Safety in Autonomous Driving: Can Tools Offer Guarantees?

Daniel J. Fremont
University of California, Santa Cruz
dfremont@ucsc.edu

Alberto L. Sangiovanni-Vincentelli
University of California, Berkeley
alberto@eecs.berkeley.edu

Sanjit A. Seshia
University of California, Berkeley
sseshia@eecs.berkeley.edu

***Abstract*—Persistent challenges in making autonomous vehicles safe and reliable have hampered their widespread deployment. We believe that formal methods will play an essential role in the enterprise of ensuring AV safety by providing tools for the modeling, verification, synthesis, and runtime assurance of AV systems. In this paper, we outline the progress we and others have made towards this goal, and the challenges that remain.**

***Index Terms*—Autonomous vehicles, formal verification, design methodology, cyber-physical systems, machine learning.**

I. Introduction

A decade ago, the development of autonomous vehicles (AVs) and automated driving systems (ADS) seemed to be progressing very rapidly, and predictions were made of fully self-driving cars being widely deployed by the year 2020. That vision has now been tempered by the challenges of developing AVs and ADS, none more important than that of ensuring their safe operation. An important characteristic of AVs and ADSs is the expanding use of machine learning (ML) and other artificial intelligence (AI) based components in them. ML components, such as deep neural networks (DNNs), have proved to be fairly effective at perceptual tasks, such as object detection, classification, and image segmentation, as well as for prediction of agent behaviors. However, ML components can be easily fooled by so-called adversarial examples [1], and there have also been well-documented failures of AVs in the real world for which the evidence points to a failure (in part) of ML-based perception [2], [3]. The current crash/disengagement rate per mile for AVs remains much higher than for human drivers [4], [5].

It is clear that the development process for AVs and ADSs needs to be improved, but this requires several challenges to be overcome. First, AVs have all the complexities of traditional automotive systems, being distributed, highly interconnected cyber-physical systems (CPS) that integrate hundreds of heterogeneous components, contain hundreds of millions of lines of software, and involve networking, both on board and externally. Second, they operate in complex environments that present a huge variety of possible traffic scenarios, with edge cases that cannot be exhaustively tested and involving dynamic and unpredictable agents including humans and human-operated vehicles. Third, this complex environment is sensed through numerous modalities, and, as observed above, there is extensive use of AI and ML techniques to interpret sensor data and predict the actions of agents. These ML components tend to be large black-box models for perceptual tasks for which it is very difficult, if not impossible, to verify whether they can ensure safe behaviors when deployed in real life.[1]

It is our opinion that safety of autonomous vehicles should be *guaranteed* against reasonably complete scenarios. To do so, we need to develop tools that can formally analyze the behavior of ML-based decision making in the context of autonomous vehicles and design for safety. Formal analysis cannot be performed without a mathematically-sound set of specifications and system characterization including the environment in which the system operates.

In this paper, we advocate the use of formal methods and software tools to address the safety challenges described above. We discuss four important areas:

- formalisms for modeling the environments of AVs and capturing the specifications and contracts they must satisfy (Sec. II).
- novel and scalable approaches to the verification and testing of AVs and ADS (Sec. III).
- tools and techniques for synthesis of AV components, control strategies, etc., including synthesis of models, data, controllers, and implementations (Sec. IV).
- techniques for runtime assurance, including runtime monitors, minimal risk maneuvers, and techniques for control in semi-autonomous systems (Sec. V).

We propose research directions to go beyond the state of the art in each area, as well as cross-cutting concerns (Sec. VI).

II. Formal Modeling and Specification

Any formal analysis must start with defining models of the system, its environment, and its specifications. Modeling of AVs presents several challenges due to the complexity of their environments, the necessity of *quantitative* metrics for evaluating their safety, and the complex interactions between their numerous software and hardware components.

A. Environment Modeling Languages

Even a single sensor of an AV, such as a camera whose images are fed into a DNN for object detection, can yield

This work was supported in part by NSF grants CNS-1545126 (VeHICaL) and CNS-1739816, DARPA contracts FA8750-18-C-0101 and FA8750-20-C-0156, and by the iCyPhy center and Berkeley Deep Drive.

[1] A more detailed discussion of the challenges of designing robust and verified ML/AI-based systems can be found in [6].

```
ego = EgoCar at 38.6 @ 183.9,
    with behavior DriveTo(40 @ 225.2)
ped = Pedestrian at 19.782 @ 225.680,
    facing 90 deg relative to roadDirection,
    with behavior Hesitate,
    with startDelay Range(7, 15),
    with walkDistance Range(4, 7),
    with hesitateTime Range(1, 3)
```

Fig. 1. A partial Scenic program for a hesitating pedestrian scenario. [10]

an input space with millions of dimensions. To mitigate this complexity, we have proposed reasoning with higher-level, *semantic* features such as the positions of objects [7].

However, even after moving to a semantic feature space, environment modeling is still difficult due to the diversity and complex structure of AV operating conditions. For example, generating a test by placing cars randomly on a road would yield highly unrealistic or even physically impossible configurations. Real traffic scenarios have complex geometric structure: e.g., people typically drive in lanes, with several distinct types of maneuvers. Moreover, it is essential to be able to test or analyze the system in a particular subspace of interest: for example, an AV may only be intended to operate under certain driving conditions; alternatively, we may want to test the system on variants of a real-world failure case to validate a candidate fix for the problem. In general, we need techniques that allow designers to target test generation (and formal analysis) towards cases of interest, similarly to constrained-random verification in electronic design automation or generative fuzz testing in software development.

We have tackled this challenge by developing Scenic, a domain-specific probabilistic programming language for modeling the environments of AVs and other CPS [8], [9]. A Scenic program defines a distribution over configurations of objects and agents in space, as well as their behaviors over time. Our tools can sample from this distribution to obtain concrete scenarios which can be executed in a simulator to test (or train) an AV system. Scenic provides convenient constructs for geometry, as well as the ability to add constraints declaratively, which make it possible to define complex scenarios in a concise, readable way: for example, Fig. 1 shows part of a Scenic program from one of our case studies, where a pedestrian crosses the street in front of the AV, hesitating along the way. Scenic can be used for precise environment modeling both for simulation-based testing and also for formal verification, playing a similar role to languages such as Verilog.

B. Properties and Contracts

Traditionally, in formal methods, specifications are Boolean, evaluating to true or false for a given behavior. However, in AI-based autonomy, it can be useful to develop hybrid Boolean-quantitative specifications, which blend logical specifications with objective functions that quantify risk [6]. Additionally, autonomous vehicles often have multiple objectives of varying importance, necessitating a way to organize those objectives in a systematic fashion for use in verification and control, such as the notion of *rulebooks* [11] where properties and metrics can be organized in a hierarchy, compared, and aggregated.

Contracts are mathematical objects that offer rigorous support for modular and hierarchical system reasoning [12], [13]. A contract represents a design specification split into two parts: assumptions expected from the environment, and responsibilities assigned to the design element provided its environment satisfies the assumptions. While the basic theory of contracts is well established, better tools and specification languages are needed to implement this theory, particularly for autonomous systems. Additionally, there is a need to extend the theory to richer classes of properties, particularly *hyperproperties* that go beyond standard safety and liveness properties to capture quantitative, aggregate specifications, including robustness of systems and ML components (see, e.g., [14]).

III. Verification and Testing

To scale verification to entire AVs, while integrating automated analyses and experts' domain knowledge, we have focused on a paradigm of *formally-guided simulation*, implemented in the open-source VerifAI toolkit [15]. VerifAI's basic functionality is to falsify closed-loop CPS, that is, to find environments under which they violate their specifications. To start, the user defines an *abstract feature space* parametrizing the environments and system configurations of interest; defining this space using a Scenic program enables placing a prior distribution on the space to guide initial tests. VerifAI then generates test cases by searching this space using a variety of sampling and optimization algorithms. For each test case, we simulate the system and monitor to what extent its specifications are satisfied or violated. This data is used to guide subsequent tests, and to analyze where failures occur.

We have successfully used VerifAI to find unsafe behaviors of an actual AV software stack, both in simulation and in reality [10]. We modeled a pedestrian scenario in Scenic as described above, and specified safety properties for the AV in MTL. VerifAI then simulated over 1,000 tests sampled from the scenario, finding several dozen safety violations. Visualizing the data logged by VerifAI, we identified regions of the search space which were robustly safe, robustly unsafe, or marginal, and selected representative tests from several such regions. We then implemented these tests on a test track with an actual AV and a robotic pedestrian dummy. We found good qualitative agreement between the AV's behavior in simulation and on the track, obtaining several unsafe runs and one actual collision. These results demonstrate that formally-guided simulation can be effective in finding undesirable behaviors of AVs in the real world.

Guaranteeing safety will require a broader range of verification techniques for CPS and DNNs [16], [17]; however, existing methods do not scale to systems with complex ML perception components. We believe that addressing this challenge will require compositional reasoning [18], with contracts and abstractions for ML components as mentioned above.

IV. Synthesis

Tools for correct-by-construction synthesis have had a big impact on the design of certain classes of systems, most notably of digital circuits through the use of logic synthesis and equivalence checking in the RTL design flow. Can we develop a similar toolset for autonomous vehicles?

While this is an important direction for research, it is also an extremely challenging direction due to the complexity and heterogeneity of AVs and their components. Compositional techniques for synthesis are essential for synthesis to scale, and this can build upon the theory of contracts [12] discussed in the earlier section, using, e.g., the quotient operation to find "missing" components [19]. There are two promising directions in particular we would like to highlight: (i) *oracle-guided inductive synthesis*, discussed below, and (ii) *semi-autonomous control*, which involves synthesizing controllers for autonomous systems that work in concert with humans inside and outside the vehicle. For lack of space, we refer the reader to previous articles on the second topic [6].

Oracle-Guided Inductive Synthesis: As noted earlier, the use of ML is widespread in AV components; however, this ML is based on a standard supervised/unsupervised learning model, where the generalizability of the learned model is tested using a held-out test set. Such ML components do not have the kind of rigorous correct-by-construction properties desired for safety-critical systems. Can we design ML components and systems that verifiably satisfy a formal specification?

To achieve this, we suggest to use the paradigm of *oracle-guided inductive synthesis* (OGIS), which is also termed *oracle-guided learning* [20], [21]. In OGIS, the learning algorithm interacts with an oracle, such as a verifier, that provides richer inputs than simply labeled/unlabeled data. A particularly effective instance of OGIS is *counterexample-guided inductive synthesis* (CEGIS). Specifically, the data set used for training and testing ML components can be augmented in a counterexample-guided fashion; the resulting retrained ML components then provide stronger guarantees of safety than the original components [22], [23].

In [23], we demonstrated using VerifAI to debug and redesign TaxiNet, an ML-based aircraft taxiing system. Performing falsification from a Scenic program as above revealed several counterexamples to safety, such as poor performance at certain times of day (Fig. 2, orange data). We then used VerifAI to generate a new training set of the same size as the original, but with better coverage of times of day: this eliminated the failure case and improved performance overall (Fig. 2, blue data). Using VerifAI's cross-entropy sampler, we could *learn* a distribution concentrated on counterexamples and suitable for generating data to augment training sets.

Apart from counterexample-guided retraining, one can also synthesize abstractions of complex perception components (such as DNNs) that can be used for correct-by-construction control. The idea is to synthesize a simplified, but conservative, abstraction that can be used to synthesize a safe controller that is robust to failures of perception [24].

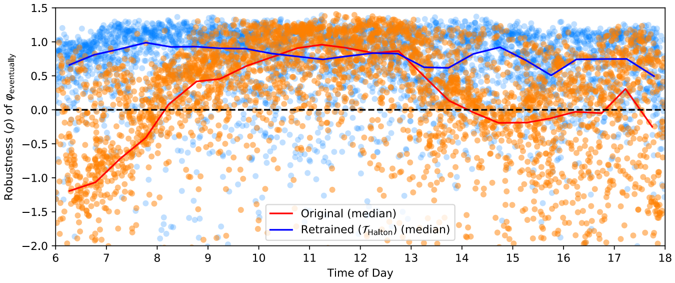

Fig. 2. [23] Robustness of MTL safety specification for TaxiNet (original and retrained) vs. time of day. Positive robustness means the spec is satisfied.

V. Runtime Assurance

Techniques for verification and synthesis can only guarantee safety with respect to a specified operating environment, the so-called *operational design domain* (ODD) of the vehicle. What happens if the vehicle leaves the ODD at run time? What if there are faults at run time that were not anticipated at design time? Can we still guarantee safety?

We believe design tools can play an important role, with two main areas of impact.

Runtime Monitoring: A runtime monitor is a component, implemented in hardware or software, that determines whether or not an AV is within its ODD. There are two key questions: (i) how do we generate accurate runtime monitors, and (ii) how do we implement efficient, real-time runtime monitors?

The paradigm of *introspective environment modeling* (IEM) offers an approach to monitor generation [25]. In IEM, a tool automatically extracts, given the design of an AV and models of its sensors, an environment assumption, expressed in terms of the sensors, under which the AV is guaranteed to be safe. IEM builds upon research on extracting environment assumptions for guaranteed-safe controller synthesis (see, e.g., [26], [27]). The major challenge is to develop good sensor models and to synthesize monitors that have low false alarm rates. Several languages have been developed, with associated compilation toolchains, to ensure that they are time- and space-efficient and can operate in real time; see, e.g., the work on the RTLola system [28].

Safe Fallback and Switching: When the runtime monitor detects that the AV is about to exit its ODD, it must trigger a mitigating action that ensures safety, albeit in a more limited form. We refer to this as the *safe fallback* action. There are multiple approaches to safe fallback: (i) defining a minimal risk maneuver where the AV can ensure safety of its passengers and people in its environment but may not be able to continue on its mission, and (ii) invoking an intervention from human supervisors, either inside the vehicle or via teleoperation.

Design tools are needed to enable designers to program the safe fallback and reason about its guarantees. The Simplex method for fault tolerance [29] provides an architecture for switching between a safe fallback controller and the nominal-mode controller to achieve overall safe operation. Programming frameworks for implementing Simplex with real-time guarantees, such as SOTER [30], must be developed.

Additionally, there is a need for methods to automatically synthesize safe fallback controllers, which may violate some specifications (e.g. parking in a no-parking zone) in order to ensure satisfaction of critical safety properties (see, e.g., [31]).

VI. Outlook

We have seen that formal methods tools can help ensure the safety of autonomous vehicles at all stages of the design process: modeling, specification, verification, synthesis, and runtime assurance. In addition to the opportunities we outlined above, the following cross-cutting challenges across these categories must also be addressed:

- *Scalability of Formal Tools:* The efficiency of formal techniques for verification and synthesis needs to be improved along multiple axes. For simulation-based verification, we need greater efficiency when running long simulations while verifying multiple properties/objectives. Advances in compositional reasoning will be crucial for achieving scalability, and this requires corresponding advances in finding good abstractions and contracts for components, including those based on AI and ML (see [6]).
- *Bridging Simulation/Models and Reality:* Verification based on simulations and other models is only meaningful if the models are faithful to the real system. In our experiments testing an AV in simulation and on the track, we observed significant differences in the simulated/real AV trajectories despite overall qualitative agreement [10]. Bridging this gap will require techniques for validating system and environment models, and for bounding the effects of simulator imprecision and incompleteness on the system.
- *Harmonizing with Standards:* Several AV safety standards and frameworks have been proposed or are being developed, including ISO 26262, ISO/PAS 21448 (SOTIF), UL 4600, BSI PAS 1881, IEEE P2009 and P2846, and the World Economic Forum's Safe Drive Initiative framework [32]. However, these standards and frameworks need strong tool support so that implementers can check compliance with applicable standards. We believe formally-grounded tools and techniques can play a crucial role in this process.

References

[1] C. Szegedy *et al.*, "Intriguing properties of neural networks," in *International Conference on Learning Representations (ICLR)*, 2014.

[2] N. E. Boudette, "Tesla's autopilot technology faces fresh scrutiny," *The New York Times*, Apr. 2021. [Online]. Available: https://www.nytimes.com/2021/03/23/business/teslas-autopilot-safety-investigations.html

[3] J. Plungis, "What Uber's fatal self-driving crash can teach industry and regulators," *Consumer Reports*, Nov. 2019. [Online]. Available: https://www.consumerreports.org/car-safety/what-ubers-fatal-self-driving-crash-can-teach-industry-and-regulators/

[4] E. Juliussen, "Calif. DMV data shows autonomous vehicles improving," *EE Times*, Feb. 2021. [Online]. Available: https://www.eetimes.com/california-dmv-data-shows-autonomous-vehicles-improving/

[5] N. Kalra and S. M. Paddock, *Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?* Santa Monica, CA: RAND Corporation, 2016.

[6] S. A. Seshia *et al.*, "Towards Verified Artificial Intelligence," *CoRR*, 2016. [Online]. Available: http://arxiv.org/abs/1606.08514

[7] T. Dreossi *et al.*, "Semantic adversarial deep learning," in *Computer Aided Verification (CAV)*, Jul. 2018, pp. 3–26.

[8] D. J. Fremont *et al.*, "Scenic: A language for scenario specification and scene generation," in *40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2019, pp. 63–78.

[9] ——, "Scenic: A language for scenario specification and data generation," *CoRR*, vol. abs/2010.06580, 2020. [Online]. Available: https://arxiv.org/abs/2010.06580

[10] ——, "Formal scenario-based testing of autonomous vehicles: From simulation to the real world," in *23rd IEEE International Conference on Intelligent Transportation Systems (ITSC)*, Sep. 2020.

[11] A. Censi *et al.*, "Liability, ethics, and culture-aware behavior specification using rulebooks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8536–8542.

[12] A. Benveniste *et al.*, "Contracts for system design," *Foundations and Trends in Electronic Design Automation*, vol. 12, no. 2-3, pp. 124–400, 2018.

[13] P. Nuzzo *et al.*, "A platform-based design methodology with contracts and related tools for the design of cyber-physical systems," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 2104–2132, Nov 2015.

[14] S. A. Seshia *et al.*, "Formal specification for deep neural networks," in *Automated Technology for Verification and Analysis (ATVA) 2018*, ser. Lecture Notes in Computer Science, S. K. Lahiri and C. Wang, Eds., vol. 11138. Springer, 2018, pp. 20–34.

[15] T. Dreossi *et al.*, "VerifAI: A toolkit for the formal design and analysis of artificial intelligence-based systems," in *31st International Conference on Computer Aided Verification (CAV)*, 2019, pp. 432–442.

[16] J. V. Deshmukh and S. Sankaranarayanan, *Formal Techniques for Verification and Testing of Cyber-Physical Systems*. Cham: Springer International Publishing, 2019, pp. 69–105. [Online]. Available: https://doi.org/10.1007/978-3-030-13050-3_4

[17] H.-D. Tran *et al.*, "Verification approaches for learning-enabled autonomous cyber-physical systems," *IEEE Design & Test*, Aug. 2020.

[18] T. Dreossi *et al.*, "Compositional falsification of cyber-physical systems with machine learning components," in *9th NASA Formal Methods Symposium (NFM)*, 5 2017, pp. 357–372.

[19] Í. Íncer Romeo *et al.*, "Quotient for assume-guarantee contracts," in *16th ACM/IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, Oct 2018.

[20] S. A. Seshia, "Combining induction, deduction, and structure for verification and synthesis," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 2036–2051, 2015.

[21] S. Jha and S. A. Seshia, "A Theory of Formal Synthesis via Inductive Learning," *Acta Informatica*, vol. 54, no. 7, pp. 693—726, 2017.

[22] T. Dreossi *et al.*, "Counterexample-guided data augmentation," in *27th International Joint Conference on Artificial Intelligence (IJCAI)*, 7 2018, pp. 2071–2078.

[23] D. J. Fremont *et al.*, "Formal analysis and redesign of a neural network-based aircraft taxiing system with VerifAI," in *32nd International Conference on Computer Aided Verification (CAV)*, Jul. 2020.

[24] S. Ghosh *et al.*, "Counterexample-guided synthesis of perception models and control," in *American Control Conference*, 2021.

[25] S. A. Seshia, "Introspective environment modeling," in *Runtime Verification (RV)*, October 2019, pp. 15–26.

[26] W. Li *et al.*, "Mining assumptions for synthesis," in *Proceedings of the Ninth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, July 2011.

[27] ——, "Synthesis for human-in-the-loop control systems," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, April 2014, pp. 470–484.

[28] P. Faymonville *et al.*, "StreamLAB: Stream-based monitoring of cyber-physical systems," in *Computer Aided Verification (CAV)*, ser. Lecture Notes in Computer Science, vol. 11561. Springer, 2019, pp. 421–431.

[29] L. Sha, "Using simplicity to control complexity," *IEEE Software*, vol. 18, no. 4, pp. 20–28, 2001.

[30] A. Desai *et al.*, "Soter: A runtime assurance framework for programming safe robotics systems," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Jun. 2019.

[31] J. Tůmová *et al.*, "Minimum-violation LTL planning with conflicting specifications," in *American Control Conference (ACC)*. IEEE, 2013, pp. 200–205.

[32] Tim Dawkins et al., World Economic Forum, "Safe Drive Initiative: SafeDI scenario-based AV policy framework – an overview for policy-makers," http://www3.weforum.org/docs/WEF_Safe_DI_AV_policy_framework_2020.pdf, Nov. 2020.