Verifying Cyber-Physical Interactions in Safety-Critical Systems

Abstract

Safety-compromising bugs in software controlled systems are often also hard to detect. In one of the 2007 DARPA Urban Challenge vehicles such a defect remained hidden in more than 300 miles of test driving and hours of extensive simulations, to manifest for the first time in a particular physical environment during the competition which led to a safety violation and its disqualification. With this incident as an example, here we discuss formalisms and techniques that are now available for safety analysis of cyber-physical systems (CPS). Starting with simulation-based approaches, we turn to more formal approaches, and discuss the emerging area that attempts take advantage of both. We highlight their merits and the limitations, and identify open problems the resolution of which will bolster the development of reliable safety-critical cyber-physical systems.

Keywords

Formal verification, simulation-based verification, autonomous vehicles, invariant checking.

Acknowledgement

This research was partially supported by AFOSR through the MURI program and NSF CNS grant 1016791.

Author names, bios, and mailing addresses

Sayan Mitra is currently an Assistant Professor of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign. He was a CMI postdoctoral scholar at Caltech, received PhD from MIT in 2007, MSc from the Indian Institute of Science, and B.E. from Jadavpur University, Kolkata. He received the National Science Foundation's CAREER award in 2011, AFOSR Young Investigator Research Program Award in 2012, and several best paper awards. His research interests are in formal methods, distributed systems, and cyber-physical systems.

Sayan Mitra
Coordinated Science Laboratory
1304 W. Main St (MC228)
Urbana, IL 61801.
Phone: 217 333 7824 fax: 217 244 5685.

Tichakorn Wongpiromsarn received the B.S. degree in mechanical engineering from Cornell University and the M.S. and Ph.D. degrees in mechanical engineering from California Institute of Technology. She is currently with the Ministry of Science and Technology, Thailand. Her research interests span hybrid systems, distributed control systems, formal methods, transportation networks and situational reasoning and decision making in complex, dynamic and uncertain environments.

Tichakorn Wongpiromsarn
16 Vibhavadi Rangsit Road
Bangkok 10900, Thailand

Richard M. Murray received the B.S. degree in Electrical Engineering from California Institute of Technology in 1985 and the M.S. and Ph.D. degrees in Electrical Engineering and Computer Sciences from the University of California, Berkeley, in 1988 and 1991, respectively. He is currently the Thomas E. and Doris Everhart Professor of Control & Dynamical Systems and Bioengineering at Caltech. Murray's research is in the application of feedback and control to networked systems, with applications in biology and autonomy. Current projects include analysis and design biomolecular feedback circuits; specification, design and synthesis of networked control systems; and novel architectures for control using slow computing.

Richard M. Murray
Control and Dynamical Systems 107-81
California Institute of Technology
1200 E. California Blvd
Pasadena, CA 91125 USA

1. Introduction

Turning left during the third round of the 2007 DARPA Urban Challenge, Alice---an autonomous Ford Econoline van dangerously deviated from the computer-generated path and started stuttering in the middle of a busy intersection. Earlier in the competition, Alice had completed two rounds of missions involving on and off-road driving, parking, merging, and u-turns, while obeying traffic rules---all with style and with no human driver. A sure testament to 15 months of programming, debugging, and test-driving by a team of 50 students and researchers from Caltech, JPL and Northrop Grumman. Alice's onboard hardware included 10 cameras, 8 LADARs, 2 RADARs, an inertial navigation system, 2 Pan-Tilt-Units, 25 CPUs and actuators for the steering, throttle, brake, transmission, and ignition. The software included the sensing and the control systems, with 48 individual programs and more than 100 concurrent threads. The control system (see Figure 1) has modules for making actuation decisions at different spatial and temporal scales based on the processed data streams. The *Mission Planner*, for instance, computes routes for completing high-level missions using a roadmap, the *Traffic Planner* ensures traffic rule conformance based on finite state machines, the *Path Planner* generates waypoints based on inputs from the upper levels and obstacles, and finally, the *Controller* computes acceleration and steering signals for the vehicle to follow the waypoints. For safety, each component was ``unit-tested'' against reasonable-sounding but informal assumptions about the other components and its physical environment.

An unforeseen interaction among the control modules and the physical environment, not witnessed in more than 300 miles of autonomous test driving and hours of extensive simulations, had led to this safety violation and Alice's unfortunate disqualification. What happened? A reactive obstacle avoidance subsystem (ROA) was implemented to rapidly decelerate Alice for collision avoidance. Under normal operation, ROA would send a ``brake'' command to Controller when Alice got too close to an obstacle or when it deviated too much from the planned path. Controller would then rapidly stop Alice and the Path Planner would generate a new path. It turns out that, for protecting the steering system, the interface to the physical hardware limits the rate of steering at low speeds. If the Path Planner produces sharply turning paths at low speeds, then Alice is not able to follow and it deviates. These two effects interacted during the third task which involved making sharp left-turns while merging into traffic. Alice deviated from the path; the ROA activated and slowed it down; Path Planner generated a new path with an even sharper turn to try a successful merge, and this cycle continued taking Alice to the verge of a collision.

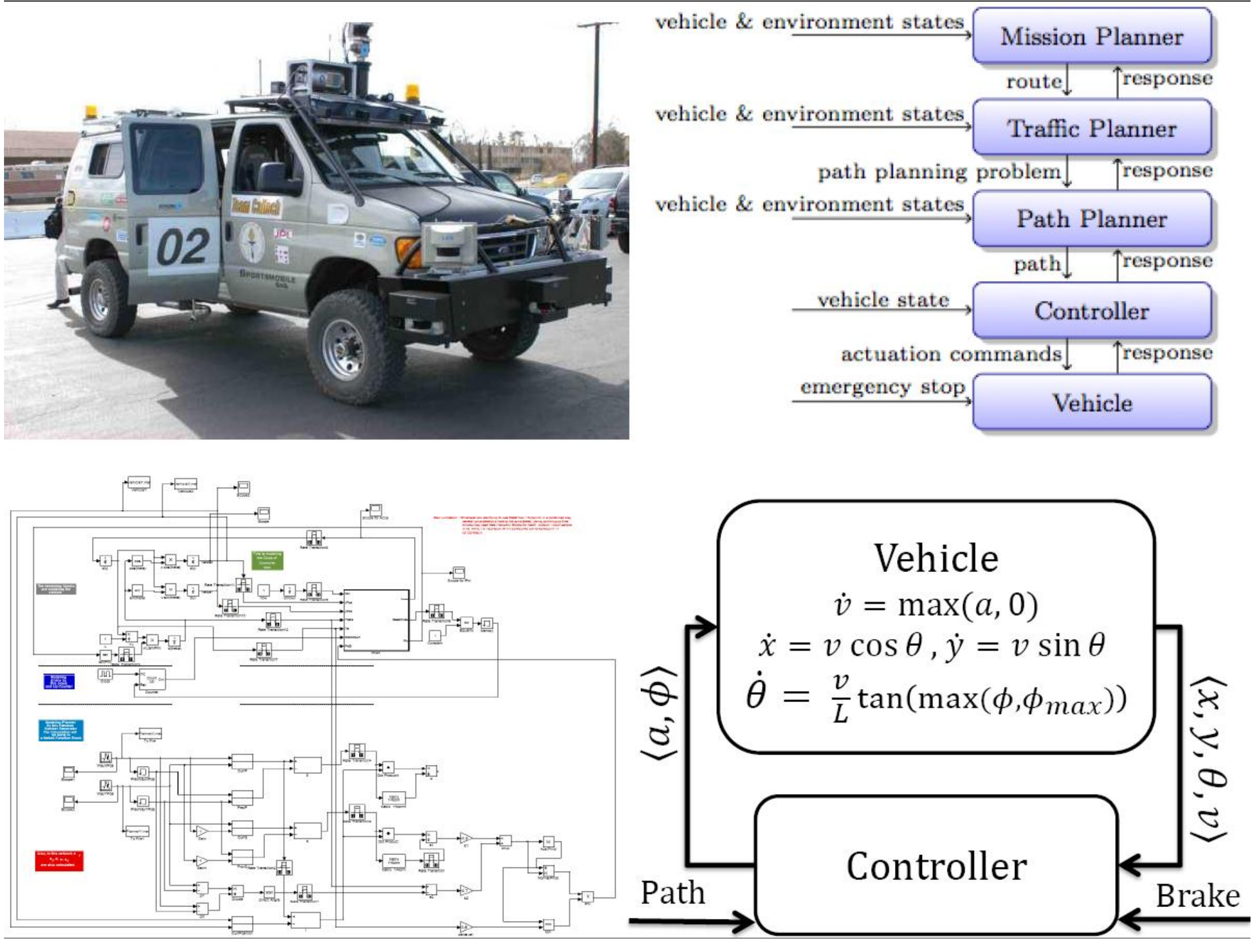
vehicle & environment states
Mission Planner
route
response
vehicle & environment states
Traffic Planner
path planning problem
response
vehicle & environment states
Path Planner
path
response
vehicle state
Controller
actuation commands
response
emergency stop
Vehicle
Vehicle
$\dot{v} = \max(a, 0)$
$\dot{x} = v\cos\theta\,,\, \dot{y} = v\sin\theta$
$\dot{\theta} = \frac{v}{L}\tan(\max(\phi, \phi_{max}))$
$\langle a, \phi \rangle$
$\langle x, y, \theta, v \rangle$
Controller
Path
Brake

Figure 1. Clockwise from top left: (1) Alice, Team Caltech's entry in the 2007 DARPA Urban Challenge. (2) The control protocol stack. (3) Controller-Vehicle automata and their interface with Path planner and Brake controller. (4) A Simulink/Stateflow model.

It is clear that unforeseen interactions between cyber and physical components can wreak havoc in critical systems from individual control systems in vehicles, factories, to networks of systems in air-traffic control and the power grids. The Alice incident illustrates that interaction of software, hardware and physical components (e.g., the ROA and the steering protection) outside of the informally assumed environment is often source of such surprises. One insight from our postmortem analysis of the incident (presented in [18]) suggests adoption of *open modeling*---which forces the designers of a module to precisely pin down all (possibly nondeterministic) behavior of its environment. Then, the component in question is designed and tested in the context of that environment. Analysis based on open models exposes unmodeled or unforeseen interactions. Open models necessarily introduce uncertainties in the form of environmental inputs. This nondeterminism increases the size of the space of possible behaviors one has to consider in the safety analysis, but, fortunately, the state-of-the-art tools from formal methods and control theory are well poised to tackle such models. Based on open models and utilizing these analysis techniques we have been able to diagnose design defects in Alice, air-traffic control protocols, and various other control systems.

In this article, in the context of the analysis of Alice we discuss formalisms and techniques that available for safety analysis of cyber-physical systems (CPS). Starting with simulation-based approaches, we turn to more formal approaches, and discuss the emerging area that attempts take advantage of both. We highlight their merits and the limitations, and identify open problems the resolution of which will bolster the development of safety-critical cyber-physical systems.

2. Modeling Cyber-Physical Systems

2.1 Simulation Models

Tools like LabView, Simulink and Stateflow have made the creation of complex CPS models from simpler building-blocks a standard skill for today's engineering graduates. A Simulink-Stateflow model describes a CPS as a collection of interconnected functional blocks---for example, integrators, switches, and filters, and hierarchical state machines. Figure 1 shows a model of a part of Alice's control system. These models are primarily used for generating simulation traces. Simulink's simulation engine generates runs of a *deterministic* model by numerically integrating the model equations over time with a user-specified method. Though simulation-centric tools are indispensable for rapid-prototyping, design, and debugging, they are limited in providing safety guarantees. Naively running a finite number of simulations cannot eliminate the possibility of safety violations, even for finite state models that run for a long time. The simulation traces record state information only at discrete points in time. Whatever happens in-between requires additional analysis. Nondeterminism---a key feature for capturing uncertainty and underspecified open models---is not supported in these tools either. Finally, even for deterministic systems, numerical errors may cause the simulation points to deviate arbitrarily from the actual states visited by the system. While sophisticated simulation engines vary the sampling period and different numerical techniques for minimizing error in each simulation step, it is tricky to obtain global error bounds, especially in models where the system dynamics changes discontinuously. When Alice's Controller changes the steering input to the vehicle based on a new waypoint, for example, it does so in single discrete steps. Such discontinuous changes are commonplace in CPS models.

Two research directions aim to make simulation tools useful for safety analysis: First, expressive simulation environments have been built from the ground-up, such as Modelica [14] and Ptolemy [12], in which the models have precise semantics, and therefore, can be used for rigorous safety analysis. The second approach recognizes the widespread adoption of Simulink, and aims to obtain safety guarantees from the imperfect simulations. The latter combines formal static analysis of the models with the dynamic, inaccurate information of the simulations and is discussed further in Section 3.

2.2 Formal Models and Properties

Reconciling the differences between the discrete models used for computer hardware and programs and the continuous models used for dynamical systems, the *hybrid systems* community has developed several frameworks for modeling and analyzing CPS. The differences between these frameworks have to do with their origins and the types of analysis they support. The switched system [13] and the hybrid dynamical system [7] frameworks emerged from the control theory and are used for studying stability, robustness, controllability and observability. Computability have not been the focus and behaviors are described in terms of abstract mathematical objects. The Hybrid Input/Output Automaton framework [10] emerged from the distributed computing literature. It supports open models, and has well-developed theories for abstractions and composition. The popular the hybrid automaton framework of Alur, Henzinger et al.[1] was designed for studying automatic verification and decidability questions. Platzer's hybrid dynamic logic supports proof rules for reasoning about systems many identical components [16]. For this article these differences are less important than the shared principles. For concreteness, we consider the Hybrid I/O Automaton (HIOA) framework.

A HIOA is a *nondeterministic* state machine with well-defined input/output variables and actions. Its state can evolve through *discrete transitions*, as in a finite state machine, as well as through continuous *trajectories* which are solutions of differential equations. The behavior of the input variables and actions can be underspecified which enables one to write open models. The framework requires the models to be *input enabled* which essentially forces the modeler to write down how the automaton must react to any inputs from a given class. We describe simple HIOA models of the vehicle and Alice's controller.

Vehicle.

The *state* of an HIOA is described in terms of the valuations a set of *state variables*. For example, the **Vehicle** (see Table 1) automaton's the state variables include the position of the vehicle in the plane (x, y), its heading θ, and its speed v. These variables are declared as outputs as we want them to be accessible to the **Controller**. **Vehicle** also has two input variables: acceleration (a) and steering angle (ϕ). The trajectories of the state variables are defined in terms of differential equations (and inclusions) involving the state as well as the input variables. The key point is that for a given class of input signals for a and ϕ, say, piece-wise continuous signals, the differential equations are guaranteed to have solutions which define trajectories for x, y, v, and θ. This captures the notion that the HIOA model, here **Vehicle**, should be able to react to all possible behaviors of its environment namely the **Controller**. This *input enabling* requirement allows one to define the semantics of, and therefore analyze, *open* hybrid models that have underspecified input variables and transitions.

Controller.

Table 1 shows the **Controller** model which periodically reads the state of the **Vehicle** and computes the acceleration and the steering. The controller also receives interrupts from the *Path planner* and the *Brake controller*. The interrupt information is recorded in state variables, which influence the computation of a and ϕ.

The variables of a **Controller** include the last sampled state information \$$(\bar{x}, \bar{y}, \bar{\theta}, \bar{v})$, the sequence of waypoints provided by a planner *path*, a internal timer clk, information received from the braking unit $brakeReq$, and other internal variables used for computing the controller outputs. The periodic update is modeled by an *urgent* update action. By setting the stopping condition of the trajectories to be ``$clk = \Delta$'', the clk variable is prevented from increasing beyond Δ and the same condition enables the update transition.

When this action occurs the clock is reset to zero and the outputs a and ϕ are computed using some function of the input and the other state variables. The input actions plan and brake captures the interrupts received from the path planner and the brake controller. Since the occurrences of these actions are controlled by the **Controller**'s environment, they can potentially occur at any time. Once again, the **Controller** is an open system and has well-defined behaviors in any environment in which it receives the appropriate input actions and variables.

Table 1. Partial HIOA specifications of the Vehicle and the Controller. The details of the function f are omitted.

Automaton Vehicle(ϕ)	**Automaton Controller**(Δ)
variables	**variables**
output $x, y, \theta, v : \mathbb{R} := \langle x_0, y_0, \theta_0, v_0 \rangle$	**output** $a, \phi : \mathbb{R} := \langle 0,0 \rangle$
input $a, \phi : \mathbb{R}$	**input** $x, y, \theta, v : \mathbb{R}$
	internal $clk : \mathbb{R} := 0, path, brakeReq, \bar{x}, \bar{y}, \bar{\theta}, \bar{v}$
trajectories	
evolve	**transitions**
$\dot{x} = v \cos\theta$ $\dot{y} = v \sin\theta$	**input** plan
$\dot{\theta} = \frac{v}{L} \tan \max(\phi, \phi_{max})$	**pre** true
if $v > 0 \lor a > 0$ **then** $\dot{v} = a$ **else** $\dot{v} = 0$	**eff** \\ update path
	input brake
	pre true
	eff \\ update brakeReq
	internal update
	pre $clk = \Delta$
	eff $clk := 0; \bar{x} := x; \bar{y} := y; \bar{\theta} := \theta; \bar{v} := v;$
	$\langle a, \phi \rangle := f(\langle input\ and\ internal\ vars \rangle)$
	trajectories
	evolve $\dot{clk} = 1$ **stop when** $clk = \Delta$

Open Models and Nondeterminism.

The HIOA framework supports modeling of open systems. Uncertainties and underspecification in the external environment and within the automaton itself are modeled as nondeterministic choices. Indeed, there are sources of nondeterminism in a hybrid model that do not exist in purely discrete models. First of all, the input transitions and trajectories are exogenous to the model, but the model should have well-defined behavior for any input from a reasonable class. Component failures and noise are often modeled as input transitions and variables as the automaton has very little control over them. Secondly, even without input variables, the differential equations and inequalities may permit multiple solutions over time. For example, a standard way of modeling a clock *clk* that is drifting with some bounded rate $\pm\rho$ is to write the differential inclusion $(1-\rho) \leq \dot{clk} \leq (1+\rho)$. This implies that along any solution of the system, starting from $clk = 0$, the value of clk after time *t* is a point in the interval $[clk_0 + (1-\rho)\,t, clk_0 + (1+\rho)\,t]$. Secondly, at a given state of an automaton there may be a choice between several discrete transitions or for some amount of time to elapse. This is exploited for modeling transitions with some uncertainty in the timing behavior. For example, relaxing the precondition of the update action for the **Controller** to $clk \in [\Delta - \epsilon, \Delta]$ we could model a slightly less stringent time-triggered controller which updates its output once every $[\Delta - \epsilon, \Delta]$ time. In addition, as in the case of discrete models, there may be nondeterministic choices among multiple start states and multiple enabled actions in the initial state.

Semantics: Executions, Reachability, Invariants, Safety, and Robustness

We have introduced hybrid models through examples of Alice's subsystems. Now we introduce some concepts related to semantics of such models which will prepare us for the discussion of safety analysis techniques in Section 3.

The semantics of a state machine-based modeling framework, such as HIOA, is defined in terms of *states* and *executions*. A state of the system is defined by valuations of all the variables in the model. *Predicates* or constraints on the variables can be interpreted as sets of states. For example, the semantic interpretation of the predicate: $Deviation(\epsilon) = dist(x, y, path) \leq \epsilon$ is the set of states of the system such that the position is within ϵ distance (measured by some metric $dist$) of the path.

An execution of a HIOA is an alternating sequence of its transitions and trajectories. Figure 2 shows the plot of an execution with the backdrop of a left turn. The red-dots indicate the occurrence of brake action (triggered by the ROA) which then results in a new path (generated by the path planner), and a new controller output. The blue lines indicate the projection of the intervening vehicle trajectories on x and y.

As a hybrid automaton is nondeterministic, it has a set of executions which capture all its behaviors. A state is said to be *reachable* if there exists an execution which terminates at that state. The set of all such states, $Reach_A$, is called the *reach set*. A *safety property S* is specified by a predicates on variables. We could assert that *Deviation(2 m)* is a safety requirement, meaning that the reach set of the system is contained in the set *Deviation(2 m)*.

An over-approximation of $Reach_A$ is called an *invariant set*. Invariants can provide useful semantic information about a system. For programs for instance, states outside an invariant are never reached. This leads to a useful approach for proving safety: find or compute an invariant set that is disjoint from the given unsafe set (complement of the safety property).

Reachability and invariance have their bounded-time counterparts. A state that is reachable within a time horizon of T is *T-reachable* and so on. We will say A is ϵ-safe with respect to a set *S* within time *T*, if all states within distance ϵ from some state reachable within time T are within *S*. A is *robustly safe*, if it is ϵ-safe for some ϵ. Often, the verification approaches are designed to establish robust safety.

Abstractions.

A key benefit of formal modeling is that it enables one to precisely define *abstractions*. Semantically, a hybrid automaton B is an abstraction of another automaton A if every execution[1] of A is also an execution of B. An abstraction B over-approximates the behaviors A, and therefore, if B is constructed to be more tractable than A then by proving its safety we can infer safety of A. The HIOA framework provides rules for establishing abstraction relationships between open models. However, even though

[1] In general, only the visible parts of the executions have to match-up, which provides more flexibility in defining what is important.

this definition of abstraction in terms of the containment of executions, is useful for reasoning, it does not lead to algorithms for computing the abstraction B. Instead, one computes some sort of a *simulation relations* which encodes the abstraction. A forward simulation relation from A to B relates the variables of A and B such that for every transition and trajectory of A, there exists a corresponding transitions (or trajectory) of B which reserves the relation. Various different notions of forward and backward simulation relations have been developed for hybrid modeling frameworks. The importance of abstractions and simulation relations in automated safety analysis will become evident in the next Section.

3. Safety Verification Tools and Techniques

In this section, we discuss three approaches that are available today for rigorous correctness analysis or verification of safety-critical systems like Alice. Two classes of properties capture most of the common correctness requirements in systems, namely, safety and progress. Here we focus on the more extensively studied problem of safety verification and refer the interested reader to [5] for some recent results on verification of progress and stability. The tools and algorithms underlying these verification approaches have also been applied to synthesize controller code that is correct by construction; for an overview of this growing area see [2] and [11].

Safety or Invariance: Reach set of A is contained in some safe set S or A never goes outside S.	**Example 1.**The **Vehicle-Controller** system never deviates more than 1m from the straight line joining the successive waypoints (for any behavior of *planner* and *brake controller)*. **Example 2.** No two aircrafts in an air-traffic management system ever come within d distance.
Progress: For sets of states \$I\$ and \$F\$, every execution starting from \$I\$ reaches \$F\$ within bounded time.	**Example 1.** Starting near a waypoint the **Vehicle** reaches the next waypoint within T time, provided there are no incessant brake requests; T depends current state, the waypoints, and the issued brake requests. **Example 2.** Starting from safe states, every aircraft *eventually* gets clearance to land.

3.1 Automatic Reach Set Over-approximations

The automatic procedure for proving that a hybrid model A is safe with respect to some unsafe set U involves (a) computing the (unbounded) reach set $Reach_A$ of A and (b) checking that $Reach_A$ and U are disjoint. If $Reach_A$ is computable then this procedure not only proves safety, but also whenever the sets are not disjoint, then the procedure finds bugs by giving executions that lead to U.

Classes of *decidable* hybrid automata have been identified for which $Reach_A$ can be computed exactly. These include hybrid automata with only clocks, automata with variables that only evolve at constant bounded rates and are reset whenever the rates change, and so called o-minimal and STORMED hybrid models. Other decidable classes are obtained when the number of continuous variables is small. The core idea in all of these results is a bisimulation argument which constructs a finite state machine (FSM) which can exactly mimic the transitions and the trajectories of the hybrid automaton A being analyzed. The reach set of this bisimilar FSM can then be computed by exploring its control graph to obtain $Reach_A$. There is a distinguished history of software tools like UPPAAL, HyTech, d/dt, Checkmate, PHAVer, and the more recent SpaceEx which embody these reachability algorithms and have been applied to successfully analyze the safety of a variety of CPS models [8],[6]. In each step along the way, new insights about the data-structures (e.g., polyhedra, rectangles, ellipsoids) used for representing the partially computed reach sets have played an important role in improving the efficiency of the algorithms.

For general CPS though, decidable hybrid automata models are few and far between. For most models with more than a small number of continuous variables and linear or nonlinear dynamics, computing $Reach_A$ exactly is undecidable. Even for a decidable automaton, obtaining scalable exact reachability algorithms is a big challenge. The **Vehicle**-**Controller** model described above, for instance, would require stretching the capabilities of current reachability tools. To address this, the reachability problem is relaxed to compute over approximations of $Reach_A$ through abstractions.

A specific type of abstraction is proposed by the *hybridization* scheme of [3] in which the complicated nonlinear dynamics of A is approximated by piece-wise linear or piece-wise rectangular dynamics in B. The state space of A is partitioned into regions, and within each region, the complex nonlinear vector field is over-approximated by a set-valued rectangular (or linear) vector field.

Hybridization is one of several different methods for constructing property-agnostic abstractions. The partitioning and the over-approximation described above does not rely on the unsafe set *U*. In contrast, *Counter-example guided abstraction refinement (CEGAR)* algorithms which have been successful in software verification, constructs abstractions for proving or disproving a specific safety property.

CEGAR starts with a coarse abstraction B_0 of A; in each iteration, $Reach_{B_i}$ is computed and checked for safety. If B_i is safe then A is safe and the algorithm terminates. Otherwise $B_i$is unsafe and a counter-example α is produced (provided the $Reach_{B_i}$'s can be computed). If α corresponds to a valid execution of A then one can immediately infer that A is unsafe. This is called the validation step. Otherwise, α is a spurious counter-example arising from the overapproximation in B_i and it is used to obtain a new *refined* abstraction B_{i+1}, and the procedure continues. If a CEGAR algorithm terminates with ``safe'' or ``unsafe'', the answer is correct, but it is guaranteed to terminate only if (a) the abstract automata are from a decidable class, (b) the validation step is computable, and (c) the refinement step eliminates enough spurious counter-examples to make progress towards a decision. The existing CEGAR algorithms for restricted classes of hybrid models show promise of improved scalability, however, termination guarantees are likely to be achievable only under additional robustness assumptions. For instance, termination may be guaranteed if the automaton A is not only safe with respect to *U* but is also robustly safe. When A is safe but not robustly safe, the algorithm may not terminate. Development of property-directed abstraction-refinement schemes for verification of nonlinear hybrid models remains an open problem.

3.2 Finding Inductive Invariants with Human Guidance

An alternative to automatically computing over-approximations of $Reach_A$ or invariants of the given system model, is to find *inductive invariants*. An invariant I is inductive for the system A if (a) all start states of A satisfy I, and (b) starting from a state that satisfies I and following a transition or a trajectory, A continues to satisfy I. Given a predicate I, checking whether it is an inductive invariant is typically an easier problem than computing the reach set. For the discrete transitions, this involves symbolically executing the model for all possible transitions that are enabled from I. For the continuous trajectory, this can be achieved by checking that the vector field defining the evolution of A ``points inwards'' at the boundary of I. This so called *sub-tangential* condition can be checked symbolically without necessarily solving the differential equations.

This is the approach we used in [18] for establishing the key safety properties of a corrected model of Alice. Often the initial guess I has to be strengthened by introducing constraints on the state variables. For example, for establishing that the deviation of the **Vehicle**-**Controller** model from the planned path is upper-bounded by some e_{max}, we also have to introduce bounds on the disorientation of **Vehicle** with respect to the direction of the path. The inertia of the vehicle, the limits on its maneuverability, and the controller periodicity require that for the deviation to be bounded the vehicle cannot be highly disoriented. Specifically, we defined a collection of predicates I_k (for each natural number *k*) that incorporates the following constraints:

1. The deviation from the path is bounded by a constant ϵ_k, for a constant $\epsilon_k \leq \epsilon_{max}$.
2. The disorientation is small enough such that the steering angle computed by the controller (as a function of the deviation and disorientation) is in the range $[-\phi_k, \phi_k]$ for a constant ϕ_k that does not exceed the physical limit of the vehicle steering capacity.
3. The speed is bounded by a constant v_{max} .

The collection of predicates I_k projected onto the deviation-disorientation plane is shown in Figure 2. Then, we showed that, provided (a) that the brakes are not triggered ``too frequently'', (b) that the turns in the path are not ``too sharp'' compared to the length of the path and (c) that the execution speed of the controller is not ``too slow'' with respect to v_{max}, each I_k is in fact an inductive invariant of the **Vehicle**-**Controller** system. This establishes the bounded deviation (safety) property. The analysis makes the notion of too frequent, too sharp and too slow precise in terms of the vehicle and controller parameters. Checking inductive invariance of the I_k 's using the sub-tangential condition is a routine exercise once the correct set of assumptions are in place. As an added bonus, here, the invariants also aided the progress analysis. During each control period, as the vehicle makes progress towards the next waypoint, we show that under the above assumptions, it moves from I_k to I_{k+1} that is contained in I_k. That is, in executing a long segment, the vehicle converges to a small deviation and disorientation with respect to the path, and therefore, the instruction for executing a subsequent sharp turn does not make the deviation and disorientation grow too much.

Towards Automatic Search for Inductive Invariants

The above approach is applicable to a general class of open CPS models. Also, when successful it provides useful information about the system's behavior apart from just establishing safety. For instance, by checking inductive invariants for the open system---the vehicle and the controller---we derived restrictions on its environment, namely, the brake controller and the path planner, which were sufficient for safety. The main limitation is the first guess. Finding useful inductive invariants requires creativity and insights about the behavior of the system.

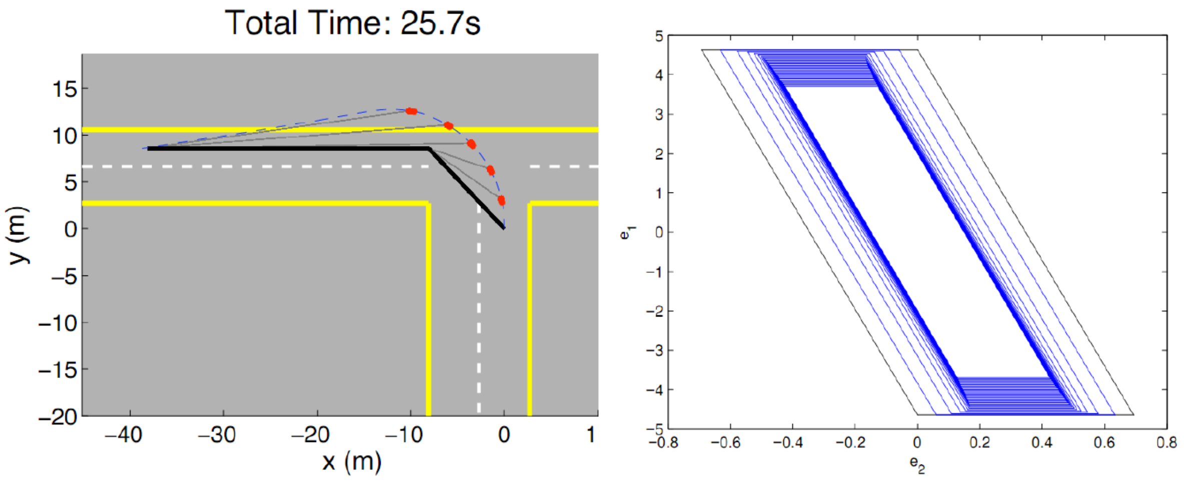

Figure 2. Left: An execution of a Vehicle-Controller system with input $\act{brake}$ transitions. Right: The invariant sets for different values of deviation and disorientation.

In addressing this issue, Sankaranarayanan et al. [17] have proposed several methods for finding inductive invariants of hybrid models automatically. In these approaches, the insights are encoded in the form of *invariant templates*. These templates define the possible shape of the invariants and have to be instantiated to find the actual invariants from that shape class. In [17], for example, the search for inductive invariants is seeded by a family of, in this case linear, template constraints $F(\bar{c},\bar{x}) = \Sigma_i c_i x_i \leq d$, where $\bar{c}$ and d are parameters to be synthesized. Then the boundary of the invariant set is given by the set of points $\bar{x}$ where $\Sigma_i c_i x_i = d$ and the sub-tangential condition can be encoded as the constraint:

$$\forall\ \bar{x}\ \left[\Sigma_i c_i x_i\ =\ d \Rightarrow \frac{d(\Sigma_i c_i x_i)}{dt}\ <\ 0\right]$$

In this linear case, the Lagrange relaxation of the resulting linear programs can be solved for automatically finding linear invariants and in some cases even the strongest linear invariants. Several interesting open problem arise in this area in identifying subclasses of nonlinear models and more general templates for which the resulting set of constraints can be solved effectively to yield a richer variety of inductive invariants.

3.3 Reaping Benefits of Simulations for Verification

The final verification approach we will discuss takes us back to simulation. As mentioned before, simulation models are widely used and (possibly inaccurate) simulation traces are inexpensive to generate in a pragmatic sense. How can we overcome the semantic imprecisions and ambiguities of these simulation models, and use finite but possibly large number of simulations traces to obtain formal safety guarantees? The ``proofs from tests'' or ``verification from simulations'' paradigm has produced success stories in the hardware and software verification and now it is being developed for CPS [15],[19],[4],[9]. The key idea is to compute from an individual simulation trace (or a test) that starts from a single start state $\boldsymbol{x_0}$, a tube which contains all the executions of the system starting from an open ball around $\boldsymbol{x_0}$. Since this ball has non-zero radius, from a finite number of simulation traces it is possible to compute tubes that over-approximate all executions from a set of initial states, and therefore, to verify safety from all initial states.

The success of this approach for CPS hinges on being able to compute these tubes for hybrid models from simulation traces that only record discrete snapshots. The gaps in the trace have to be filled for constructing the tubes that are guaranteed to contain the executions. Further, as mentioned in Section 2.1, the recorded traces from a state $\boldsymbol{x_0}$ may have arbitrarily large errors compared to the actual states that are visited in the execution starting from $\boldsymbol{x_0}$**.** In [4],[9] it is shown how, for deterministic models, control theoretic properties of the models like stability, contractiveness, and continuity, can be used for computing these tubes. For example, if the trajectories of the system are Lipchitz continuous or asymptotically stable, then the tube containing all executions starting from a ball around $\boldsymbol{x_0}$ can be computed in a straightforward manner. It is worth remarking that checking asymptotic stability of a dynamical system is in general not an easy problem, however, it is reasonable to expect that the simulation models are annotated with certificates that assure these control theoretic properties (e.g., Lyapunov functions, Lipchitz constants, contraction metrics etc.) hold. As long as the simulation model meets the annotations, for bounded time analysis, the soundness and the completeness of the procedure can be guaranteed, even if the exact formal semantics of the model is not known.

Consider a *deterministic* hybrid automaton A with a set of initial states Q_0, and an unsafe region *U*. Using the above approach for computing tubes, we can check whether A is robustly safe with respect to *U* up to a time bound *T* by simulating A from *finitely* many initial states. The stability properties ensure that for every $\epsilon > 0$ there is a computable $\delta > 0$ such that for any two executions starting within a δ-ball remain within an ϵ-ball up to time *T*. If *U* is an *open* set of unsafe states, Q_0 a bounded set of initial states, then under certain mild assumptions about the continuity of the executions of A it can be shown that the the robust safety problem can be decided by simulating A from finitely many initial states. This approach has been explored in several papers and has been implemented in the Breach [4] and C2E2[9] tools. It offers a significant advantage over the reachability based approaches discussed in Section 3.1. First, those methods can theoretically only work for systems whose continuous *flows* are described by polynomials, and even then is unlikely to scale to large systems because of the need to reason about non-linear arithmetic. Finally, since simulation engines can work with systems with complicated non-linear dynamics, and the subsequent analysis of simulation traces does not concern itself with the actual dynamics, this could potentially handle systems that cannot be handled by today's model checkers.

Though this line of research is still in its early stages, but, the preliminary experimental results are very promising and handily outperform reachability-based tools in natural examples. Development of a simulation-based verification framework for *nondeterministic models*, possibly including actual controller code, remains an open problem. In particular, for distributed systems with message delays, clock skews, and scheduling uncertainties, handling nondeterminism poses major challenges in generating traces, in identifying useful annotations, as well as in developing effective safety verification algorithms.

4. Concluding Remarks

Our experiences with safety analysis of Alice and several other real world CPS models strongly suggest that component-level verification alone is not sufficient to guarantee safety. Open and formal modeling and verification should be applied to identify dangerous and unpredictable component interactions. The current state of verification tools provide several options for analyzing different kinds of models with different levels of investment. For systems with linear dynamics and a dozen or so continuous variables, completely automatic, safety analysis is becoming feasible on a modest verification budget. For more general models, mechanical verification guided by human inputs (e.g., in the form of inductive invariant templates) is possible with a larger budget. Finally, widely adopted simulation models can be leveraged for obtaining bounded time safety guarantees for high-dimensional nonlinear models. While social, legal, and economic barriers remain for widespread adoption, we believe that computer-aided safety analysis tools are ready for entering engineer's inventory.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. Theoretical Computer Science}, 138(1):3--34, 1995.

[2] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G. J. Pappas. Symbolic planning and control of robot motion. *IEEE Robotics and Automation Magazine*, 14(1):61--70, 2007.

[3] T. Dang, O. Maler, and R. Testylier. Accurate hybridization of nonlinear systems. In {\em HSCC}, pages 11--20, 2010.

[4] A. Donz{\'e}. Breach, a toolbox for verification and parameter synthesis of hybrid systems. *Proceedings of the 22nd international conference on Computer Aided Verification*, CAV'10, pages 167--170, Berlin, Heidelberg, 2010. Springer-Verlag.

[5] P. S. Duggirala and S. Mitra. Lyapunov abstractions for inevitability of hybrid systems. In *Hybrid Systems: Computation and Control*, pages 115--124. ACM, 2012.

[6] G. Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In M. Morari and L. Thiele, editors, {\em HSCC}, volume 3414 of *LNCS*, pages 258--273. Springer, 2005.

[7] R. Goebel, R. G. Sanfelice, and A. R. Teel. Hybrid Dynamical Systems:Modeling, Stability, and Robustness. Princeton University Press, 2012.

[8] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In {\em Computer Aided Verification (CAV '97)}, volume 1254 of {\em LNCS}, pages 460--483, 1997.

[9] Z. Huang and S. Mitra. Computing bounded reach sets from sampled simulation traces. In Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control, HSCC '12, pages 291--294, New York, NY, USA, 2012. ACM. New version of tool available from: http://publish.illinois.edu/c2e2-tool/main-2/

[10] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. The Theory of Timed {I/O} Automata. Synthesis Lectures on Computer Science. Morgan Claypool, November 2005. Also available as Technical Report MIT-LCS-TR-917.

[11] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal-logic-based reactive mission and motion planning. {\em IEEE Transactions on Robotics}, 25(6):1370--1381, 2009.

[12] E. A. Lee, C. Hylands, J. Janneck, J. Davis II, J. Liu, X. Liu, S. Neuendorffer, S. S. M. Stewart, K. Vissers, and P. Whitaker. Overview of the ptolemy project. Technical Report UCB/ERL M01/11, EECS Department, University of California, Berkeley, 2001.

[13] D. Liberzon. {\em Switching in Systems and Control}. Systems and Control: Foundations and Applications. Birkhauser, Boston, June 2003.

[14] S. Mattsson, M. Otter, and H. Elmqvist. Modelica hybrid modeling and efficient simulation. In {\em Proceedings of the 38th IEEE Conference on Decision and Control}, pages 3502--3507, Phoenix, Arizona, December 1999.

[15] T. Nghiem, S. Sankaranarayanan, G. Fainekos, F. Ivanci\'{c}, A. Gupta, and G. J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In {\em Proceedings of the 13th ACM international conference on Hybrid systems: computation and control}, HSCC '10, pages 211--220, New York, NY, USA, 2010. ACM.

[16] A. Platzer. Differential dynamic logic for hybrid systems. {\em J. Autom. Reasoning}, 41(2):143--189, 2008.

[17] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Constructing invariants for hybrid systems. {\em Formal Methods in System Design}, 32(1):25--55, 2008.

[18] T. Wongpiromsarn, S. Mitra, A. G. Lamperski, and R. M. Murray. Verification of periodically controlled hybrid systems: Application to an autonomous vehicle. {\em ACM Trans. Embedded Comput. Syst.}, 11(S2):53, 2012.

[19] P. Zuliani, A. Platzer, and E. M. Clarke. Bayesian statistical model checking with application to simulink/stateflow verification. In {\em Proceedings of the 13th ACM international conference on Hybrid systems: computation and control}, HSCC '10, pages 243--252, New York, NY, USA, 2010. ACM.