When you have more than a hammer: multi-formalism verification of autonomous plans

ABSTRACT

As the potential capabilities of autonomous systems expand, it is increasingly necessary to verify the safety properties which evolve in both spatial and temporal domains. Currently, many next generation autonomous systems for both military and civilian applications remain research projects because there is a lack of confidence that they can be used in safety critical applications. In this paper, we develop a framework for formally verifying the safety of autonomous systems operating in unstructured environments and in the presence of other unpredictable agents. Using autonomous vehicles as a running example, we present a decomposition of an autonomous systems' mission into scenarios and agents. The scenario modeled by instantiating agents to create a network of hierarchical communicating hybrid automata (HCHA). We propose that such scenarios clumsily coexist in multiple domains, which when considered together result in undecidable verification problems. We show a method which separates the modes of the HCHA of the target agent and solves a multiplicity of verification problems by applying the best hammer for each mode. In turn we show through a coupling of the problems using the richer HCHA representation as an oracle we are able to state verification results for a complex passing maneuver with nonlinear vehicle dynamics.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

ACM proceedings, LaTeX, text tagging

1. OUTLINE

this section will be removed from final paper, of course

Introduction

- The benefits of autonomous systems.
- The acceptance of autonomous systems in the economy and our daily lives.
- Autonomous vehicles as a running example. What are the "real-world" challenges? (e.g. lane change in some detail, then just list others like pedestrians, other drivers,etc)
- What are the technical challenges facing the development of autonomous vehicles? (e.g. properties with priorities)
- How are these challenges addressed in today's technology and literature?
- What remains to be done?
- Which piece of the puzzle are we addressing in this paper? Illustrate with the running example of a lane change maneuver.
- How are we addressing it?
- The need for formal methods.

Modeling framework

- Scenarios and agents (don't dwell too long on this, just need to mention that we are verifying one scenario in this paper, but we have a plan for how this fits with overall mission verification)
- HCHA: because we need a representation rich enough to express on it different types of properties, which apply to different components of the complex autonomous vehicle
- Specification languages: should be suited to the verification goal. Here, reachability and LTL

Lane change in the formalisms.

- Formal model of lane change in terms of HCHA for system, LTL for scenario goal, and reach sets for safety. Controller is a discrete controller in grid world.
- Lane change requirements can be verified with timed automata, so need to translate HCHA. Give translation.

Figure 1: Aspects of an autonomous car.

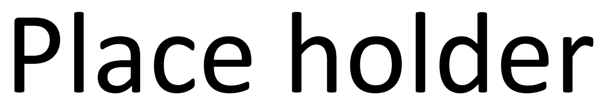

Figure 2: The execution abstraction stack and corresponding typical control hierarchy for autonomous navigation.

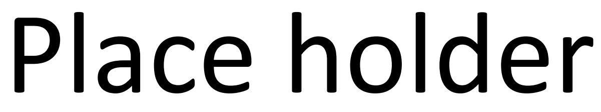

Figure 3: Discrete view of the world for the planning controller. The road is a grid with binary occupancy in each box, and the closed-loop system (car + controller) is given by a finite automaton.

- How results are mapped back to HCHA

Experiment

- introduce the tool here? Should it be introduced as a tool since we don't have a prototype?
- Experimental setup: simulation or actual little vehicles? UPPAAL, dReach, make it available online
- results

Future work

- Tool chain: from design entry to formal DSL (implementing HCHA) to design files for relevant tools
- priority of requirements?

2. INTRODUCTION

Autonomy from human intervention in the machines that surround us promises many benefits. Because of these benefits, autonomous and semi-autonomous systems are now an accepted part of the economy and the household. Examples include robots in shipping warehouses, vacuum robots and the emerging class of personal robots. *Autonomous vehicles are a particularly challenging class of autonomous systems. On a typical trip, the autonomous vehicle must recognize, enter, complete and exit many scenarios in a safe and timely manner.* Example scenarios include traffic lights, roundabouts, pedestrians, weather conditions, etc. It is not known, ahead of time, what the specific sequence of encountered scenarios will be.

The corresponding technical challenges can be broadly divided into three categories: the first is operation in a highly unpredictable environment. For example, what will traffic conditions be, how will other drivers behave and will they obey the laws, will certain parts of the road be inaccessible, and will there be emergency responders on the road needing right of way? Note that these uncertainties occur at several time scales, from the long-term (traffic conditions) to the very short-term (emergency responders driving by). Therefore the on-board controller(s) must always decide: what is the vehicle's objective in the short run?

The second challenge is that task execution must satisfy three different types of constraints: safety, comfort to the passengers and performance. Alternatively, these can be viewed as three objectives to be maximized. See Fig.1. *These constraints belong to different domains of analysis and are most naturally expressed and verified with different formalisms.* How can verification results from one domain be ported over to another domain, so a coherent view of the system's operation arises? *In particular, the safety of the car's passengers and of the people in its immediate environment, is imperative at all times. Because of its importance, guaranteeing safety to a socially acceptable degree requires formal guarantees.* Potential legal repercussions of autonomous car accidents make such guarantees even more pressing. Thus it is essential to understand: what does safety mean in a given driving situation?

The third challenge is that there is a wide separation between the highest levels of plan execution ("go from A to B") and the lowest levels of plan execution ("accelerate steadily for the next 5 seconds"). How do we guarantee consistency between the commands issued at the different levels? The most common approach to dealing with this *execution abstraction stack* is to perform a corresponding separation of the control objectives and design separate controllers for each layer [14]. See Fig.2.

Today we have theories that deal with high-level planning in a discrete grid world *(see Fig.6.5), determining which 'box' the car should go to given what boxes are occupied by the other vehicles, and given non-deterministic descriptions of their behaviors in Linear Temporal Logic(LTL).* E.g., [16, 12]. We note that these controller synthesis algorithms suffer from a doubly-exponential computational complexity [11], though polynomial complexity algorithms exist for a fragment of LTL [7]. *We also have methods for verification of temporal logic properties of the closed-loop system* modeled as some kind of finite automaton, e.g., [**?**]. *At the other end of the abstraction spectrum, control theory provides analysis and design tools of low level controllers. Automatic analysis tools exist [8] but are limited in scope.* See, e.g.,

Sriram's Lyap from simulations, Geir Dellerud's STORMED hybrid systems, decidable classes of HS

, or provide probabilistic guarantees of completeness

GF's staliro

. *Because of the large degree of unpredictability, compute- and memory-intensive low level methods can't be used alone (let alone manual methods).*

The gap between high-level grid world as we will call it, and the low-level physical world is two-fold: first, high-level synthesis and verification algorithms apply to the grid world and to model representations that bear no a priori connection to the most accurate models of the system that we have, namely, the models of the continuous dynamics and of the code controlling them. Secondly, the verification effort at the low-level is unaware of the commands that will be issued at the high level. *To bridge the gap between the high-level planning and verification methods, and the lower-level control-theoretic analysis tools, guaranteed abstractions have been proposed.* Given a representation of a system, e.g. as a system of differential equations, a guaranteed abstraction of that system produces a second representation which is simpler, and yet whose external behavior is similar, in a certain precise sense, to that of the original system. Some of these abstractions produce finite-state systems ???h. tanner, tabuada, which are amenable to model checking [???Principles of MC]. Others result in lower-dimensional (but still infinite-state) systems [???girard, julius, Mahesh Vishwanathan and IMEAD pabithra], for which some verification techniques are better suited. Yet, the conditions under which these abstractions can be obtained restrict their application when it comes to industrially developed systems, where the system is conceptually decomposed into different *aspects*, each of which is developed and tested by a different team.

Because of the safety imperative, we can't rely on non-guaranteed abstractions. In this paper, we demonstrate the need for using multiple formalisms in the verification of autonomous plans. We illustrate this with a case study of a typical lane change maneuver.

EXAMPLE 1 (LANE CHANGE 1). *change to Lane change 1,2,...*
We use a passing scenario as a running example. The scenario shows the ego vehicle driving in the right lane of a unidirectional two lane road network. Another car is driving in front of the ego vehicle at a lower speed. The ego vehicle's planner decides to initiate a passing maneuver, which involves moving to the left lane, accelerating to pass the other vehicle, then moving back into the right lane ahead of the other vehicle. The scenario terminates with an intersection governed by a traffic light. The safety requirement is that the ego vehicle must remain at least a certain distance away from the other vehicle. The mission goal is to arrive at and cross the intersection. The verification engineer is given a vehicle controller and must check formally that the closed-loop system (ego vehicle + controller), in the representation used by the controller, performs correctly and safely. E.g., that representation can be a finite automaton, and the road is represented by a finite grid. Thus model checking will happen on this finite representation. Yet the safety requirement requires us to work with the more accurate continuous dynamics to determine the cars' movements during a lane change, especially during lateral changes of position. Thus we need both model checking techniques, and reachability computations, to verify the safe and correct completion of this scenario. From a planning perspective we define the safety of the discrete controller of the target vehicle to be the satisfaction of the following safety and liveness properties:

1. *The target vehicle may not exceed the speed limit (safety).*
2. *The target vehicle may not collide with any other vehicles (if they exist) on the road (safety).*
3. *The target vehicle must stop at a red light (safety).*
4. *The target vehicle must eventually complete the scenario (liveness). It is possible to specify these properties using LTL operators (always and eventually).*

2.1 Notation

We denote the set of integers including 0 with $\mathbb{N}$. Given a subset S of the reals, $S^* = S\backslash\{0\}$ and $S_+ = S\cap[0,\infty)$, while $S_+^* = S\cap(0,\infty)$. A *discrete* variable takes values in a finite set, and a *continuous* variable takes values in a closed subset of $\mathbb{R}^n$ for some n. Given a vector $x = (x_1,\dots,x_n) \in \mathbb{R}^n$, and a set of indices $V \subset \{1,\dots,n\}$, $\mathbf{pr}_V(x)$ is the projection of x onto the subspsace whose dimensions are indexed by V. E.g. with $x = (x_1,x_2,x_3,x_4)$ and $V = \{1,3\}$, $\mathbf{pr}_V(x) = (x_1,x_3)$. We also define $x^\uparrow := \mathbf{pr}_V(x)^{-1}$ when V is clear from the context. A set-valued map $F : \mathbb{R}^n \rightrightarrows \mathbb{R}^m$ maps each x in its domain to a subset of its range. For a set $K \subset \mathbb{R}^n$, we abuse notation by writing $F(K)$ for $\cup_{x\in K}F(x)$.

3. THE MODELING OF AUTONOMOUS PLANS

3.1 Decomposing a mission into scenarios and agents

To motivate our thinking about the problem of safe autonomous navigation, we consider the case of a trip between two designated points. On such a trip, the autonomous car will face a number of situations, or *scenarios*, which it must know how to recognize, enter, negotiate, and exit in a safe and timely manner. For example, the car will encounter roundabouts, traffic lights and four-way intersections, on-ramps to highways and merging lanes, lane changes, pedestrians at designated and non-designated crossing points, and various traffic signals like speed limits, school zones, etc. We therefore think of a driving mission as an ordered sequence of scenarios. The exact sequence that will be encountered is not know ahead of time. Moreover, it is clear that some scenarios, like traffic lights, will be encountered more than once, albeit with minor site-specific differences. *The key observation however is that the diversity of scenarios is, to a first order of approximation, finite.* That is, there is a recognizable, finite set of scenario types that is sufficient to describe most autonomous navigation missions.

Formally, let $\mathcal{S} = \{s_0, s_1, \dots, s_{N-1}\}$ be a set of scenarios. E.g., s_0 = Roundabout, s_1 =Lane change, s_2 = PassOnTheLeft, etc. (Each of these scenario types will be itself formalized in what follows). Then a mission M is a finite string on $\mathcal{S}$, $M \in \mathcal{S}^*$. Note that each such scenario may have multiple realizations (e.g. a roundabout with 3 vehicles in it, a roundabout with 1 vehicle, etc.), so in what follows we will distinguish between scenario types (elements of S) and scenario *instances* which can be thought of as instantiations of scenario types. When we speak of just a scenario, we mean a scenario instance.

Therefore, if we can verify the safety of the autonomous system's behavior in each scenario, and compose the scenarios in a safe manner, then we have verified that the mission is safe.

Within each scenario, we can recognize a recurring set of entities: the autonomous vehicle whose operation we seek to verify (a.k.a. the *ego* vehicle), the other vehicles on the road, pedestrians, traffic signage, and the road network itself. We will refer to these as *agents*:an agent is an entity that functions continuously and autonomously in the environment, and whose presence can be sensed by other agents.

main literature for agents

Formally, we recognize the following set of four agent *types*

$$\mathcal{A} = \{\texttt{vehicle, pedestrian, road, trafficSignage}\}$$

Each agent type can have many (parametrized) instances, e.g. `trafficSignage` can have instances speedLimit(70mph), speedLimit(25mph), HOVLane(3pm-6pm). Note that the **`vehicle`** agent type can have instances that are autonomous vehicles, and other instances that are human-driven (and therefore that have different behaviors from the ego vehicle, which is by definition autonomous). We denote the set of instances of agents in $\mathcal{A}$ by $I(\mathcal{A})$. Clearly, $I(\mathcal{A})$ is infinite. When we just speak of an agent, we mean an agent instance.

In addition to the perceptible traffic signage, the car must observe the imperceptible laws of the road, i.e., laws that don't have a physical manifestation. For example, one such law says that on the highway, a faster car must always pass on the left of a slower car that's in front of it. These laws constitute constraints on the car's behavior in some or all of the above scenarios.

Formally, a scenario instance is a tuple $(T, A, \mathcal{B}, L)$ where

- $T \in \mathbb{N}^*$ is a positive discrete time horizon on the scenario.
- A is a collection of agent *instances* from the set $I(\mathcal{A})$. The set A always includes the ego vehicle, i.e., the system whose safe behavior we want to verify.
- $\mathcal{B} = \{\mathcal{B}_a \mid a \in A \setminus \{egoVehicle\}\}$ is a bounded-time behavior description for each of the agent instances. Describing the behavior $\mathcal{B}_a$ of an agent instance requires us to formally describe an agent. We do so in Section 3.2.
- $\mathcal{L} = \{l_0, \dots, l_p\}$ is a finite set of $p \in \mathbb{N}$ traffic laws. A law l_i is a temporal logic formula indicating how the law (when it is active) constrains the behavior of the vehicle. I.e. it is a specification on the ego vehicle that must be satisfied. The logic in which l_i is expressed will depend on the formalism used to model the scenario. We will have more to say about this in the following sections.

Each law represents a time-varying constraint on the allowed ego vehicle's behavior, *but not necessarily on the behavior of other vehicles.* That is, our verification algorithms do not necessarily assume that other vehicles will obey traffic laws. What assumptions we make on the behavior of other vehicles is captured in $\mathcal{B}$.

Example 2 (Passing continued). *This scenario contains two instances v_{ego} and v_2 of the `vehicle` agent type, one `road` agent, no `pedestrians` and one `trafficSignage` agent. The behavior $\mathcal{B}_{v_2}$ is given by a sequence of reach sets as detailed in Section 4. An example applicable law, expressed in discrete time LTL, is $l := \Box(position_{ego} = a \implies Xposition_{ego} \geq a)$, to indicate that backing up is not allowed. Note that LTL is not ideally suited for this requirement since we need one formula per speed threshold a. A more concise logic would allow us to directly say $l := \Box(position_{ego}(t+1) \geq position_{ego}(t))$ (e.g., TPTL), while still being reasonably computationally tractable.*

cite Alur and Henzinger TPTL JACM94

3.2 Communicating hierarchical hybrid automata

As noted in the introduction, the execution of an autonomous plan involves many levels of abstraction. The verification must cover all these levels.

We use the formalism of hierarchical communicating hybrid automata (HCHA) to model agents in the scenario. We start with a few basic elements: an agent (of one of the types described above) has a state vector $x \in \mathbb{R}^n$. (Different agents may have different dimensions.) The state includes all information of relevance to the system designers.An agent is modeled as a *communicating* hybrid automaton.

Definition 3.1 (Communicating hybrid automaton). *A driven **communicating hybrid automaton** (CHA) Σ with inputs in U is a tuple*

$$\Sigma = (X, L, E, Inv, \{F_\ell\}_{\ell \in L}, \Gamma, Re, \Pi, C)$$

where

- *$X \subseteq \mathbb{R}^n$ is the continuous state space of the system.*
- *$L \subset \mathbb{N}$ is a finite set of control locations that the system switches through.*
- *$H \triangleq L \times X$ is then the hybrid state space.*
- *$F_\ell : X \times U \to \mathbb{R}^n$ defines a differential inclusion governing the evolution of the state in location ℓ: $\dot{x} \in F_\ell(x, u)$.*
- *$Inv : L \to 2^X$ defines an invariant condition on the state while in location ℓ: $x \in Inv(\ell)$.*
- *$E \subseteq L \times L$ is the set of location transitions (a.k.a. switches, or jumps).*
- *$\Gamma : E \to 2^X$ is the guard condition that enables a location transition. Namely, the system switches between ℓ_i and ℓ_j when the state x is in $\Gamma((\ell_i, \ell_j))$.*
- *$Re : E \times X \to L \times X$ is a reset map that, given a transition $e = (\ell_i, \ell_j) \in E$ and a point x on the guard $\Gamma(e)$, maps x to a point x^+ in the invariant set of the next location ℓ_j:*
$$Re : ((\ell_i, \ell_j), x) \to (\ell_j, x^+)\,, x^+ \in Inv(\ell_j)$$
- *$\Pi : \mathbb{R}^n \to \mathbb{R}^p$ is an output function,*
- *$C = \{c_1, \dots, c_p\}$ is a set of $p \in \mathbb{N}$ perception conditions (defined below).*

The differential inclusion F_ℓ in each location of the CHA models the continuous-time closed-loop behavior of the low-level controlled systems, i.e., the electromechanical systems of the car such as the powertrain. As stated earlier, the state x captures all the relevant information about the agent's dynamics. It is widely recognized now that different navigation situations call for different control laws.

cite

This is modeled by the locations of the CHA: each location represents the application of a different control law that is appropriate for the situation.

The solutions of regular ODEs are given as functions of the independent variable time t. Because hybrid automata also include a discrete evolution of the state (namely, via the reset map Re), their solutions will be given as a function of time t and jump number j. We first define hybrid time domains and arcs.

Definition 3.2 (Hybrid time domains and arcs [4]). *A subset $E \subset \mathbb{R}_+ \times \mathbb{N}$ is a* compact hybrid time domain *if*

$$E = \bigcup_{j=0}^{J-1} [t_j, t_{j+1}] \times \{j\}$$

for some finite increasing sequence of times $0 = t_0 \le t_1 \le t_2 \le \ldots \le t_J$. We say E is a hybrid time domain *if for all $(T, J) \in E$, $E \cap ([0, T] \times \{0, \ldots, J\})$ is a compact hybrid time domain. Define $\sup_t E = \sup\{t \mid \exists(t, j) \in E\}$, $\sup_j E = \sup\{j \mid \exists(t, j) \in E\}$, and $E.last = (\sup_t E, \sup_j E)$.*

A hybrid arc *ϕ is a function supported over a hybrid time domain $\phi : E \to \mathbb{R}^n$, such that for every j, $t \mapsto \phi(t, j)$ is absolutely continuous in t over $I_j = \{t : (t, j) \in E\}$; we call E the domain of ϕ and write it $\mathbf{dom}\phi$. Finally, $\mathbf{dom}_t\phi = \mathbf{pr}_{\mathbb{R}_+}(\mathbf{dom}\phi)$ and $\mathbf{dom}_j\phi = \mathbf{pr}_{\mathbb{N}}(\mathbf{dom}\phi)$*

Note that each I_j is an interval. As observed in [13], hybrid time domains are equivalent to hybrid time trajectories used in [9] to define executions of hybrid automata. Output and state trajectories of hybrid automata will be modeled as hybrid arcs.

Given a hybrid automaton Σ and a point $h_0 = (\ell_0, x_0)$, a *solution* ϕ_{h_0} of Σ (aka *state trajectory*) is given by a hybrid arc ϕ satisfying the following conditions:

1. $\phi_{h_0}(0, 0) = h_0$
2. $\phi_{h_0}(t, j) = (\ell(t, j), \mathbf{x}(t, j)) \in H \ \forall (t, j) \in \mathbf{dom}\phi$
3. For each j s.t. I_j has a non-empty interior,
$$\dot{\mathbf{x}}(t, j) \in F_{\ell(t,j)}(\mathbf{x}(t, j))$$
for almost all $t \in I_j$, and $\mathbf{x}(t, j) \in Inv(\ell(t, j))$ for all $t \in \text{int}I_j$. Moreover,
$$\dot{\ell}(t, j) = 0 \ \forall t \notin \{\inf I_j, \sup I_j\}$$
4. For all j s.t. $(t, j) \in \mathbf{dom}\phi$, $(t, j+1) \in \mathbf{dom}\phi$, it holds that $\mathbf{x}(t, j) \in \Gamma(\ell_i, \ell_k)$ for some i, k, $\mathbf{x}(t, j+1) = Re(\mathbf{x}(t, j), (\ell_i, \ell_k))$, and $\mathbf{x}(t, j+1) \in Inv(\ell_j)$.

See Fig. 4 for an illustration. We consider that the discrete time variable j simply counts the location changes (via (3) and Condition 4). The transition itself between locations takes 0 real time. The location changes can only occur when the continuous state x enters a guard set (Condition 4). The sequence of control locations that the trajectory ϕ visits is denoted by loc(ϕ).

If ϕ is a state trajectory, then $\Pi \circ \phi : \mathbf{dom}\phi \to H$ is called an *output trajectory*. We assume that Π is such that an output trajectory is also a hybrid arc, supported on the same hybrid time domain as the corresponding state trajectory: $\mathbf{dom}\phi = \mathbf{dom}\Pi \circ \phi$.

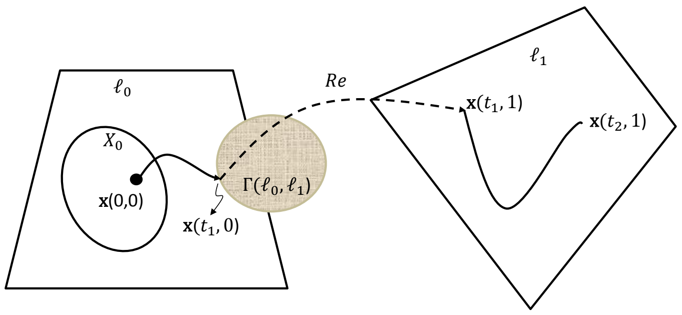

Figure 4: State trajectory of a hybrid automaton.

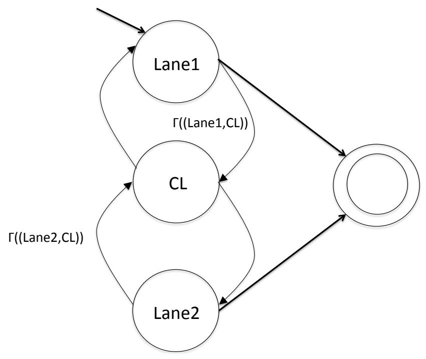

Figure 5: CHA for Ego vehicle

Definition 3.3. *$\mathcal{S}_\Sigma(h_0)$ will denote the set of solutions of Σ with initial condition $\phi(0, 0) = h_0$, and $\mathcal{S}_\Sigma$ will denote the set of all solutions of Σ. A hybrid automaton is said to be* deterministic *if $\mathcal{S}_\Sigma(h_0)$ is a singleton for any h_0.*

We will use ϕ_h to refer to the trajectory starting at h. The system has a finite set of initial locations $L_0 \subset L$ in which it could start, and within each such location ℓ, a set of initial continuous states $X_\ell \subset X$ in which it could start. Any solution to the system's equations must start there. We denote

$$H_0 = \bigcup_{\ell \in L_0} \{\ell\} \times X_\ell$$

the hybrid initial set, and write

$$Init = \bigcup_{\ell \in L_0} X_\ell$$

Example 3 (Lane change). *We focus on the lane change sub-scenario of the Passing scenario, and model the ego vehicle as an HCHA. The closed-loop ego vehicle can be represented by the CHA of Fig.5. Location ℓ_1 indicates the vehicle is in lane 1, and ℓ_2 indicates it is in lane 2. Location CL is the Change Lane location. The state vector $x = (\beta, \Psi, \dot{\Psi}, v, s_x, s_y, \delta)$ consists of various dynamical quantities from the* bicycle model *??? cite of a car. In particular, Ψ is the heading angle, v is the speed, and (s_x, s_y) is the vehicle position. The continuous dynamics for the ego vehicle are identical for locations ℓ_1, ℓ_2, CL, except for a discrete jump in the heading angle Ψ to effect the lane change. For example, $Re((\ell_1, CL), x) = (\beta, \Psi + \pi/4, \dot{\Psi}, v, s_x, s_y, \delta)$. See ???Matthias for details. The transitions (ℓ_i, CL) are trig-*

gered by the guard condition

$$\Gamma((\ell_i, CL)) = \{\|x_e - x_o\| \leq d \wedge 0 < v_o < v_e\}$$

Guard conditions on (CL, ℓ_i) insure that a lane change can not be aborted (more complex models will allow maneuver interruption). The final state (indicated by the usual double circle) indicates some pre-determined end to the scenario, and transition into it can be guarded, for example, by a clock.

3.2.1 Abstraction levels

We formally define the abstraction levels at which it is controlled, and consequently at which it is verified. A *hierarchical* CHA is a CHA with a chain of partitions defined on its location set L, and a corresponding chain of simplified continuous dynamics.

Definition 3.4 (Hierarchical CHA). *A* ***hierarchical CHA*** *(HCHA) is a tuple $(\Sigma, \{V_i\}_{i=0}^a, \{P_i\}_{i=0}^a, \{\Sigma_i\}_{i=0}^a)$ where Σ is an n-dimensional CHA, each V_i in the chain*

$$V_a \subset V_{a-1} \subset \ldots \subset V_1 \subset V_0 = \{1, \ldots, n\}$$

is the set of surviving variables *at level i, and $\{P_0, P_1, \ldots, P_a\}$ is an ascending chain of partitions of the set L, that is:*

- $P_0 = L$
- *For every $i = 1, \ldots, a$, $P_i \subset 2^L$ partitions L: $[\ell] \cap [\ell'] = \emptyset$ for all $[\ell], [\ell'] \in P_i$ and $\cup_{\ell \in L}[\ell] = L$*
- *For all $i < a$, for all $[\ell] \in P_i, \exists [\ell'] \in P_{i+1}$ s.t. $[\ell] \subset [\ell']$*

Every element of $\{\Sigma_i\}_{i=1}^a$ is an n_i-dimensional CHA

$$\Sigma_i = (X_i, L_i, E_i, Inv_i, \{F_{q,i}\}_{q \in L_i}, \Gamma_i, Re_i, \Pi_i, C_i)$$

where $\Sigma_0 = \Sigma$, $n_i \geq n_{i+1}$,

$$\begin{array}{rcl}
L_i & = & P_i \\
(q, q') \in E_i & \textit{iff} & \exists \ell \in q, \ell' \in q' \text{ s.t. } (\ell, \ell') \in E_{i-1} \\
X_i & = & \mathbf{pr}_{V_i}(X_{i-1}) \\
\Gamma_i((q, q')) & = & \mathbf{pr}_{V_i}(\cup_{(\ell,\ell') \in q \times q'} \Gamma_{i-1}((\ell, \ell'))) \\
Inv_i(q) & = & \mathbf{pr}_{V_i}(\cup_{\ell \in q} Inv_{i-1}(\ell)) \\
F_{q,i}(x) & \supset & \mathbf{pr}_{V_i}(\cup_{\ell \in q} F_{\ell,i-1}(x^\uparrow)) \\
Re_i(e, x) & = & \mathbf{pr}_{V_i}(\cup_{e' \subset e} Re_{i-1}(e', x^\uparrow)) \\
\forall x \in X_{i-1},\ \mathbf{pr}_{V_i}(x) & = & \mathbf{pr}_{V_i}(Re_{i-1}(e, x)) \\
C_i & = & C
\end{array}$$

Finally, for all $e \in E_{i-1}, x \in \Gamma_{i-1}(e)$,

$$\mathbf{pr}_{V_i}(x) = \mathbf{pr}_{V_i}(Re_{i-1}(e, x))$$

should F be the convex hull of the union?

Every element P_i of the partition chain represents an abstraction level. An element $[\ell]$ of P_i is a 'super-mode' which aggregates several modes of P_{i-1} (namely, the elements of P_{i-1} in the equivalence class $[\ell]$). The state vector is reduced in dimension by projecting the state vector at abstraction level $i-1$ onto the variables V_i of the new level. The set V_i is part of the data of the HCHA. For example, it may consist of the state variables that appear in the guard conditions leading out of the super-modes $q \in L_i$. The guards and invariants are defined accordingly, by doing the same projection for all guard and invariant conditions on a given transition in Σ_{i-1}. The new level's dynamics F_i are part of the data of Σ_i. For example, they may be obtained by Model Order Reduction from the dynamics of Σ_{i-1}, or some domain-specific transformation. The input set is correspondingly modified. Note that by imposing $C_i = C$ we impose that the variables involved in each perception condition survive all levels. This is a reasonable assumption when we consider that perception conditions involve physical quantities that can be perceived by other agents in the environment, and therefore are unlikely to be dropped by designers as 'irrelevant' at some level of abstraction.

Remark 3.1. *It is possible to define successive abstraction levels to perform more elaborate mappings on the state vector than simple projections, and the rest of the paper would follow with some simple modifications. We define the HCHA with projections because this is more typical of the manner in which different design teams design their systems, with attention paid to the quantities of interest at their level, while assuming that lower levels somehow provide them with readings (or computations) of this data.*

add HCHA fig and mention of fig to clarify

To formally state the relation between executions at different abstraction levels, we must first define executions of hybrid automata.

3.2.2 Properties of the abstraction chain

Theorem 3.1. *Let $(\Sigma, \mathcal{V}, \mathcal{P}, \{\Sigma_i\})$ be a HCHA. Let $\phi = (\ell, \mathbf{x}) \in \mathcal{S}_\Sigma$ be a solution of CHA Σ with continuous part $\mathbf{x}$. Then for every $1 \leq i \leq a$, there exists a solution $\phi^{(i)} = (q, \mathbf{x}^{(i)}) \in \mathcal{S}_{\Sigma_i}$ such that $\mathbf{x}^{(i)} = \mathbf{pr}_{V_i}(\mathbf{x})$. That is, for every $(t, j) \in \mathbf{dom}\phi$, there exists $(t, j') \in \mathbf{dom}\phi^{(i)}$ s.t. $\ell(t, j) \in q(t, j')$ and $\mathbf{x}^{(i)}(t, j) = \mathbf{pr}_{V_i}(\mathbf{x}(t, j'))$.*

Proof. We prove this for $i = 1$. The general case follows by induction. Fix $i = 1$, and write V for V_i in this proof. Let $\mathbf{x}$ be as stated, and define $y : \mathbf{dom}\mathbf{x} \to \mathbb{R}^{n_i}$ by $y(t, j) = \mathbf{pr}_V(\mathbf{x}(t, j))$ for all $(t, j) \in \mathbf{dom}\mathbf{x}$. We will construct from y a hybrid arc that satisfies the conditions of a solution of Σ_i.

Let I_j be an interval in $\mathbf{dom}\phi$ with non-empty interior, and take $t \in I_j$ s.t. $\dot{\mathbf{x}}(t, j) \in F_\ell(\mathbf{x}(t, j))$. Then $\dot{y}(t, j) = d(\mathbf{pr}_V(\mathbf{x}(t, j)))/dt = \mathbf{pr}_V(\dot{\mathbf{x}}(t, j)) \in \mathbf{pr}_V(F_\ell(\mathbf{x}(t, j))) \subset F_q(y^\uparrow(t, j))$. For all $t \in \text{int}I_j$, $y(t, j) = \mathbf{pr}_V(\mathbf{x}(t, j)) \in \mathbf{pr}_V(Inv(\ell)) = Inv(q)$. Moreover, for all t different from the endpoints of I_j, $q(t, j) = [\ell]$ and so $\dot{q}(t, j) = 0$. Thus condition 3 is satisfied for y.

Now consider the jumps: let $t_r = \sup I_J$. If the jump at (t_r, j) leads from ℓ to a location ℓ' such that $[\ell']$ is not q, then it can be shown that this is also a jump for y. However, if the transition is *internal* to mode q, i.e.

$$[\ell] = [\ell']$$

then this violates condition 4 since $y(t_r, j)$ is not in a guard set of Σ_i. Moreover, because of the reset that occurs at (t_r, j), if $\mathbf{pr}_V(\mathbf{x}(t_r, j+1)) \neq \mathbf{pr}_V(\mathbf{x}(t_r, j))$, then y is not absolutely continuous as a function of time within each mode. To get around these difficulties, we invoke the last property defining an abstraction chain. Namely, that the surviving variables at each level are not affected by resets on internal transitions. This removes the second difficulty above since now $y(t_r, j+1) = y(t_r, j)$.

We define $\mathbf{x}^{(i)}$ by erasing the internal transitions from the domain of y. Let $\mathcal{J}$ be the set of indices of internal transitions, that is, $\mathcal{J} = \{j \mid \exists t.(t, j) \in \mathbf{dom} \mathbf{x}, (t, j+1) \in \mathbf{dom} \mathbf{x}, [\ell(t, j)] = [\ell(t, j+1)]\}$.

TBC

$\square$

initial points

THEOREM 3.2. *Let $(\Sigma, \mathcal{V}, \mathcal{P}, \{\Sigma_i\})$ be a HCHA. Either Σ_i be a CHA in the abstraction chain. Then either Σ_i has the following property:*

- *For every solution $y \in \mathcal{S}_{\Sigma_i}$, there exists $\mathbf{x} \in \mathcal{S}_\Sigma$ s.t. $y = \mathbf{pr}_{V_i}(\mathbf{x})$*

or Σ_i can be refined in at most $|L| \cdot n$ steps so that it has the property, where L is the location set of Σ and n is its dimension.

PROOF. (Sketch) Assume that a solution $\mathbf{x} \in \mathcal{S}_\Sigma$ as stated does not exist. Broadly speaking, this can happen because Σ_i can choose a sequence of continuous dynamics that can not be selected by Σ, since the dynamics of Σ_i are larger (in the sense of Def. 3.4) than those of Σ. More specifically, there are two ways this can happen:

1. Either, within the same I_j in the domain of y, $\dot{y}$ obeys two different dynamics F_ℓ and $F_{\ell'}$ for $\ell, \ell' \in q \in L_i$. In general, this can not be produced by a Σ trajectory.[1]

2. Or, within the same I_j in the domain of y, $\dot{y}$ obeys two different dynamics: over $[t, t') \subset I_j$ it obeys $F_\ell(\mathbf{x}(t))$ where $\mathbf{x}$ is a Σ solution that extends y, i.e. $\mathbf{x} = [z, y] \in \mathbb{R}^n$. And at t' it obeys $F_\ell(x')$ where $x' \in y^\uparrow(t') \setminus \{\mathbf{x}(t')\}$.

Note that if we exclude the above two cases, then the discrete dynamics (i.e. the transitions) of Σ_i can always be reproduced by Σ.

Now suppose Case 1 happens: then we refine the partition $\mathcal{P}$ by creating a separate part for ℓ'. I.e. if $\ell' \in P_1$, then we refine $\mathcal{P}$ to $\mathcal{P}' = \mathcal{P} \setminus \{P_1\} \cup (P_1 \setminus \ell') \cup \{\ell'\}$. This guarantees that there won't be invisible dynamics changes caused by the (previously internal) transition (ℓ, ℓ'). Since there are $|L|$ locations, this refinement is applied at the most $|L|$ times. Doing the refinement $|L|$ times corresponds to preserving the discrete dynamics perfectly.

Now suppose Case 2 happens: then we must eliminate the invisible dynamics changes that occur because of the projection of the state vector onto V_i. Let W be the set of indices where $\mathbf{x}(t')$ and x' differ. First, because $\mathbf{x}$ extends y, and therefore agrees with it over V_i, W and V_i don't intersect. Thus $W = \{i_1, i_2, \ldots, i_m\}$ consists entirely of 'dead' variables (i.e. non-surviving). We refine Σ_i as follows: promote the smallest index in W to surviving at level i: i.e. define $V_i' = V_i \cup \{i_1\}$. Then the dynamics of Σ_i must be refined accordingly, namely $F_q'(y) := \mathbf{pr}_{V_i'}(\cup_{\ell \in q, x \in y^\uparrow} F_\ell(x))$. Thus the solutions of Σ_i can no longer choose the dynamics $F_\ell(x')$ which led to Case 2 above. Of course, the rest of the variables in W could still cause this problem. But we refine one index at a time, since every refinement produces a new set of dynamics, which might lead to a different violation, and make others impossible. I.e., the set W depends on the current V_i and is not static. Since there are n indices, and a new index is added to V_i with each refinement, this can happen a maximum of n times. Surviving all indices (i.e., $V_i' = \{1, \ldots, n\}$) corresponds to preserving the continuous dynamics perfectly.

Thus there is a total of $|L|n$ refinements possible before every Σ_i trajectory is the projection of a Σ trajectory. $\square$

[1]Unless it so happens that some element of $y(t_j, j)^\uparrow$ is in $\Gamma(\ell, \ell')$ (where t_j is the transition time).

Note that this maximum number of refinements is *independent of the level of abstraction i*.

3.2.3 *Inputs and perception conditions*

Without the perception conditions C, and without the inputs, a CHA is simply a hybrid automaton [5]. The inputs are used to model two phenomena: first, some inputs will model the output of the perception layer of the system.

See fig. – tartan figure from intro –

The perception layer of agent A interprets the perceptible aspects of other agents, i.e. $\Pi_{A'}(x)$ for $A' \neq A$. Depending on the abstraction level at which they are considered, these inputs may be discrete (e.g., a boolean to indicate the presence of a car in the current lane) or continuous (e.g., the estimated position of a pedestrian). Second, other inputs will model control commands to the vehicle issued by its own autonomous controllers. E.g. a controller may cause the vehicle to switch modes: such inputs are discrete. Another controller (at a different level of abstraction) produces a piecewise continuous throttle angle command. Such an input is continuous.

To model how agents perceive each other at a high level, we consider that certain aspects of an agent can occasionally be perceived by other agents. The simplest such aspect is the agent's visual appearance. Formally, associated to an agent is a function $\Pi : \mathbb{R}^n \to \mathbb{R}^p$. At each time instant (t, j), the agent broadcasts $\Pi(x(t, j))$. Not every other agent can perceive this broadcast. Given two agents A_1 and A_2 in the same scenario, with agent A_1 broadcasting $y = f(x)$, agent A_2 must meet certain conditions on its state to be able to receive (and act upon) the information broadcast by A_1. Specifically, if $y(t, j) = (y_1, \ldots, y_p)$, then to listen to y_k, $k = 1, \ldots, p$, the state $x_2(t, j)$ of A_2 must satisfy some boolean 'listening' condition $c(x_2, k)$. For example, every agent A_1 must transmit its visual appearance - it can't become invisible. For another agent A_2 to perceive it, A_2 must be close enough and with a line of sight to A_1. Then

$$c(x_2) \equiv ||x_1 - x_2|| \leq d \wedge \forall i \neq 1, 2, A_1, A_2, A_i \text{ not aligned}$$

This model of communication subsumes traditional models of processes that communicate via shared variables, like that in

insert charon citation

, and generalizes it by imposing conditions under which communication is possible, rather than allow communication at all times.

It should be noted here that the above definitions define a mathematical model of an agent, and not a programming language for autonomous agents. I.e. we are not concerned with how such agents are simulated, or how interrupts and error conditions (like violation of invariants) are handled in a software package that implements HCHAs. The reader interested in such details can consult, for example, the Charon literature. [2]

cite charon

Ptolemy

Loos on parametrized arch views

HCHA can be translated to various other formalisms. We consider the case of timed automata, and of ODEs (i.e. dynamical systems).

4. OTHER AGENTS

In this section we give the model of a `road` *agent, and specialize the HCHA to other (non-ego)* `vehicles` *and* `pedesrians`*.*

DEFINITION 4.1. *A **road agent instance** is given by a tuple $(R, G_1, \dots, G_r)$ where R is a subset of $\mathbb{R}^2$ and $G_i = (V_i, E_i)$ is a directed graph for which there exists a partition R_i of R and a bijection $b_i : V_i \to R_i$ between R_i and V_i such that*

- $(v, v') \in E_i$ *iff* $b_i(v) \cap b_i(v') \neq \emptyset$
- R_i *is a refinement of* R_{i+1}

The direction of each edge in G_i is part of the road agent's data. The set R is the most accurate representation of the road network in a given scenario. The successive partitions R_i represent coarser and coarser representations of the road, and their operational view that is used by controllers is given by the graphs G_i.

For verifying the autonomous agent, only the perceptible behavior of other agents is important, not their internal structure.

The behavior of other agents is part of the scenario description.

For the other vehicles and pedestrians, the basic perceptible external behavior is their position at given moments in time.

DEFINITION 4.2. *Let s be a scenario, and let A be an agent instance of type* `vehicle, pedestrian` *in s. The **basic HCHA of** A takes the form*

$$\Sigma = (\mathbb{R}^2, \{1\}, \emptyset, \mathbb{R}^2, \{F_1\}, \emptyset, Id, Id, \{c\})$$

where $c(x) = \|x - x_A\| \leq d$ for some $d > 0$.

The basic HCHA for pedestrians and other vehicles simply models how their 2D position evolves. Notice that because the dynamics are given by an inclusion rather than an equation, other agents' behavior is nondeterministic and their position at each moment in time is a non-singleton. We denote that set by $X(t, 0)$

$$X(t, 0) := \{x(t) \mid x \text{ is a solution of the inclusion.}\}$$

(The j parameter always equals 0 since the basic HCHA only has one mode.) This models the uncertainty about their behavior. Deterministic behavior is naturally captured by making $F(x)$ a singleton for every x, i.e. a differential equation.

[2]Charon is a formal programming implementation of hierarchical communicating hybrid automata, although the systems they model have some differences with the ones defined here. Covering these differences is outside the scope of this paper.

This model is sufficient for basic safety and correctness verification, and for imposing minimally reasonable restrictions on their behavior. For example, we can impose that a vehicle never leaves the road by making the velocity vectors $F(x)$ turn inward at road edges. Because their position is perceptible, the ego vehicle can formulate safety as the property that it never intersect their position set. If needed, this basic HCHA can be enriched to model more sophisticated sensing by the ego vehicle (e.g. tracking of shape), or different behavior of the other agents as required by the scenario. Finally, note that in practice (e.g., in a software implementation) it may be more practical to directly give the trajectory $X(\cdot, 0)$ of an agent rather than the dynamics $\dot{x} \in F(x)$. It might also be necessary to give that trajectory in discrete time (e.g. as a sequence of reach sets.) This doesn't affect the verification algorithms.

Examples of traffic signage modeling will be covered in the examples.

4.1 What is safety?

For an autonomous vehicle, safety is equivalent to not colliding with other agents (cars, pedestrians, and traffic signage), or entering unsafe regions of the environment, like opposing traffic lanes or obstacles.

In a given scenario, the safety imperative gives rise to more specific requirements. We express these in a temporal logic: it allows formal verification, and applies to output traces regardless of the level of abstraction of the generating system. We can also express (non-safety) mission goals in temporal logic.

The specific logic we use depends on the property being expressed.

EXAMPLE 4 (LANE CHANGE CONTINUED). *The safety imperative in the lane change scenario can be expressed as the conjunction of the following LTL formulae on the grid world evolution and on the reach sets.*

The mission goal can be expressed as follows

Future: Can we take the absolute safety mandates and automatically specialize them to scenarios? I.e. given the set of state trajectories of the car, automatically derive, from the scenario, some constraints that constrain the search for the subset of trajectories that end in harming someone in that particular scenario?

safety algebara?

5. VERIFYING THE SAFETY OF ABSTRACT PLANS FOR DISCRETE MANEUVERS

To verify the lane change scenario given in the previous sections, we must integrate two formalisms: timed automata for correctness of the controller's actions (is the scenario's goal achieved?), and reachability for safety of the controller's actions. We have already defined the basic representation of an autonomous driving scenario as a HCHA. A single mode of the HCHA can be viewed from two perspectives: discrete as represented by a state in a timed automaton, and continuous as represented by a plant and systems of nonlinear ODEs. Interaction between these representations occurs when guard conditions in the discrete representation are encountered. One way of reasoning about the goal specifying behavior of the scenario controller is to investigate the safety and performance of the timed automaton using model checking methods.

Thus, we translate the scenarios's HCHA to timed automata. Given the HCHA it is possible to create a map which has abstracts the system to a timed automaton on which we can perform verification tasks.

THEOREM 5.1. *Let X and Y be vector fields on M and N respectively and let $\phi : M \to N$ be a smooth surjective map. Then vector field Y is an abstraction of vector field X with respect to ϕ iff for every integral curve c of X, $\phi \circ c$ is an integral curve of Y. [10]*

If we consider that the abstraction of the system is over-approximate, then we may conclude that it is sufficient to check safety properties on the abstraction [10]. A result that returns safe must be safe. However, given a counterexample to a proposition specifying a safety property one cannot be sure that it is not spurious. Thus, it would also be natural to attempt to refine abstraction to remove any spurious counter examples. We will address this notion of counterexample guided abstraction refinement shortly [2], but first it is essential to discuss the hierarchical aspect of the HCHA.

Hierarchical controls are commonly used to reason about systems which control realistic levels of complexity. Thus, the scenario controller as implemented in the real system will interact with the motion planner. When considering the motion planning controller, reachability is the formalism for which it is convenient to explore the feasibility and safety of the planned trajectories. Reachability analysis determines safety by checking the intersection of reachable sets of the underlying dynamics with unsafe regions. This investigation leads to the fundamental question:*if the discrete controller is safe, and the motion planner is safe, is the hierarchical composition of the two safe?* Fortunately, the answer to this question can be yes.

PROPOSITION 5.1. *Let X_1, X_2, X_3 be vector fields on manifolds M_1, M_2, and M_3 respectively. If X_2 is an abstraction of X_1 with respect to the map $\phi_1 : M_1 \to M_2$ and X_3 is an abstraction of X_2 with respect to map $\phi_2 : M_2 \to M_3$ then X_3 is an abstraction of X_1 with respect to abstracting map $\phi_1 \circ \phi_2$ [10]*

Thus, once each subsystem can be verified, we must map back the verification results to the HCHA formalism.

Going a step beyond the results in [10] We propose that the relationship between verification, in the sense of logical analysis of a system, and control, in the sense of the evolution of state variables described by nonlinear ODEs, is so tightly intertwined that new approaches to system design are necessary. By viewing the activities of correct by construction controller design and verification as tightly coupled processes instead of distinct options we can design solutions for richer, more complex problems. Returning to the lane change case study, we summarize state of the art work in the area of provably safe nonlinear hybrid systems verification and control.

One approach to designing safe lane change and evasive vehicle maneuvers explores the use of reachability to generate an automaton which switches between motion primitives in an offline phase. Online the precomputed reachable sets of motion primitives are compared to the reachable sets of the other agents. It is possible to perform the reachable set calculations online because there is less non-determinism about the initial state of the other agents, there is no tracking controller in the loop, and the dynamics are simpler.[6] Unlike this work, we seek to verify the combination interaction of the target vehicle and other agents using a scenario-based approach entirely offline.

Another approach explores the use of reachability entirely online as a means of verifying and controlling vehicle maneuvers. In order to ease restrictions on computation timing linearized vehicle dynamics are used when considering the target vehicle. Perturbations representing external conditions such as road surface quality and sensor noise are added in order to improve the accuracy of the linearized model.[1] Current run times are on the order of 4-5 seconds making this approach impractical for unexpected events. Furthermore the linearization requires the overapproximation of the system making the results conservative. Our approach does not require linearization of vehicle dynamics. Thus, more complex maneuvers can be considered.

Notions of reachability have also been used within the CEGAR framework introduced by Clarke et al. The CEGAR framework for automatic refinement of system abstractions checks to see whether counterexamples generated during model checking of an abstraction of the system of interest are real by examining whether the counterexample is spurious on a less abstract version of the system. Through this process the abstract version of the system can be iteratively refined.[2]

The CEGAR framework can be extended as an alternative to the online reachability analysis presented by Althoff and Hess. Through the combination of correct by construction controller synthesis with approximate solutions to constrained reachability problems feasible trajectories are computed for nonlinear systems. While the authors present a solution to the motion-planning problem for nonlinear systems, they do not consider other dynamic agents. Furthermore, the authors' decision procedure is not complete. Finally, the presented method of refining the control automaton relies on enumerating and ranking solutions (trajectories) under a cost function which is not well defined in their work.[15]

In our problem we consider several other vehicles in the environment with simple 2-D kinematics and the target vehicle with the nonlinear bicycle model dynamics. While in general the reachability problem for non-linear hybrid systems is undecidable. Recent work has shown that if a user is able to specify a bound on the ODE which is an arbitrary positive rational number δ, then propositions in first order logic over a region *unsafe* can be evaluated [3]. Specifically a system, H, is $safe$ if it cannot reach $unsafe$, and H is $\delta - unsafe$ if H^d can reach $unsafe$.

The unsafe region of a driving scenario as the intersection between the reachable set of the target vehicle and the reachable set of the other vehicles in the scenario over a specified time horizon Δt. We note that using the concept of $\delta - reachability$ does not have any affect on the quality of our results when compared with traditional approaches to reachability. If the system is safe then a logical proof is generated showing that the original system H cannot reach the unsafe region. Alternately if the $\delta - reachability$ analysis returns unsafe then the answer means that an overapproximated version of the system, H^δ reaches the unsafe region. Clearly, one can adjust δ so as to discover the robustness of the system.

In this work we also propose a variation of formal abstraction for the composition of verification results. Clearly the CEGAR framework remains a viable approach for systems which are true model based designs, but in many instances the controller which is designed for the discrete scenario planning level does not stem from the network of HCHA representing the scenario. The controller may be an ad-hoc design, a synthesized design that cannot account for vehicle dynamics due to computational limits and decidability results, or even a purposefully incomplete design used as a sketch for communicating intent between engineering groups.

Lets get these definitions right: Dynamics is a branch of physics (specifically classical mechanics) concerned with the study of forces and torques and their effect on motion, as opposed to kinematics, which studies the motion of objects without reference to its causes.

For brevity we consider a design which has an incomplete representation of the vehicle dynamics. In particular it is faithful to the forward dynamics of the vehicle in 1D space, but has a heavily simplified version of maneuver dynamics and trajectories which occur in 2D space. We place no restrictions on whether these maneuver dynamics represent an under or overabstraction of the HCHA's behavior. What we seek to show is that this ad-hoc controller design or any informal model of the system which is not a direct over or underabstraction of the hybrid system can be proven to interact safely with unmodeled continuous dynamics.

Houssam: add information about sampling-based motion planning techniques.

Matt: add in something about Sarah Loos' work on thm proving

literature review

Pending the lit review, might be worth investigating two approaches we discussed: if the boxes of the partition used by controller are over-approximations of the reach set alphabet, and model checker returns SAT, then must compute reachability for every satisfying trace. Obviously, can get expensive.
Else, we can partition the state space using the reach set alphabet as explained on the board, so the model checker's answer needn't be concretized: by construction it gives a sound and complete answer.
I'd be very surprised if these two don't have antecedents in the literature, they're natural enough...
Because ours is a verification task, it's not up to us to dictate the controller's view of the world.

6. CASE STUDY

6.1 Scenario Description

Lane change manuever scenario occurs on a 2 lane road with the target vehicle and 1 environmental vehicle. The target vehicle must change lanes in order to get into the left turn lane before the end of the scenario.

6.2 Target Vehicle Model

1. Target vehicle has non linear dynamics described by bicycle model.

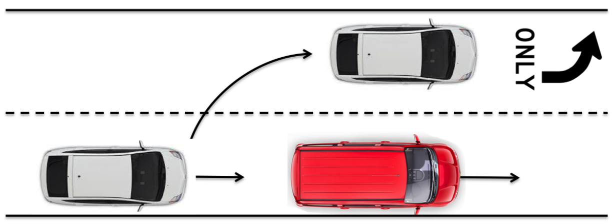

Figure 6: Pictorial description of target vehicle scenario

2. Trajectories are defined for right to left lane switch and left to right lane switch.

3. Hybrid system modes are described by drive forward and lane change.

   (a) Both lane change modes have the same dynamics and controller but implement a different motion primitive (trajectory).

4. Target vehicle has non linear dynamics described by bicycle model:

$$\dot{\beta} = \left(\frac{C_r l_r - C_f l_f}{mv^2}\right)\dot{\psi} + \left(\frac{C_f}{mv}\right)\delta - \left(\frac{C_f + C_r}{mv}\right)\beta + y_\beta \quad (1)$$

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

$$\dot{v} = a_x + y_v \quad (3)$$

$$\dot{s_x} = v\cos\beta + \psi + y_{s_x} \quad (4)$$

$$\dot{s_y} = v\sin\beta + \psi + y_{s_y} \quad (5)$$

$$\dot{\delta} = v_w + y_d \quad (6)$$

   C_f, C_r and l_f, l_r describe respectively the cornering stiffness and distances from the center of gravity to the axles. I_z is the moment of inertia and m is the vehicle mass. 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 logitudinal acceleration, and v_w the rotational speed of the steering angle. The y terms represent disturbances to the system. For example y_β and $y_{\dot{\psi}}$ represent disturbances to the slip angle at the center of mass and the yaw rate.

5. Trajectories are defined for right to left lane switch and left to right lane switch.

6. Hybrid system modes are described by drive forward and lane change.

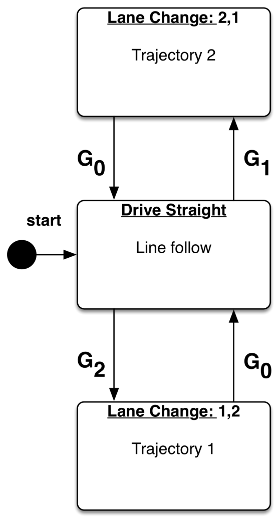

Figure 7: Hybrid automata describing lane change

(a) Lane change is hierarchical and has sub-modes, right to left, straight, and left to right.

(b) Each submode has same dynamics but implements a different motion primitive.

7. Final control equations:

$$\begin{aligned} v_w = k_1(cos(\Psi_d)(s_{y,d} - s_y - w_y) \\ -sin(\Psi_d)(s_{x,d} - s_x - w_x)) \\ +k_2(\Psi_d - \Psi - w_\Psi) \\ +k_3(\dot{\Psi}_d - \dot{\Psi} - w_\psi) \\ -k_4(\delta - w_\delta) \end{aligned} \tag{7}$$

$$\begin{aligned} a_x = k_5(cos(\Psi_d)(s_{x,d} - s_x - w_x) \\ +sin(\Psi_d)(s_{y,d} - s_y - w_y)) \\ +k_6(v_d - v - w_v) \end{aligned} \tag{8}$$

6.3 Environmental Vehicle Model

Environmental vehicle has simple 2D dynamics, no specific controller, non-determinism describes state evolution. When considering other agents (especially vehicles) in a scenario such non-linear dynamics are completely unnecessary. The primary reasoning is that the uncertainty about the state of other vehicles is not do to poor model parameter identification, but rather unknown inputs [2]. The dynamics of the other agents are described by equations (7) and (8) as in [3]:

$$\dot{s_x} = v \cos \beta \tag{9}$$

$$\dot{s_y} = v \sin \beta \tag{10}$$

1. Environmental vehicle must drive forwards.
2. Environmental vehicle must not change lanes if other lane is occupied.
3. At every step the environmental vehicle can apply +/- some specified level of acceleration.

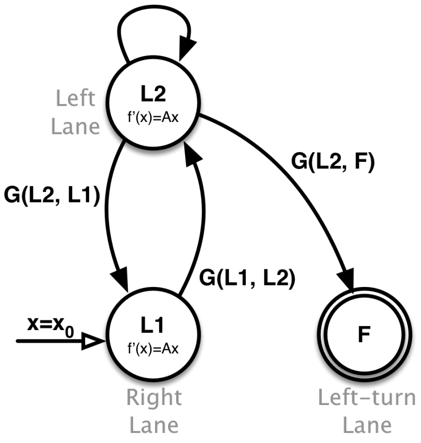

Figure 8: Automata with partial dynamics describing scenario

4. At every step environmental vehicle can switch lanes if condition 2 is not violated.

Environmental vehicle has simple 2D dynamics, no specific controller, non-determinism describes state evolution.

1. Environmental vehicle must drive forwards.
2. Environmental vehicle must not change lanes if other lane is occupied.
3. At every step the environmental vehicle can apply +/- some specified level of acceleration.
4. At every step environmental vehicle can switch lanes if condition 2 is not violated.

6.4 dReal Model

1. 1 or 2 modes depending on lane change or passing. @@ -28,5 +27,4 @@ Passing manuever is scenario on 2 lane road, target vehicle, 1 environmental veh
2. Plant controller is tracking controller which follows trajectory.

6.5 Controller with Partial Dynamics

A finite transition system describing the passing manuever is given in UPPAAL (or NuSMV).

7. CONCLUSIONS

Autonomous plan verification presents two major challenges: a high level of unpredictability of the autonomous system's environment, and system execution that spans many levels of abstraction.

In this paper, we proposed a multi-formalism approach to verification to address these challenges, and illustrated it on a lane change scenario.

We also advocated a scenario verification framework.

In future work, we will automate the modeling and translation procedures, to allow the use of formal plan verification methods by control and system engineers.

We will also address the issue of prioritizing requirements in a given scenario to deal with emergencies.

8. REFERENCES

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

[2] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM (JACM)*, 50(5):752–794, 2003.

[3] S. Gao, S. Kong, W. Chen, and E. Clarke. Delta-complete analysis for bounded reachability of hybrid systems. *arXiv preprint arXiv:1404.7171*, 2014.

[4] R. Goebel and A. Teel. Solutions to hybrid inclusions via set and graphical convergence with stability theory applications. *Automatica*, 42(4):573 – 587, 2006.

[5] T. A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 278–292. IEEE Computer Society Press, 1996.

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

[7] U. Klein and A. Pnueli. Revisiting synthesis of gr(1) specifications. In *Proceedings of the 6th International Conference on Hardware and Software: Verification and Testing*, HVC'10, pages 161–181, Berlin, Heidelberg, 2011. Springer-Verlag.

[8] M. Kloetzer and C. Belta. A fully automated framework for control of linear systems from temporal logic specifications. *IEEE Transactions on Automatic Control*, 53(1):287–297, 2008.

[9] J. Lygeros, K. H. Johansson, S. N. Simic, J. Zhang, and S. Sastry. Dynamical properties of hybrid automata. *IEEE Transactions on Automatic Control*, 48:2–17, 2003.

[10] G. J. Pappas. *Hybrid systems: Computation and abstraction*. PhD thesis, UNIVERSITY of CALIFORNIA at BERKELEY, 1998.

[11] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on principles of programming languages*, pages 179–190. ACM Press, 1989.

[12] V. Raman and H. Kress-Gazit. Explaining impossible high-level robot behaviors. *IEEE Transactions on Robotics*, 29(1):94–104, 2013.

[13] R. G. Sanfelice and A. R. Teel. Dynamical properties of hybrid systems simulators. *Automatica*, 46(2):239–248, 2010.

[14] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, J. Dolan, D. Duggins, D. Ferguson, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. Howard, A. Kelly, D. Kohanbash, M. Likhachev, N. Miller, K. Peterson, R. Rajkumar, P. Rybski, B. Salesky, S. Scherer, Y. Woo-Seo, R. Simmons, S. Singh, J. Snider, A. Stentz, W. Whittaker, J. Ziglar, H. Bae, B. Litkouhi, J. Nickolaou, V. Sadekar, , S. Zeng, J. Struble, M. Taylor, and M. Darms. Tartan racing: A multi-modal approach to the darpa urban challenge. Technical report, Carnegie Mellon University, 2007.

[15] E. Wolff, U. Topcu, and R. Murray. Automaton-guided controller synthesis for nonlinear systems with temporal logic. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 4332–4339, Nov 2013.

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