavior of the AV (like mobility goal and traffic laws), is it ever possible for the AV to violate the desired behavior? Fig. 14 summarizes a single execution of the verification engine. For each trajectory selected by the planner (highlighted in red), APEX calls dReach [8], a reachability analysis tool for nonlinear hybrid systems.

We emphasize that the verification process is *offline* - the vehicle does *not*

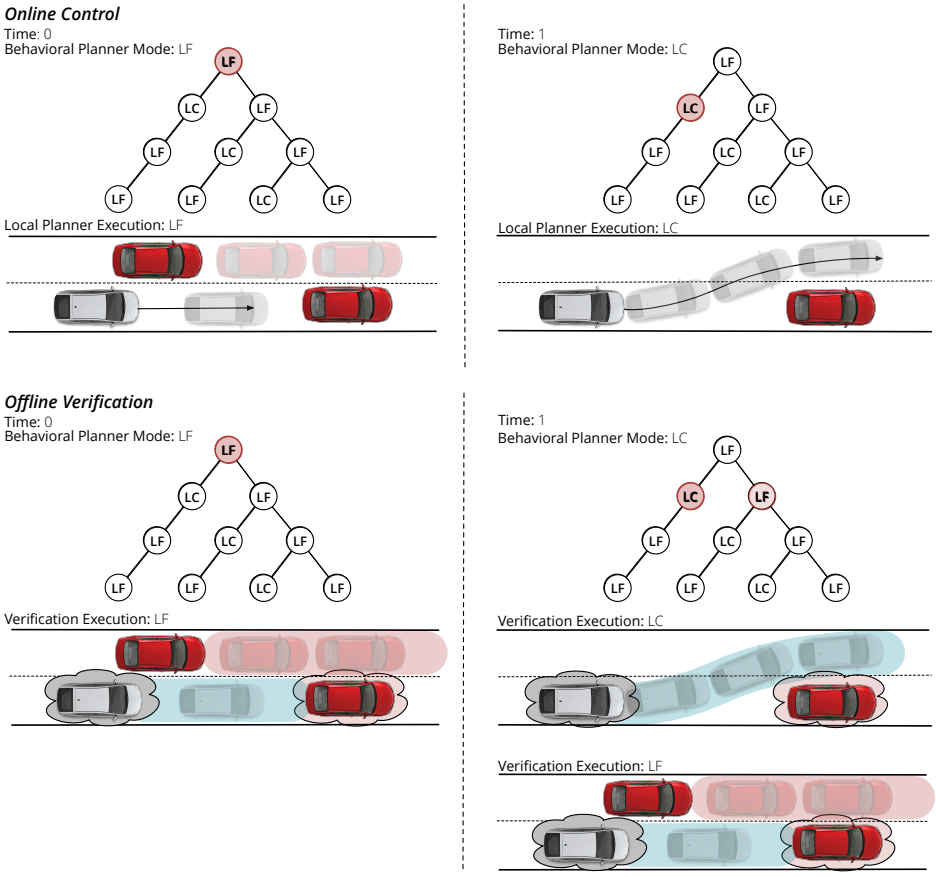

Figure 15: Stages of APEX verification and their correspondence to control execution. Top left: at $t = 0$, mode is Lane Follow (LF) and the vehicle follows the current lane. Top right: at $t = 1$, mode is Lane Change (LC) and vehicle starts a lane change maneuver. Bottom left: offline, APEX verifies that in mode LF, the vehicle can track the trajectory. Bottom right: offline, APEX verifies both possible executions, a lane change and a lane following.

run APEX while it is driving. At each decision point encountered by the behavioral planner there may be multiple executions of the verification engine depending on the design of the behavioral planner. Fig. 15 describes how the execution of the controller online relates to the *offline* verification process. In contrast to the simulation based approach outlined in the introduction, the result of the APEX approach is that we have converted a brute force search over real intervals into a finite series of tractable bounded reachability problems over a finite verification horizon.

9.2 Tool Input

APEX is a *command line tool* for verification of autonomous vehicle missions written in Python and C++. The input to the verification process is a *mission definition file*. The *mission definition file* defines the sequence of waypoints or road links which the vehicle will traverse in order to achieve a mobility goal.

The *mission defintion file* describes the following:

- *The collection of agents in the scenario*, consisting of the ego vehicle and other cars in the scenario. The agents are described via ODEs that describe the evolution of their state with time, and their behavioral planners, which give the next waypoints for each vehicle. All agents operate in an ontology specific to the mission, in this case the world model consists of a geometric description of a road network.
- *Set of initial states* for each state variable of every vehicle.
- *The constraints that the AV should satisfy*, such as traffic laws and the unsafe conditions that ego vehicle must avoid. These are described in MITL.
- *The goal of the ego vehicle*, also expressed in MITL [5].

The mission definition file is part of a *mission definition script*. The latter manages the execution of the behavioral planner and trajectory generator. Each (state, goal) pair that is encountered on the mission generates at least one trajectory which must be verified. The *mission definition script* automatically updates a *scenario verification instance*. The *scenario*

verification instance is a dReach (.drh) file which combines the results of the plan execution with the dynamical model of the vehicle and a low-level trajectory tracking controller. The *agent definition file* contains the dynamical model of the vehicle and the tracking controller is also written using the syntax of dReach, it may be manually edited in order to match mission specific vehicle models. We provide an example of the syntax of the composed *scenario verification instance* in Fig. 16.

Together, the constraints of the environment ξ and ego vehicle goal and constraints ϕ constitute the *specification* of the mission. The mission is a success if every execution of the system (i.e., every simulation) satisfies the specification.

9.3 Tool Output

Each *scenario verification instance* can return either SAFE or δ-UNSAFE. SAFE means that for all possible executions of the system we can not reach an unsafe state. δ-UNSAFE means that there exists an execution of the system which comes within a δ of the unsafe region, and possibly enters it. If the system is δ-UNSAFE the tool will return a counter-example describing a tube around a concrete trajectory whose intersection with the unsafe region is not empty. Users of the APEX tool should be aware that selecting too large of a precision value (δ) may result in δ-UNSAFE results which are false positives, but any declaration of SAFE is guaranteed to be correct.

10 Simulation and GUI

```
// Each verification instance is automatically updated to
// to reflect the new planned trajectory.
#define s (22.681867)
#define kappa_0 (0.000000)
#define kappa_1 (0.033194)
// ...

// Below are constants regarding the vehicle and controller.
// These do not vary between trajectories, manually edited
#define m 2273.0
// ...

// Below are automatically generated parameters, helper
// definitions, etc.
#define w 3.7

// Cubic spline inputs
#define a (kappa_0)
// ...

// ------------------------------------------------------------

// Below are the automatically generated vehicle dynamics
// equations

mode 1;

invt:
// X position must be greater than or equal to 0
(sx>=0);
// Velocity must be greater than or equal to 0
(v>=0);

flow:
/* Target Vehicle Dynamics*/
d/dt[tau]=1;
// Compute rate of change of the slip angle
d/dt[Beta]=((cl/(m*v*v))-1.0)*Psi_dot + Cf*delta/(m*v) - (Cf+Cr)
    *Beta/(m*v);
//...

// Controller Equations -- Generated automatically
d/dt[kappa_d] = b*v_d + 2*c*v_d*v_d*tau + 3*e*v_d*v_d*v_d*tau*
    tau ;
// ...

jump:

// ------------------------------------------------------------

// Set of Initial Conditions
init:
@1 (and (sx>=0) (sx<=0.5) (sy>=0) (sy<=0.1) (v>=10.8) (v<=11.1)
// ...
// Set of Goal Conditions
goal:
@1 (or (and (sy<w) (tau>2) (tau<3)) (and (sy<w) (sx>sx_env)));
```


Figure 16: Scenario verification instance generated by APEX

Part VI

Conclusions

10.1 Future Work: verification, learning, ethics, and control

The likely arrival of learned behaviors and the accompanying cost functions which allow the discrimination between multiple feasible strategies will necessitate careful consideration regarding the quantification of desireable robot behaviors. It is easy to personify the *software agent* which operates an AV; naturally, this leads to the consideration of such an agent's ethical duties. It is not clear whether such software can truly exhibit ethical behavior or rather simply mimic the instructions of the designer. Nevertheless, "it seems certain that other road users and society will interpret the actions of automated vehicles and the priorities placed by their programmers through an ethical lens" [**?**]. In order to improve the safety, efficacy, and percieved morals of AVs we propose 3 areas of promising future research:

- Automatic verification of learned behaviors and cost functions
- Hierarchical property satisfaction and instantiation of temporal logic constraints in a model predictive control framework
- Online verification of behavioral plans [**?**] over a range of 5-10 seconds

We have already argued that verification of learned behaviors and cost functions is necessary; we propose that such activities should be fully *automated* in manner similar to static verification of source code. Before an update is released it will be subjected to an ever increasing battery of common verification scenarios. Secondly, we intuit that describing vehicle actions as ethical implies that there exists a ranking function over potential behaviors; if vehicle specifications are described hierarchically we can actually dictate and examine the ethics of a particular AV. For example, it may be desirable that only in a near crash situation the AV disregards the speed limit and lane keeping behaviors in order to avoid an accident. Finally, we propose that such *offline* verification techniques in APEX be reimagined and refactored for short-horizon online verification of

all potential vehicle actions. It is likely that initial forays into autonomy may require handoffs between human and machine. Studies show [6] that a safe handoff requires 5-8 seconds of preparation. Thus, online verification techniques may be used to discover potential system failures and provide fair warning to the driver.

References

[1] Volvo press release: US urged to establish nationwide federal guidelines for autonomous driving. 2015.

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

[3] Matthias Althoff and John 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.

[4] Rajeev Alur, Tomas Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. In *Symposium on Principles of Distributed Computing*, pages 139–152, 1991.

[5] Rajeev Alur, Tomás Feder, and Thomas A Henzinger. The benefits of relaxing punctuality. *Journal of the ACM (JACM)*, 43(1):116–146, 1996.

[6] Myra Blanco, Jon Atwood, Holland M Vasquez, Tammy E Trimble, Vikki L Fitchett, Josh Radlbeck, Gregory M Fitch, Sheldon M Russell, Charles A Green, Brian Cullinane, et al. Human factors evaluation of level 2 and level 3 automated driving concepts. *Virgina Tech, Automated Vehicle Systems Group Research*, 2013.

[7] Luke Fletcher, Seth Teller, Edwin Olson, David Moore, Yoshiaki Kuwata, Jonathan How, John Leonard, Isaac Miller, Mark Campbell, Dan Huttenlocher, et al. The mit–cornell collision and why it happened. *Journal of Field Robotics*, 25(10):775–807, 2008.

[8] Sicun Gao, Soonho Kong, Wei Chen, and Edmund Clarke. Delta-complete analysis for bounded reachability of hybrid systems. *arXiv preprint arXiv:1404.7171*, 2014.

[9] Sicun Gao, Soonho Kong, and Edmund M Clarke. Satisfiability modulo odes. In *Formal Methods in Computer-Aided Design (FMCAD), 2013*, pages 105–112. IEEE, 2013.

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

[11] Shinpei Kato, Eijiro Takeuchi, Yoshio Ishiguro, Yoshiki Ninomiya, Kazuya Takeda, and Tsuyoshi Hamada. An open approach to autonomous vehicles. *Micro, IEEE*, 35(6):60–68, Nov 2015.

[12] Soonho Kong, Sicun Gao, Wei Chen, and Edmund 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] Matthew McNaughton. *Parallel Algorithms for Real-time Motion Planning*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2011.

[14] Bryan Nagy and Alonzo Kelly. Trajectory generation for car-like robots using cubic curvature polynomials. *Field and Service Robots*, 11, 2001.

[15] Petter Nilsson, Omar Hussien, Ayca Balkan, Yuxiao Chen, Aaron D Ames, Jessy W Grizzle, Necmiye Ozay, Huei Peng, and Paulo Tabuada. Correct-by-construction adaptive cruise control: Two approaches.

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

[17] Jarrod M Snider. Automatic steering methods for autonomous automobile path tracking. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RITR-09-08*, 2009.

[18] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[19] M. Mitchell Waldrop. Autonomous vehicles: No drivers required. *Nature*, 518(7537):20–23, feb 2015.

[20] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon control for temporal logic specifications. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '10, pages 101–110, New York, NY, USA, 2010. ACM.