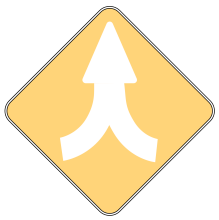

Autonomous Vehicle Plan Execution and Verification: 2015-2016 Report

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

February 19, 2016

Executive Summary

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 [12]. Estimates indicate that more than 90 percent of all accidents are due to driver error [12]. 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 [15]. 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 [6]; 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 [11] or (2) force the human occupants of an AV to make all critical decisions and shift liability to the pilot [7]. 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 [8] 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 [3]. One problem highlighted by the first ever crash between AVs at the Urban Challenge [4] 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" [10]. 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

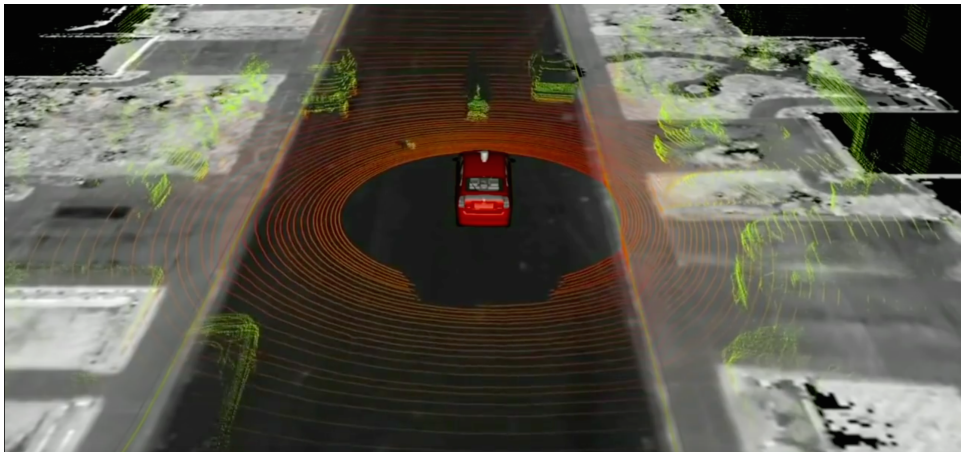

Figure 1: New Challenges: A woman in a wheelchair chasing a duck; autonomous vehicles are driving in real traffic engineers face an extremely long tail of events for which they must provide solutions

the safety of autonomous driving is paramount to putting self-driving cars in the hands of the public [10].

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 [13, 9, 12]. 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.

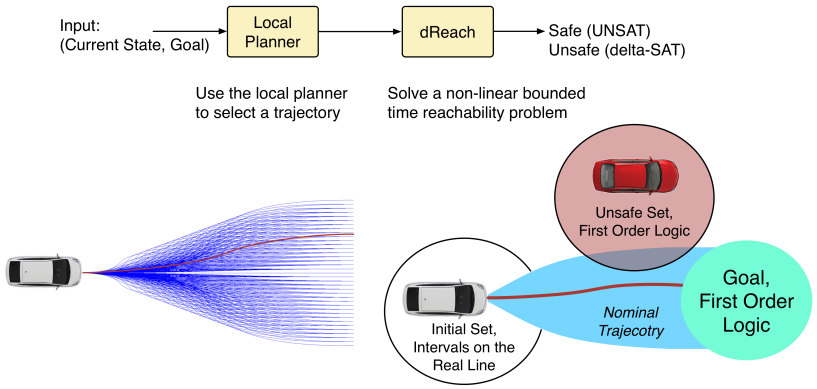

Figure 2: A single offline execution of the APEX tool calls the local planner associated with the AV in order to generate a reachability problem

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 [5] 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 [**?**].
- 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 [**?**].

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 [4]. 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

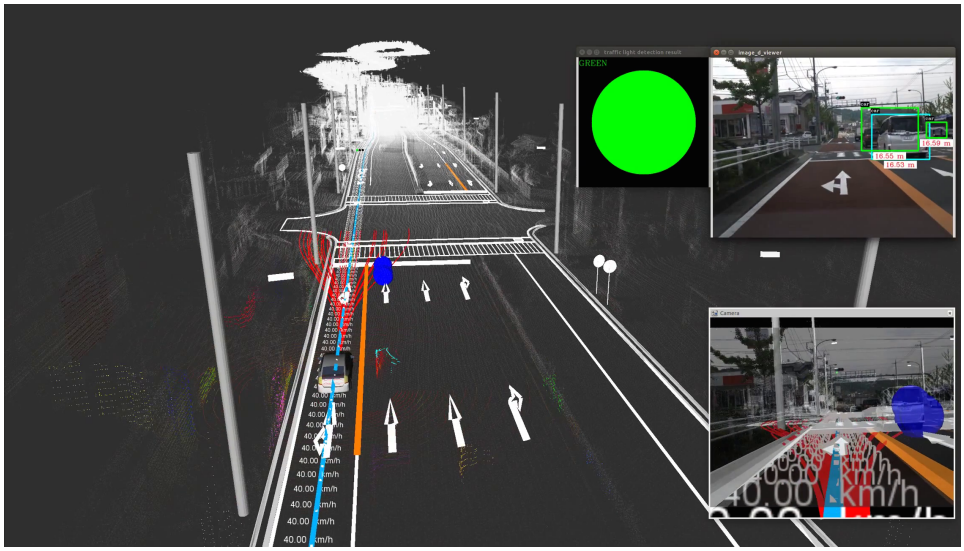

Figure 3: ROS APEX planning implementation GUI.

environment, much less a dynamic one" [10]. 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 [**?**], but such methods cannot currently scale to realistic systems and are extremely conservative. In response the authors of [**?**] 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) [**?**]. Finally, some research which eschews discretization in favor of continuous linearized dynamics focuses on moving the verification task online [1].

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 [**?**] 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.

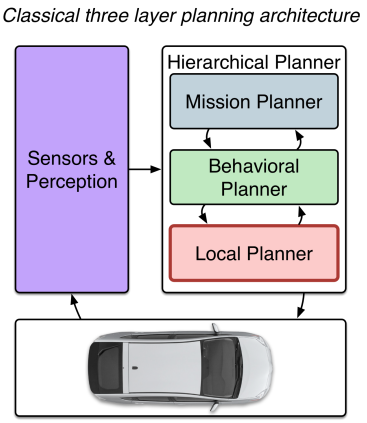

Figure 4: The three layer architecture presented by Gat is widely accepted as a standard means of implementing planning and control for an autonomous vehicle.

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 [**?**] 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 [**?**] are:

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

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

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

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

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

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

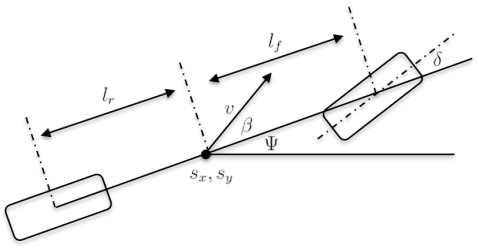

Figure 5: Nonlinear bicycle model describing the statespace for the APEX approach to vehicle dynamics

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 [**?**]. 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 [**?**]. 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 [**?**]

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 [**?**] are presented in Table 3.

Table 3: Controller Parameters [**?**]

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 [**?**] and [**?**] 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 [**?**] 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

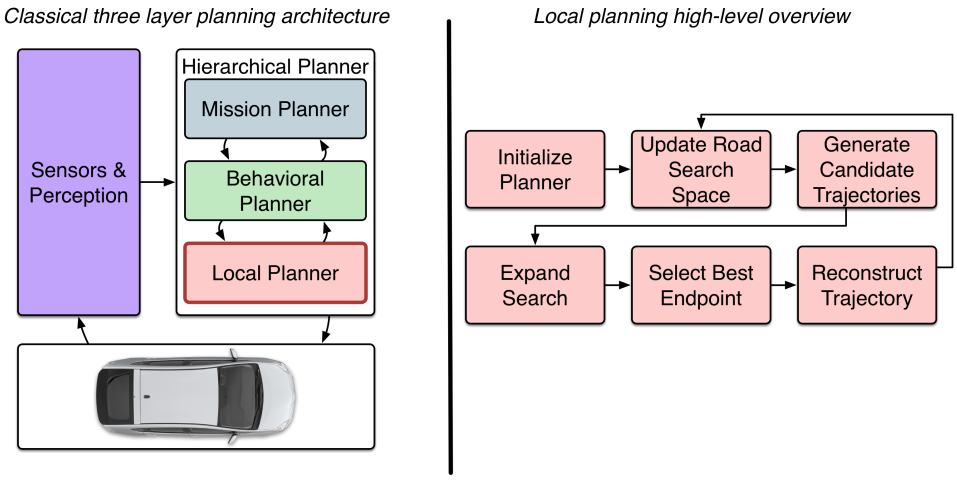

Figure 6: Details of a local planning algorithm and used by AVs employing state lattice planning

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 [**?**] such that it is compatible with a stable parameter formulation presented by McNaughton [**?**]. 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

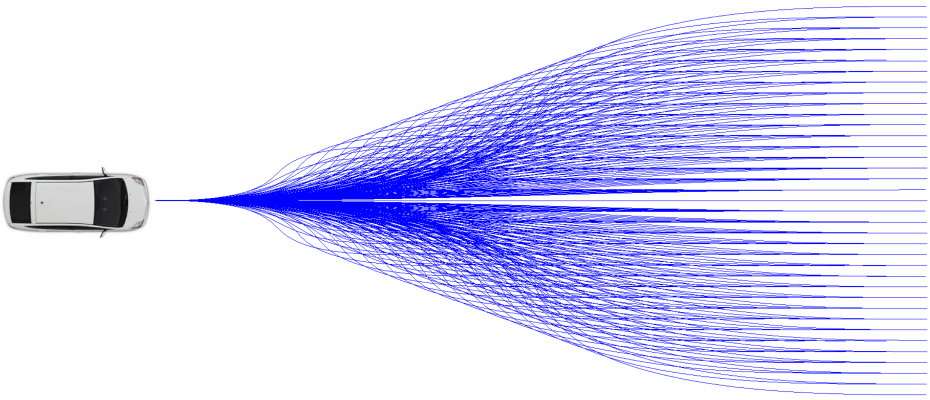

Figure 7: Output of an execution (10 Hz) of the trajectory generator, a single trajectory will be chosen from this set.

aspects of the planner may be found in [**?**] 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

Tree Based Verification

4 Case Study 1: SAE 2016

Part IV

Composable Hybrid Agents

5 Case Study 2

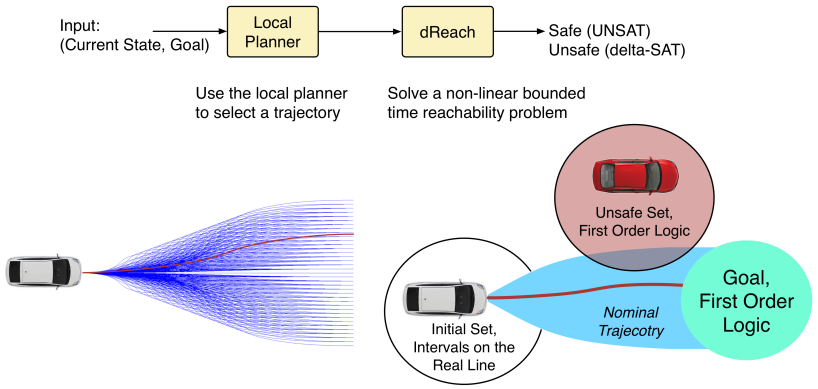

Figure 8: One step in the APEX tool: the local planner generates a trajectory, which is automatically input into the mission description file and verified using dReach

Part V

Implementation

6 APEX Tool

6.1 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. 8 summarizes a single execution of the verification engine. For each trajectory selected by the planner (highlighted in red), APEX calls dReach [**?**], a reachability analysis tool for nonlinear hybrid systems.

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

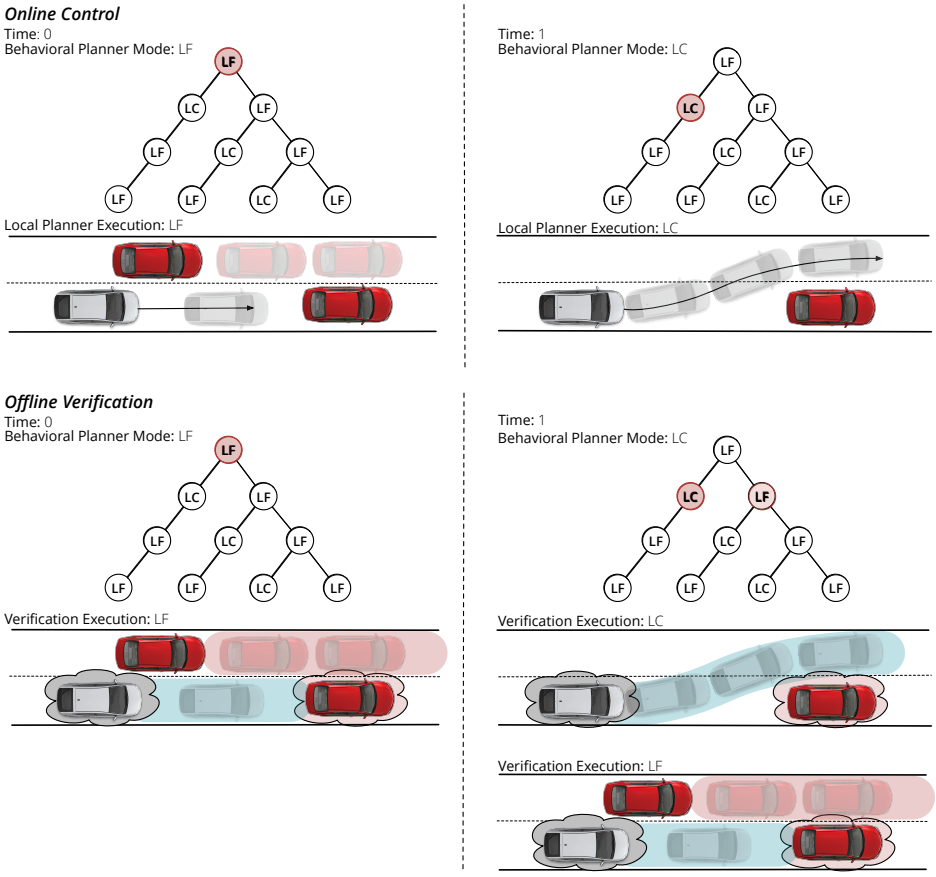

Figure 9: Stages of APEX verification and their correspondence to control execution. Top left: at $t = 0$, mode is Lane Follow (LF) and the vehicle follows the current lane. Top right: at $t = 1$, mode is Lane Change (LC) and vehicle starts a lane change maneuver. Bottom left: offline, APEX verifies 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. 9 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.

6.2 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 [**?**].

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. 10.

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.

6.3 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.

7 Simulation and GUI

Part VI

Conclusions

7.1 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" [7]. 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 [14] 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 [2] 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] 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.

[2] 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.

[3] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA Urban Challenge: Autonomous vehicles in city traffic*, volume 56. springer, 2009.

[4] 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.

[5] 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.

[6] Corinne Iozzio. Street-legal robots. *Sci Am*, 311(2):20–20, jul 2014.

[7] Markus Maurer, J Christian Gerdes, Barbara Lenz, and Hermann Winner. *Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte*. Springer-Verlag, 2015.

[8] Stuart Russell, Daniel Dewey, Max Tegmar, Anthony Aguirre, Erik Brynjolfsson, Ryan Calo, Tom Dietterich, Dileep George, Bill Hibbard, Demis Hassabis, et al. Research priorities for robust and beneficial artificial intelligence. 2015.

[9] David Silver. *Learning Preference Models for Autonomous Mobile Robots in Complex Domains*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, December 2010.

[10] 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.

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

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

[13] Junqing Wei, John M Dolan, and Bakhtiar Litkouhi. Autonomous vehicle social behavior for highway entrance ramp management. In *Intelligent Vehicles Symposium (IV), 2013 IEEE*, pages 201–207. IEEE, 2013.

[14] Junqing Wei, Jarrod M Snider, Tianyu Gu, John M Dolan, and Bakhtiar Litkouhi. A behavioral planning framework for autonomous driving. In *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, pages 458–464. IEEE, 2014.

[15] Daniel Weld and Oren Etzioni. The first law of robotics (a call to arms). In *AAAI*, volume 94, pages 1042–1047, 1994.

```
// 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 10: Scenario verification instance generated by APEX