A Driver's License for Autonomous Vehicles: Composable Hybrid Agents and Benchmark Scenarios

Matthew O'Kelly[1], Houssam Abbas[1], Soonho Kong[2], Aditya Pinapala[1], and Rahul Mangharam[1]

[1] University of Pennsylvania, Philadelphia, PA, U.S.A.
mokelly@seas.upenn.edu, habbas@seas.upenn.edu, pinapala@seas.upenn.edu
rahulm@seas.upenn.edu
[2] Carnegie Mellon University Pittsburgh, PA, U.S.A.
soonhok@cs.cmu.edu

Abstract

To do...

1 Outline

- Autonomous vehicles are awesome blurb
- Big challenge in verifying the vehicle's operation between the levels of behavioral planning and trajectory tracking, inclusive.
- Source of challenge: variations in physical environment (road networks and regulations), variations in other vehicles (number and behavior), errors of ego vehicle state estimation, imperfect plan execution.
- We present three benchmarks, which we call scenarios, and preliminary experience in running dReach to verify them.
- Why reachability rather than, say, stochastic simulation?
- Ego vehicle model: given a reference trajectory, bicycle model ODE. Environment mode contains reference trajectories for the center of the lane.
- Reference trajectory is generated by pure pursuit.
- Target point for pure pursuit is generated by hybrid automaton behavioral planner.
- Other vehicle models
- Scenario 1: 2 cars on straight road, lane changing. Describe road network, agents, properties to be satisfied, one dReach run on it. dReach run results include time bound on verification, tool runtime, number of jumps, number of time steps/jump (?)
- Scenario 2: 3 cars on curved road. Describe same as above. Then describe 3 successively more complex configurations of that same scenario, but in less detail.
- Scenario 3: General description of traffic light. Highlight the need for composable agent models. Can no longer hand design verification instance.
- Extensions of benchmarks: what can be added to the models to make them more realistic?

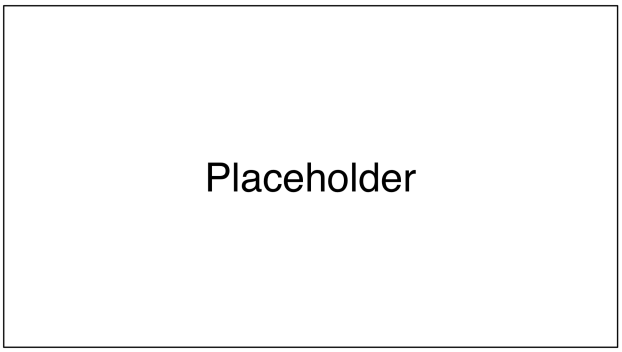

Figure 1: A grid of canonical driving scenarios which a human driver might encounter in a certification procedure, highlighted, scenarios covered in this benchmark set

2 Introduction

- NHTSB says AV software is considered a driver. How does one bound the risk posed to society by software agent.
- 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 infitesimal portion of the state space, need to gain confidence that algorithms are sound.
- Verification Methods and Tools
  - Industry Perspective: Testing and Simulation
  - Control Perspective: Lyapunov Functions
  - Software Perspective: Model Checking
  - Logic Perspective: Theorem Proving

CPS perspective is 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. We are interested in developing the appropriate formal models, a set of specifications, and a battery of scenarios. The community needs to investigate how such methods scale, or fail to scale, and provide new solutions.

3 Models

3.1 Vehicle

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. 3. The input to such a model 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

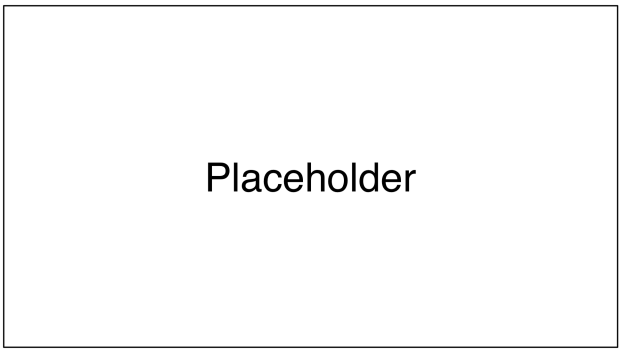

Figure 2: Abstraction of Autonomous Driving, APEX preserves... APEX hides...

[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:

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

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

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

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

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

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

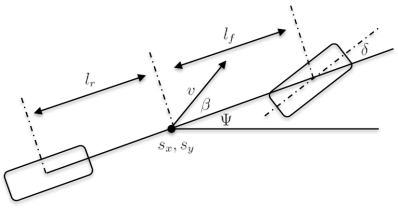

Figure 3: Nonlinear bicycle model describing the statespace for the APEX approach to vehicle dynamics

3.1.1 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 1 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 1: 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

3.1.2 Pure Pursuit

The simplest algorithm for path tracking and trajectory generation... From a type perspective what is the difference between arc and spline? Geometrically, the arc satisfies **convexity**.

- Given a current position at the vehicles rear differential and a goal...
- The algorithm computes a constant curvature arc between the current position and the goal.
- The vehicle may then actuate its steering system such that the curvature of the arc is tracked.
- Downside is discontinuity between curvatures when the algorithm is iterated
- Is also reliant on waypoints, does not generate alternate maneuvers unless explicitly told to switch goals.

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

$$x^2 + y^2 = l^2 \tag{8}$$

$$x + d = r \tag{9}$$

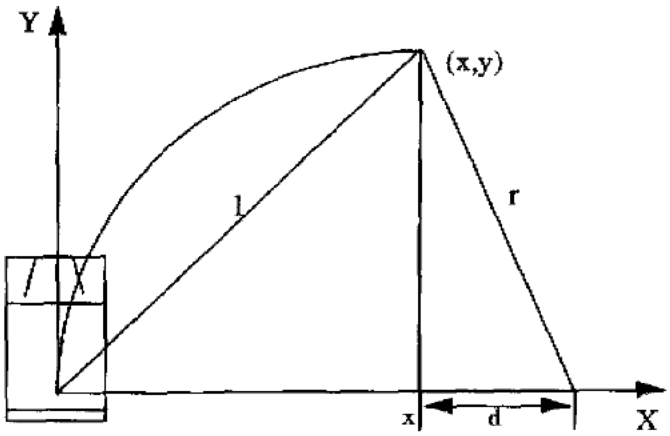

Figure 1. Geometry of the Algorithm.

Figure 4: Geometry of Pure Pursuit Algorithm

Now for the curvature such that:

$$r = \frac{l^2}{2x} \tag{10}$$

$$\gamma = \frac{2x}{l^2} \tag{11}$$

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 2.

Table 2: 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.

$$v_w = k_1(cos(\Psi_d)(s_{y,d} - s_y - w_y) - sin(\Psi_d)(s_{x,d} - s_x - w_x)) + k_2(\Psi_d - \Psi - w_\Psi) + k_3(\dot{\Psi}_d - \dot{\Psi} - w_\psi) - k_4(\delta - w_\delta) \quad (12)$$

$$a_x = k_5(cos(\Psi_d)(s_{x,d} - s_x - w_x) + sin(\Psi_d)(s_{y,d} - s_y - w_y)) + k_6(v_d - v - w_v) \quad (13)$$

Given that the ego-vehicle will be following constant curvature arcs defined by the pure-pursuit method. We must compute $\left\{\Psi_d, \dot{\Psi}_d, s_{x_d}, s_{y_d}, \right\}$.
We begin by computing the initial orientation of the vehicle:

$$\Psi_d(0) = \frac{\pi}{2} - \theta(0) \quad (14)$$

We note $\theta(0)$ can be computed via this simple relation:

$$sin(\theta(0)) = \frac{s_y - c_y}{r} \quad (15)$$

This implies that:

$$\theta(0) = sin^{-1}\left(\frac{s_y - c_y}{r}\right) \quad (16)$$

Then we compute

$$\Psi_d(t) = \frac{\pi}{2} - \theta(t) \quad (17)$$

For an arc subtended by the angle $\theta(t)$, the length of the arc, ρ is:

$$\rho = \theta(t) \cdot r \quad (18)$$

Alternately one might compute the length of the arc by integrating the velocity as directed along the arc such that $\rho(t) = \int_0^t v(\alpha)d\alpha$. Now we let $v(\alpha) = v_d$ and hold v_d to be constant. Thus:

$$\rho(t) = v_d \cdot t \quad (19)$$

Which implies that:

$$\Psi_d(t) = \frac{\pi}{2} - \left(\theta(0) + \frac{v_d \cdot t}{r}\right) \quad (20)$$

Differentiating we conclude that:

$$\dot{\Psi}_d = -\frac{v_d}{r} \quad (21)$$

Thus we can derive the remaining equations for a point mass moving in a plane:

$$\dot{\Psi}_d = -v_d/r \quad (22)$$

$$\dot{s}_{x_d} = v_d\,(cos(\Psi_d)) \quad (23)$$

$$\dot{s}_{y_d} = v_d\,(sin(\Psi_d)) \quad (24)$$

$$\ddot{\Psi}_d = 0 \quad (25)$$

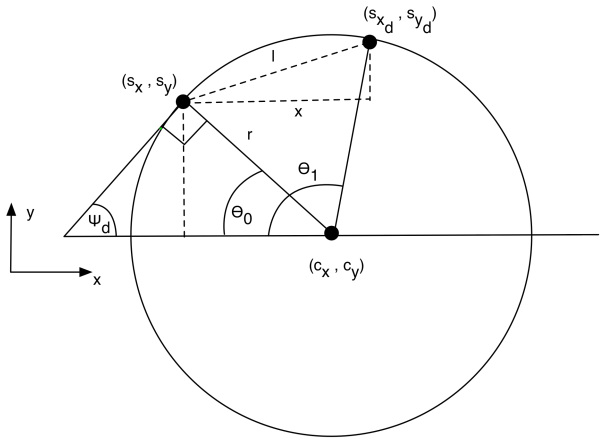

Figure 5: Geometric Description of Ego-Vehicle Trajectory Geometry

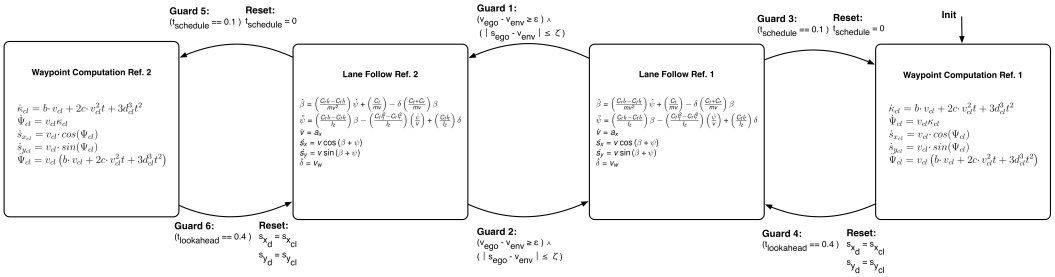

Figure 6: The behavioral controller governing the lane selection of the ego-vehicle

3.1.4 Behavioral Controller

Define the behavioral controller as an FSM. Can come from anywhere. Example we show is Fig. 6.

3.2 Road

High level view of the road is a network of connected segments. Graph describes reachability of one segment from another. Each segment need not be straight etc. We will choose to define segments via a collection of boundary points.

3.2.1 Boundaries

Each segment must have a left and right boundary. The boundaries are defined by an ordered sequence of points. We interpolate linearly between points.

3.2.2 Computing Centerline Trajectories

Each centerline is defined by an ordered sequence of points. In order to come to a continuous representation of the trajectory we interpolate between the points using Cubic splines. This ensures that the centerline trajectory is continuous. We may introduce offset trajectories from the centerline, but do not do so in this work. Each execution of the centerline computation we require as an input the current state and a goal state as defined by the ordered sequence of waypoints. This computation is done outside of the verification instance and supplied as part of the environment data. It is clear that this computation can be automated given an annotation of a map supplied by OpenStreetMaps etc.

The "vehicle" model, a point mass, represents the kinematic constraints which we wish to impose on such an idealized trajectory. We note that we will call the vehicle state x_{cl} because it does not necessarily have to be the same as the model used for verification (although it can be). The centerline represents an ideal trajectory, therefore, lower order models are often substituted here which do not account for control inputs as we are not actually executing a vehicle. In this implementation we define x_{sl} as:

$$x_{sl} = (s_x, s_y, v, \Psi, \kappa) \tag{26}$$

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

$$\dot{x} = v * cos(\Psi) \tag{27}$$

$$\dot{y} = v * sin(\Psi) \tag{28}$$

$$\dot{\theta} = \kappa * v \tag{29}$$

$$\dot{\kappa} = \frac{\dot{\Psi}}{L} \tag{30}$$

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:

$$x_p = (s_x, s_y, \Psi) \tag{31}$$

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:

$$\kappa(s) = \kappa_0 + a\kappa_1 s + b\kappa_2 s^2 + c\kappa_3 s^3 \tag{32}$$

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:

$$\kappa(0) = p_0 \tag{33}$$
$$\kappa(s_f/3) = p_1 \tag{34}$$
$$\kappa(2s_f/3) = p_2 \tag{35}$$
$$\kappa(s_f) = p_3 \tag{36}$$

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

$$a(p) = p_0 \tag{37}$$
$$b(p) = -\frac{11p_0 - 18p_1 + 9p_2 - 2p_3}{2s_f} \tag{38}$$
$$c(p) = \frac{9 * (2p_0 - 5p_1 + 4p_2 - p_3)}{2s_f^2} \tag{39}$$
$$d(p) = -\frac{9(p_0 - 3p_1 + 3p_2 - p_3)}{2s_f^3} \tag{40}$$

Which results in the following initialization heuristic:

$$p_0 = \kappa_0 = \kappa_i \tag{41}$$
$$p_1 = \kappa_1 = \frac{1}{49}(8b(s_f - s_i) - 26\kappa_0 - \kappa_3) \tag{42}$$
$$p_2 = \kappa_2 = \frac{1}{4}(\kappa_3 - 2\kappa_0 + 5\kappa_1) \tag{43}$$
$$p_3 = \kappa_3 = \kappa_f \tag{44}$$

3.2.3 Computing Goal Points

Given a parameterized trajectory, it is only possible to compute the evolution of the vehicle's planning state vector via forward simulation of the point-mass dynamics...

$$\dot{\kappa}_{cl} = b\left(v_{cl}\right) + 2c\left(v_{cl}^2 t\right) + 3d\left(v_{cl}^3 t^2\right) \tag{45}$$
$$\dot{\Psi}_{cl} = v_{cl}\left(\kappa_{cl}\right) \tag{46}$$
$$\dot{s}_{x_{cl}} = v_{cl}\left(cos(\Psi_{cl})\right) \tag{47}$$
$$\dot{s}_{y_{cl}} = v_{cl}\left(sin(\Psi_{cl})\right) \tag{48}$$
$$\ddot{\Psi}_{cl} = v_{cl}\left(b\left(v_{cl}\right) + 2c\left(v_{cl}^2 t\right) + 3d\left(v_{cl}^3 t^2\right)\right) \tag{49}$$

We define a lookahead time t_{la} and forward simulate motion along the centerline trajectory through the dynamics defined above. Having, completed such an operation the next waypoint for the pure pursuit algorithm is simply:

$$(s_{x_{cl}}(t_{la}), s_{y_{cl}}(t_{la})) \tag{50}$$

3.3 Other Vehicles

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

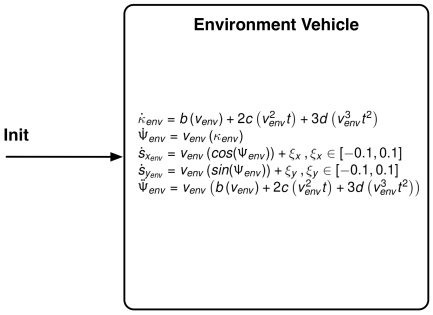

Figure 7: Environment Vehicle Automaton

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.

$$\Box\left(v_{env} \geq v_{max} \rightarrow a = 0\right) \tag{51}$$

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

$$\Box\left(v_{env} \geq 0\right) \tag{52}$$

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

$$\Box\left(a_{env} \leq a_{max}\right) \tag{53}$$

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

$$\Box\left(\neg LC \rightarrow \left(y_{min} \leq s_{y_{env}}\right) \wedge \left(y_{max} \geq s_{y_{env}}\right)\right) \tag{54}$$

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

$$\Box\left(LO \rightarrow \neg LC\right) \tag{55}$$

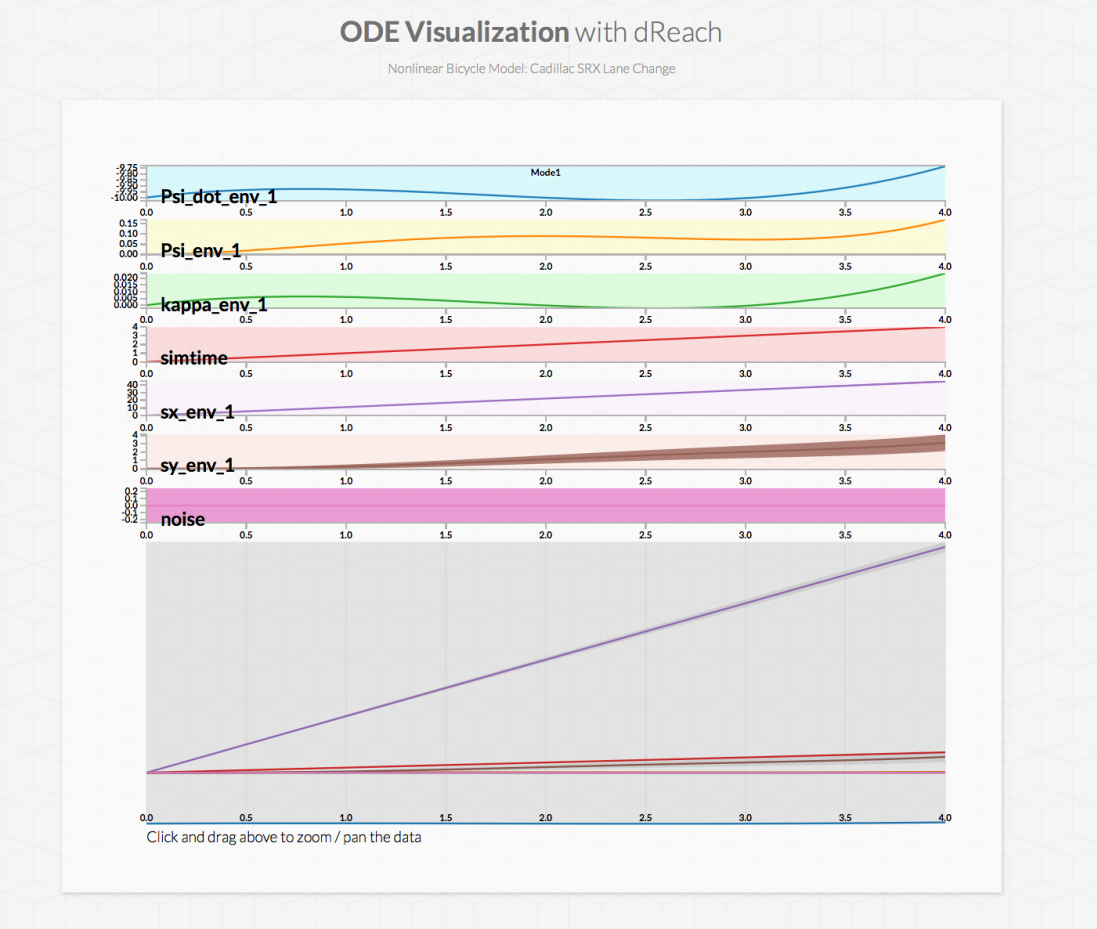

Figure 8: Environment Vehicle Automaton

4 Hybrid Model: Composition of Agents

4.1 Ego-Vehicle

4.2 Other Vehicles

4.3 Composition

4.4 General Complexity

5 Benchmarks

5.1 A Simple Lane Change

- We first present this scenario in [?]
- Other authors [?] have considered with static obstacles

- Counterexamples are somewhat obvious
- Inductive invariance argument allows verification of infinite time properties
- Most examples involving forward safety are monotonic ie selecting a lower operating speed implies an decrease in stopping time and improves the forward safety of the vehicle.
- In such problems there are no "holes" in the interval.
- Resulting proofs seem to indicate that testing would be sufficient to find counterexamples because unsafe results occur at extreme points in the state space only.
  - Verification still provides a guarantee of safety that was not previously available.
  - Verification is still useful for formally verifying specifications and interaction between supplier systems and vehicle dynamics.
  - Helps to search for viable combinations of specifications
- We extend this scenario with differential inclusions.
- Problem: we are forced into the tree representation because we cannot bring the trajectory generator in the dReach framework, requires external libraries and iteration. Could be remedied via creation of a lookup table or Neural Network. Alternative is to change the trajectory generation strategy such as Pure Pursuit.

5.2 Algorithm

How we establish safety of a decision controller. Proofs and lemmas: **For an appendix?**

Algorithm 1 Check Trajectory Sequence

checkSequence (searchDepth)
bool $safetyVar := TRUE$
int $depth := 0$
while $depth \leq searchDepth$ **do**
  float* $trajectoryParam = getNextTrajectory(depth)$
  $safetyVar := checkTrajectory(trajectoryParam)$

end while
return $safetyVar$

Theorem 1. *Proof of compositional nature of safety of sequence of trajectories*

Theorem 2. *Contract for parallel composition of verification instances*

Theorem 3. *Inductive invariance for search termination at a specified depth.*

5.3 Variations in Road Geometry

- In order to pursue more complex scenarios with more expressive agents we propose a variation in the trajectory generation strategy which admits a closed form
- Implement pure pursuit trajectory generation strategy...
- Open question if we can guarantee infinite time properties in general case, or on road by road basis via inductive invariant.

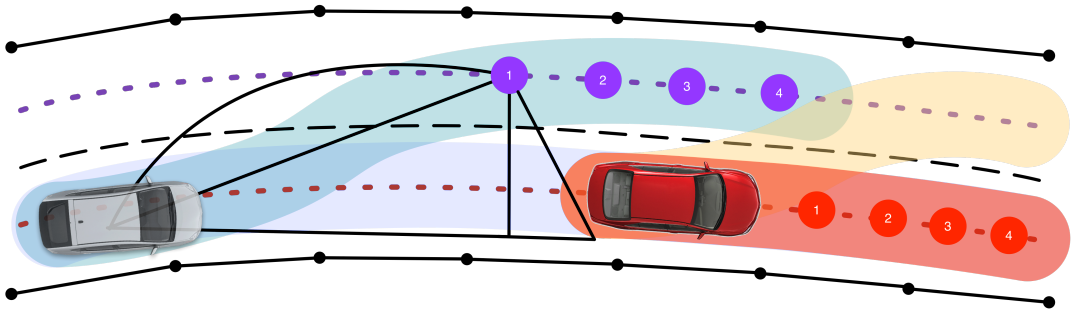

Figure 9: A graphical representation of the lane change scenario on a curved road

- A second question of interest is the scalability of results involving multiple agents and curved roads.
- Pure Pursuit can be represented directly in the Hybrid Program and admits a closed form solution for curvature.
- Using a Hybrid program we now show how the number of paths through the state space grows exponentially with the addition of other agents and events as well as increases in search depth.

We present a scenario as shown in Figure **??**

- The previous example is the simplest possible instantiation of a lane change scenario
- Here we present an increasingly nuanced view of the the vehicle environment system
  - Singleton inital sets and deterministic transitions
  - Initial sets defined by intervals
  - Non-determinism as applied to discrete decisions by the environment
  - Automaton complexity in terms of states as number of agents grows
  - Automaton complexity in terms of states as number of reference trajectories grows
  - Search complexity in terms of path length as number of agents grows
  - Search complexity in terms of path length as number of reference trajectories grows
  - Non-deterministic scheduling of trajectory update
  - Environment agents continuous dynamics are represented as differential inclusions (another form of non-determinism)
  - Ego vehicle agents continuous dynamics are disturbed via errors represented as differential inclusions
  - Ego vehicle and environmental agents subjected to discrete events representing sensor or actuator failures.

6 Conclusions

- Types of Uncertainty:
  - Scenario Configuration

  - Initial Estimation Errors
  - Noise in dynamics
  - Sensor estimation errors
  - Sensing thresholds
- Scenario 1: Lane change on straight road with two agents
- Scenario 2: Curved road with hybrid program representation
  - No longer have clear invariant cut
- Scenario 3: Four way intersection governed by a traffic light
  - Map provides knowledge of obstruction and automatically adjusts behavior
  - Variant: Traffic light
  - Variant: Pedestrian