New Frontiers in Formal Methods: Learning, Cyber-Physical Systems, Education, and Beyond

Sanjit A. Seshia

Abstract—We survey promising directions for research in the area of formal methods and its applications, including fundamental questions about the combination of formal methods and machine learning, and applications to cyber-physical systems and education.

Index Terms—Formal Methods, Machine Learning, Cyber-Physical Systems, Education.

1 Introduction

Formal methods is a field of computer science and engineering concerned with the rigorous mathematical specification, design, and verification of systems [1], [2]. At its core, formal methods is about *proof*: formulating specifications that form proof obligations, designing systems to meet those obligations, and verifying, via algorithmic proof search, that the systems indeed meet their specifications. Over the past few decades, formal methods has made enormous strides. Verification techniques such as model checking [3], [4], [5] and theorem proving (see, e.g. [6], [7], [8]) are used routinely in the computer-aided design of integrated circuits and have been widely applied to find bugs in software, analyze embedded systems, and find security vulnerabilities. At the heart of these advances are computational proof engines such as Boolean satisfiability solvers [9], Boolean reasoning and manipulation routines based on Binary Decision Diagrams (BDDs) [10], and satisfiability modulo theories (SMT) solvers [11].

Even as these advances extend the capacity and applicability of formal methods, several technical challenges remain, while exciting new application domains are yet to be explored. This article expresses the author's personal viewpoint on the new frontiers for formal methods, highlighting three areas for further research and development. First, recent results have highlighted interesting conceptual connections between formal methods and the field of machine learning (inductive learning from examples), which merits further study. Second, the field of cyber-physical systems — computational systems tightly integrated with physical processes — is growing rapidly, throwing up many problems that can be addressed with formal methods. Finally, the field of education is undergoing a sea change, with the advent of massive open online courses (MOOCs) and related technologies. Formal methods can play a significant role in improving the state of the art in technologies for education. In the rest of the paper, we examine each of these frontiers for formal methods in somewhat more depth.

• *S. A. Seshia, EECS Department, UC Berkeley, Berkeley, CA 94720-1770*
E-mail: sseshia@eecs.berkeley.edu

2 Formal Methods and Inductive Learning

A major challenge for formal methods, in the author's opinion, is to develop effective proof techniques that can capture the way the best mathematicians and scientists generalize from intuition and experience to synthesize the critical parts of a proof. A major opportunity for formal methods is to blend techniques from inductive (machine) learning into traditional deductive procedures to mimick such a process of generalization. In this section, we present a brief exposition of these positions. For a more

in-depth treatment, the reader is referred to a prior article by the author [12], [13].

We begin by examining the traditional view of verification as a decision problem, with three inputs (see Figure 1):

1) A model of the system to be verified, S;
2) A model of the environment, E, and
3) The property to be verified, Φ.

The verifier generates as output a YES/NO answer, indicating whether or not S satisfies the property Φ in environment E. Typically, a NO output is accompanied by a counterexample, also called an error trace, which is an execution of the system that indicates how Φ is violated. Other debugging output may also be provided. Some formal verification tools also include a proof or certificate of correctness with a YES answer, so that an independent system can use the proof to check (usually more quickly) that the property indeed holds.

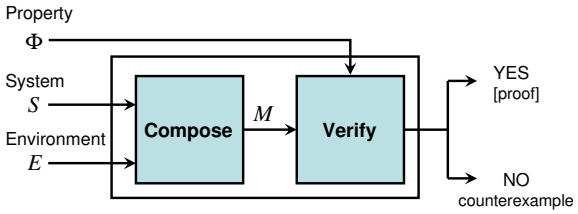


Fig. 1. Formal verification procedure.

This traditional view is somewhat high-level and idealized. In practice, one does not always start with models S and E — these might have to be abstracted from implementations or hypothesized manually. Also, the specification Φ is rarely complete and sometimes inconsistent, as has often been noted in industrial practice. Finally, the figure omits certain inputs that are perhaps the most crucial in successfully completing the verification task. For example, if performing a proof by mathematical induction, one might need to supply hints to the verifier in the form of auxiliary inductive invariants. Similarly, while performing a proof by abstraction, one may need to pick the right abstract domain and refinement strategy for generating suitable abstractions. Generating these auxiliary inputs to the proof procedure are often the trickiest steps in getting verification to succeed, as they require insight not only into the problem but also into the inner workings of the proof procedure.

The issues raised in the preceding paragraph can be distilled into to a single challenge: the effective *synthesis* of three kinds of *verification artifacts*: *models*, *specifications*, and *auxiliary inputs*. There are several examples of these verification artifacts, including inductive invariants, abstractions, environment assumptions, input constraints, auxiliary lemmas, ranking functions, and interpolants, amongst others. In Sec. 2.1, we illustrate this formulation of verification in terms of synthesis with two examples. As we will see, one often needs human insight into at least the form of these artifacts, if not the artifacts themselves, to succeed in verifying the design. In Sec. 2.2, we describe an approach to the synthesis of these artifacts by a combination of inductive learning, deductive reasoning, and human insight in the form of a "structure hypothesis."

2.1 Verification by Reduction to Synthesis

We consider here a common verification problem: proving that a certain property is an *invariant* of a system — i.e., that it holds in all states of that system. Let us first set up some notation. Relevant background material may be found in a recent book chapter on modeling for verification [14].

Let $M = (I, \delta)$ be a transition system where I is a logical formula encoding the set of initial states, and δ is a formula representing the transition relation. For simplicity, assume that M is finite-state, so that I and δ are Boolean formulas. Suppose we want to verify that M satisfies a temporal logic property $\Psi \doteq \mathbf{G}\,\psi$ where ψ is a logical formula involving no temporal operators. We now consider two methods to perform such verification.

2.1.1 Invariant Inference

Consider first an approach to prove this property by (mathematical) induction. In this case, we seek to prove the validity of the following two logical statements:

$$I(s) \implies \psi(s) \tag{1}$$

$$\psi(s) \wedge \delta(s, s') \implies \psi(s') \tag{2}$$

where, in the usual way, $\psi(s)$ denotes that the logical formula ψ is expressed over variables encoding a state s.

Usually, when one attempts a proof by induction as above, one fails to prove the validity of the second statement, Formula 2. This failure is rarely due to any limitation in the underlying validity

checkers for Formula 2. Instead, it is usually because the hypothesized invariant ψ is "not strong enough." More precisely, ψ needs to be conjoined (strengthened) with another formula, known as the *auxiliary inductive invariant*.

More precisely, the problem of verifying whether system satisfies an invariant property *reduces* to the problem of synthesizing an auxiliary invariant ϕ such that the following two formulas are valid:

$$I(s) \implies \phi(s) \wedge \psi(s) \quad (3)$$

$$\phi(s) \wedge \psi(s) \wedge \delta(s, s') \implies \phi(s') \wedge \psi(s') \quad (4)$$

If no such ϕ exists, then it means that the property ψ is not an invariant of M, since at a minimum a ϕ characterizing all reachable states of M should satisfy Formulas 3 and 4 above.

2.1.2 Abstraction-based Model Checking

Another common approach to solving the invariant verification problem is based on *sound and complete* abstraction. Given the original system M, one seeks to compute an abstract transition system $\alpha(M) = (I_\alpha, \delta_\alpha)$ such that $\alpha(M)$ satisfies Ψ if and only if M satisfies Ψ. This approach is computationally advantageous when the process of computing $\alpha(M)$ and then verifying whether it satisfies Ψ is significantly more efficient than the process of directly verifying M in the first place. We do not seek to describe in detail what abstractions are used, or how they are computed. The only point we emphasize here is that the process of computing the abstraction is a synthesis task.

In other words, instead of directly verifying whether M satisfies Ψ, we seek to synthesize an abstraction function α such that $\alpha(M)$ satisfies Ψ if and only if M satisfies Ψ, and then we verify whether $\alpha(M)$ satisfies Ψ.

2.1.3 Reduction to Synthesis

Given the two examples above, let us now step back and formalize the general notion of performing verification by reduction to synthesis.

Suppose the original verification problem is:

> Given a system M (composition of system and environment models), does M satisfy a specification Ψ?

The approach in the two examples above is to reduce this question to the form below:

> Given a specification Ω and a class of formal artifacts Σ, does there exist an element of Σ that satisfies Ω?

To make things concrete, we instantiate the symbols above in the two examples.

Invariant inference. In this case, $\Psi = \mathbf{G}\,\psi$. Σ is the set of all Boolean formulae over the (propositional) state variables of M. Ω comprises Formulas 3 and 4 above.

Abstraction-based verification. In this case, again, $\Psi = \mathbf{G}\,\psi$. Σ is the set of all abstraction functions α corresponding to a particular abstract domain [15]; for example, all possible localization abstractions [16]. Ω is the statement "$\alpha(M)$ satisfies Ψ if and only if M satisfies Ψ".

Note that this is a reduction in the complexity- and computability-theoretic sense: the verification problem has a solution if and only if the synthesis problem has one. The synthesis problem is not any easier to solve, in the theoretical sense, than the original verification problem. However, the synthesis version of the problem may be easier to solve *in practice*, especially if suitable additional constraints are imposed. We elaborate on solution techniques in the following section.

2.2 Integrating Induction, Deduction, and Structure

We have so far discussed how verification can be performed by reduction to synthesis. An effective paradigm for synthesis in recent times has been to combine induction, deduction, and structure hypotheses; an approach the author previously termed *sciduction* and formalized [12]. In this section, we present a brief introduction to some of the key ideas in this approach.

Induction is the process of inferring a general law or principle from observation of particular instances.[1] Machine learning algorithms are typically inductive, generalizing from (labeled) examples to obtain a learned *concept* or *classifier* [17], [18]. *Deduction*, on the other hand, involves the use of general rules and axioms to infer conclusions about particular problem instances. Traditional formal verification and synthesis techniques, such as

1. The term "induction" is often used in the verification community to refer to *mathematical induction*, which is actually a deductive proof rule. Here we are employing "induction" in its more classic usage arising from the field of Philosophy.

model checking or theorem proving, are deductive. One may wonder whether inductive reasoning may seem out of place here, since typically an inductive argument only ensures that the truth of its premises make it only *likely or probable* that its conclusion is also true. However, one observes that humans often employ a combination of inductive and deductive reasoning while performing verification or synthesis. For example, while proving a theorem, one often starts by working out examples and trying to find a pattern in the properties satisfied by those examples. The latter step is a process of inductive generalization. These patterns might take the form of lemmas or background facts that then guide a deductive process of proving the statement of the theorem from known facts and previously established theorems (rules). The process usually iterates between inductive and deductive reasoning until the final result is obtained.

Sciduction [13], [12] is a formalization of such a combination of inductive and deductive reasoning.[2] The key in integrating induction and deduction is the use of *structure hypotheses*, mathematical hypotheses used to define the class of artifacts to be synthesized within the overall verification or synthesis problem. Sciduction constrains inductive and deductive reasoning using structure hypotheses, and actively combines inductive and deductive reasoning: for instance, deductive techniques generate examples for learning, and inductive reasoning is used to guide the deductive engines. It is beyond the scope of this paper to explain the concept of sciduction in general. However, we provide below an intuitive explanation illustrated by counterexample-guided abstraction refinement (CEGAR) [19], a very effective approach to model checking.

2.2.1 Sciduction and Counterexample-Guided Inductive Synthesis

In sciduction, one combines an inductive inference procedure $\mathcal{I}$, a deductive engine $\mathcal{D}$, and a structure hypothesis $\mathcal{H}$ to obtain a synthesis algorithm. The structure hypothesis syntactically constrains the space of artifacts being searched. This constraint is also referred to as *syntax guidance* [20]. In machine learning, such a restricted space is called the *concept class*, and each element of that space is often called a candidate *concept*. A common sciductive approach is to formulate the overall problem as one of *active learning* using a *query-based* model. Active learning is a special case of machine learning in which the learning algorithm can control the selection of examples that it generalizes from and can query one or more oracles to obtain both examples as well as labels for those examples. (Typically the labels are positive or negative.) We refer the reader to a paper by Angluin [21] for an overview of various models for query-based active learning.

The query oracles are often implemented using deductive procedures such as model checkers or satisfiability solvers. Thus, the overall synthesis algorithm usually comprises a top-level inductive learning algorithm constrained by the structure hypothesis that invokes deductive procedures (query oracles). When the top-level strategy is inductive, we typically refer to the approach simply as "inductive synthesis" (even when deductive procedures are used). There are two important choices one must make to fix an inductive synthesis algorithm: (1) *search strategy:* How should one search the concept class? and (2) *example selection strategy:* Which examples do we learn from?

Counterexample-guided inductive synthesis (CEGIS) [22] shown in Figure 2 is perhaps the most popular approach to inductive synthesis today. The defining aspect of CEGIS is its example selection strategy: *learning from counterexamples provided by a verification oracle*. The learning algorithm, which is initialized with a particular choice of concept class and possibly with an initial set of (positive) examples, proceeds by searching the space of candidate concepts for one that is consistent with the examples seen so far. There may be several such consistent concepts, and the search strategy determines the chosen candidate. This is then presented to the verification oracle, which checks the candidate against the correctness specification. If the candidate is correct, the synthesizer terminates and outputs this candidate. Otherwise, the verification oracle returns a counterexample to the learning algorithm, which adds the counterexample to its set of examples and repeats its search. It is possible that, after some number of iterations of this loop, the learning algorithm may be unable to find a candidate concept consistent with its current set of (positive/negative) examples, in which case the learning step, and

2. sciduction stands for structure-constrained induction and deduction.

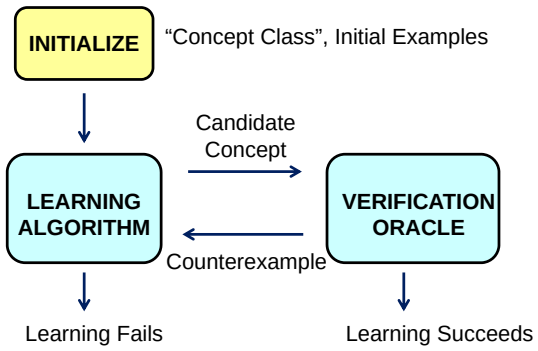


Fig. 2. Counterexample-Guided Inductive Synthesis (CEGIS)

hence the overall CEGIS procedure, fails.

Several search strategies are possible, and the choice depends on the application domain; see [20] for a more detailed discussion.

2.2.2 Counterexample-Guided Abstraction-Refinement

Counterexample-guided abstraction refinement (CEGAR) [19] is a precursor to the CEGIS approach described above. CEGAR solves a synthesis subtask of generating abstract models that are *sound* (they contain all behaviors of the original system) and *precise* (any counterexample for the abstract model is also a counterexample for the original system). One can view CEGAR as an instance of

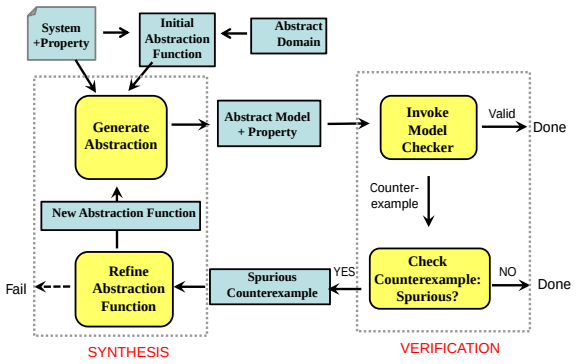


Fig. 3. Counterexample-guided abstraction refinement (CEGAR) as inductive synthesis.

sciduction as follows:

- The *abstract domain*, which defines the form of the abstraction function, is the structure hypothesis. For example, in verifying digital circuits, one might use localization abstraction [16], in which abstract states are cubes over the state variables.
- The inductive engine $\mathcal{I}$ is an algorithm to learn a new abstraction function from a spurious counterexample. Consider the case of localization abstraction. One approach in CEGAR is to walk the lattice of abstraction functions, from most abstract (hide all variables) to least abstract (the original system). This problem can be viewed as a form of learning based on version spaces [17], although the traditional CEGAR refinement algorithms are somewhat different from the learning algorithms proposed in the version spaces framework. Gupta, Clarke, et al. [23] have previously observed the link to inductive learning and have proposed versions of CEGAR based on alternative learning algorithms (such as induction on decision trees).
- The deductive engine $\mathcal{D}$, for finite-state model checking, comprises the model checker and a SAT solver. The model checker is invoked on the abstract model to check the property of interest, while the SAT solver is used to check if a counterexample is spurious.

As a point of contrast, the *cone-of-influence* approach [5] of computing an abstract model is a purely deductive approach. It is often the case that the abstract model computed via cone-of-influence is too detailed to provide the efficiency benefits of abstraction.

2.2.3 Directions for Future Work

There are four major differences between traditional machine learning algorithms (and their applications) and the ones used in formal verification and synthesis. First, in many traditional uses of machine learning, the learning algorithm operates by drawing (labeled) examples at random from some source with no control over the examples it draws. In contrast, the form of learning in verification and synthesis more commonly tends to be *active*, where the learning algorithm actively selects the examples it learns from. Second, in traditional machine learning, concept classes tend to be simpler than the ones considered in verification. In verification and synthesis, we are usually interested in synthesizing fairly general logical artifacts and programs. Third, in traditional machine learning, the learning algorithm tends to be a special-purpose algorithm designed for the specific concept class being learned. On the other hand, in inductive synthesis, the learning algorithm tends to be a general purpose logical

reasoning engine, such as a SAT or SMT solver. Fourth, while machine learning is almost entirely data-driven, inductive learning in formal verification typically operates via a combination of data-driven and model-driven algorithms.

These differences mean that there is much more to be done to develop a theory of inductive learning in the context of formal methods. Specifically, one needs to understand the impact of different logical reasoning engines on accuracy and efficiency of learning, as well as the properties of a learning algorithm that might make it well-suited for use in an overall verification or synthesis task.

In this regard, there are two results obtained by the author and colleagues that provide some initial insights. First, the notion of *teaching dimension*, elucidated by Goldman and Kearns [24], provides a lower bound on the number of examples needed by inductive synthesis techniques (such as CEGIS) to converge to the correct concept [25]. Further, one can formally analyze CEGIS to understand the impact of the quality of counterexamples provided by the verifier on the overall convergence to the correct concept [26]. Given the centrality of counterexample-guided learning in formal methods today, these results provide an initial basis for developing a deeper theoretical insights into its operation, and provide guidance for the use of other learning techniques for formal methods. An extended treatment of these results along with a theoretical framework may be found in [53].

3 Cyber-Physical Systems

Cyber-physical systems (CPS) are computational systems that are tightly integrated with the physical world [27]. Depending on the characteristics of CPS that are emphasized, they are also variously termed as *embedded systems*, the *Internet of Things* (IoT), the *Internet of Everything* (IoE), or the *Industrial Internet*. Examples of CPS include today's automobiles, fly-by-wire aircraft, medical devices, power generation and distribution systems, building control systems, robots, and many other systems. CPS have been around for a long time, but it is only recently that the area has come together as an intellectual discipline. Many CPS operate in safety-critical or mission-critical settings, and therefore it is important to gain assurance that they will operate correctly, as per specification. Thus, formal methods is an essential tool in the design of CPS. However, CPS also operate in highly dynamic and uncertain environments, and therefore they must be designed to be robust and adaptive. This imposes additional challenges on the modeling, verification, and synthesis of such systems. We examine in this section some of the main challenges in CPS design and how formal methods can help in addressing these.

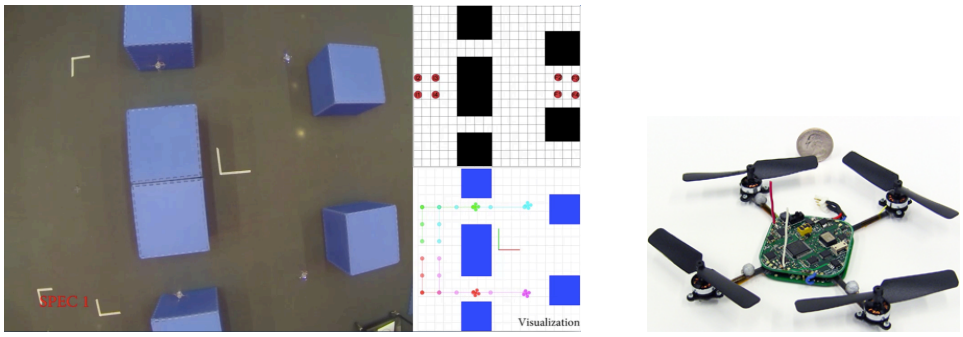
3.1 Distributed Cyber-Physical Swarms

One of the biggest trends in CPS today is the large-scale networking of devices. The author, along with several colleagues, is involved in a multi-year, multi-institution effort to develop systematic scientific and engineering principles to addressing the huge potential (and associated risks) of pervasive integration of smart, networked sensors and actuators into our connected world [28]. These distributed CPS are being termed as the "swarm" and are identified to have the following important characteristics [28]:

- *Large-scale:* the swarm comprises a vast number of nodes generating corresponding "big data;"
- *Distributed:* components of the swarm are typically networked, and are potentially separated physically and/or temporally;
- *Cyber-physical:* the swarm fuses computational processes with the physical world;
- *Dynamic:* the environment evolves continually;
- *Adaptive:* the system must adapt to its dynamic environment, and thus the distinction between "design-time" and "run-time" is blurred, and
- *Heterogeneous:* swarm components are of various types, requiring interfacing and interoperability across multiple platforms and models of computation.

This unique combination of characteristics necessitates advances in several topics in formal specification, verification and synthesis. In particular, there is a need for correct-by-construction synthesis techniques that allow quick adaptation and deployment of components into the swarm while ensuring that certain key design requirements are always satisfied.

In this section, we give the reader a flavor of the problems in the design of distributed cyber-physical systems through a sample problem: *multi-robot motion planning* for *complex requirements* specified in temporal logic.



(a) (b)

Fig. 4. Compositional SMT-Driven Multi-Robot Motion Planning: (a) Top view of sample execution and associated simulation, and (b) Nano-quadrotor platform from KMel Robotics [29] (reproduced from [30]).

Multi-Robot Motion Planning with Temporal Logic Objectives

In Robotics, the traditional motion planning problem is to move a robot from Point A to Point B while avoiding obstacles. However, more recently, there is growing interest in extending this problem along two dimensions. The first extension is to impose more complex requirements on the robot, such as visiting certain locations "infinitely often." Such requirements can be conveniently specified in a formal notation such as linear temporal logic (LTL). The second extension is to handle swarms of many robots executing coordinated plans. Such problems arise in many application settings, including persistent surveillance, search and rescue, formation control, and aerial imaging. More complex requirements require more sophisticated methods to ensure that the synthesized plans are provably correct. Scaling planning algorithms to larger swarms requires more efficient algorithms and design methodologies.

Recent work [30] addresses these challenges with a two-pronged approach. First, a compositional approach is employed, where pre-characterized motion primitives, based on well-known control algorithms, are used as a component library. Each motion primitive is specified in a suitable combination of logical theories. Second, using an encoding similar to the one used for bounded model checking, a satisfiability modulo theories (SMT) solver [11] is used to find a composition of motion primitives that achieves the desired LTL objectives. Figure 4 depicts a sample result of this approach, showing the top view of four nano quadrotor robots achieving a desired LTL specification.

These results are only a small first step. There are many more problems that remain to be solved, including inferring effective logical characterizations of motion primitives, handling dynamic and uncertain environments, dealing with non-linear encodings, incremental planning, and scaling up to an order of magnitude more robots.

3.2 Human-in-the-Loop Systems

Many cyber-physical systems are *interactive*, i.e., they interact with one or more human beings, and the human operators' role is central to the correct working of the system. Examples of such systems include fly-by-wire aircraft control systems (interacting with a pilot), automobiles with "self-driving" features (interacting with a driver), and medical devices (interacting with a doctor, nurse, or patient). We refer to the control in such systems as *human-in-the-loop control systems*. The costs of incorrect operation in the application domains served by these systems can be very severe. Human factors are often the reason for failures or "near failures", as noted by several studies (e.g., [31], [32]). Correct operation of these systems depends crucially on two design aspects: (i) *interfaces* between human operator(s) and autonomous components, and (ii) *control* strategies for such human-in-the-loop systems.

A particularly interesting domain is that of automobiles with "self-driving" features, otherwise also termed as "driver assistance systems". Such systems, already capable of automating tasks such as lane keeping, navigating in stop-and-go traffic, and parallel parking, are being integrated into high-end automobiles. However, these emerging technologies

also give rise to concerns over the safety of an ultimately driverless car. For these reasons, the field of semi-autonomous driving is a fertile application area for formal methods.

In this section, we give an overview of some of the challenges in the design of human-in-the-loop CPS (h-CPS), including:

- *Modeling:* What distinguishes a model of a h-CPS from a typical CPS?
- *Specification:* How does the formal specification change for a h-CPS?
- *Verification:* What new verification problems arise from the human aspect?
- *Synthesis:* What advances in controller synthesis are required for h-CPS?

For lack of space, we only give a brief preview of the main ideas. Further detail can be obtained in the papers by Li et al. [33] and Sadigh et al. [34].

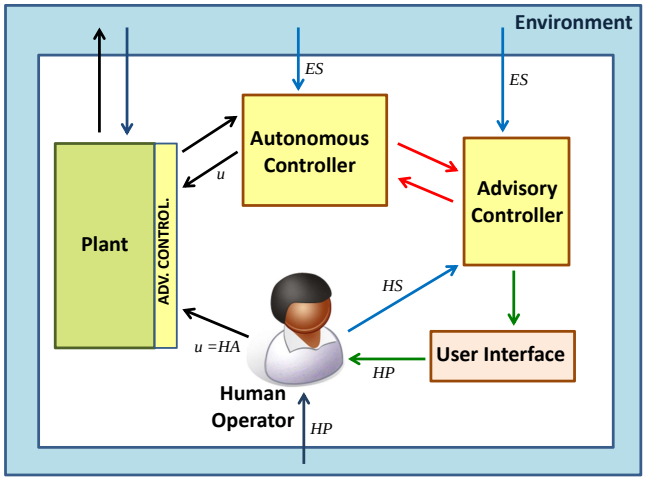


Fig. 5. **Structure of a Human Cyber-Physical System.** ES denotes environment sensing, HS denotes human sensing, HP denotes human perception, HA denotes human actions, and u denotes the vector of control inputs to the plant.

3.2.1 Modeling

For a typical (fully autonomous) control system, there are three entities: the *plant* being controlled, the *controller*, and the *environment* in which they operate. In an h-CPS, we additionally have the human operator(s) and therefore, also need a subsystem that mediates between the human operator(s) and the autonomous controller.

Formally, we model a h-CPS as a composition of five types of entities (components) [33], as illustrated in Fig. 5. The first is the *plant*, the entity being controlled. In the case of automobiles, these are the sub-systems that perform the various driving maneuvers under either manual or automatic control. The second is the *human operator* (or operators); e.g., the driver in an automobile. For simplicity, this discussion uses a single human operator, denoting her by HUMAN. The third entity is the *environment*. HUMAN perceives the environment around her and takes actions based on this perception and an underlying behavior model. We denote by HP HUMAN's perception of the environment and by HA the actions by HUMAN to control the plant. In the case of a fully-autonomous system, the human operator is replaced by an *autonomous controller* (AUTO, for short). In practice, each specialized function of the system may be served by a separately designed autonomous controller; however, for the sake of simplicity, we model the autonomous controller as a single component. AUTO perceives the environment through sensors (denoted by ES, "environment sensors") and provides control input to the plant.

The distinctive aspect of h-CPS arises from its partial autonomy. We capture this by including a fifth component, the *advisory controller* (ADVISOR, for short) [33]. The function of ADVISOR is to mediate between HUMAN and AUTO. This mediation can take different forms. For instance, ADVISOR may decide when to switch from full control by HUMAN to full control by AUTO, or vice-versa. ADVISOR may also decide to combine the control inputs from HUMAN and AUTO to the plant in a systematic way that achieves design requirements. This is indicated in Fig. 5 by the yellow box adjoining plant, between it and the AUTO and HUMAN components. We note that for legal and policy reasons, it may not be possible in many applications (including driving) for ADVISOR to always take decisions that override HUMAN. Hence, we use the term ADVISOR, indicating that the form of control exercised by ADVISOR may, in some situations, only provide suggestions to HUMAN as to the best course of action.

3.2.2 Specification

h-CPS have certain unique requirements which need to be formalized as formal specifications for verification and control. We illustrate these by the example of semi-autonomous driving.

Recognizing both the safety issues and the potential benefits of vehicle automation, in 2013 the U.S. National Highway Traffic Safety Administration (NHTSA) published a statement that provides descriptions and guidelines for the continual development of these technologies [35]. Particularly, the statement defines five levels of automation ranging from vehicles without any control systems automated (Level 0) to vehicles with full automation (Level 4). We focus on Level 3 which describes a mode of automation that requires only limited driver control:

> *"Level 3 - Limited Self-Driving Automation: Vehicles at this level of automation enable the driver to cede full control of all safety-critical functions under certain traffic or environmental conditions and in those conditions to rely heavily on the vehicle to monitor for changes in those conditions requiring transition back to driver control. The driver is expected to be available for occasional control, but with sufficiently comfortable transition time. The vehicle is designed to ensure safe operation during the automated driving mode."* [35]

Essentially, this mode of automation stipulates that the human driver can act as a fail-safe mechanism and requires the driver to take over control should something go wrong. The challenge, however, lies in identifying the complete set of conditions under which the human driver has to be notified ahead of time. Based on the NHTSA statement, we have identified [33] four important criteria required for a human-in-the-loop controller to achieve this level of automation.

1. *Effective Monitoring*. The advisory controller should be able to monitor all information about the h-CPS and its environment needed to determine if human intervention is needed.
2. *Conditional Correctness*. When the autonomous controller is in control (and not the human operator) it must satisfy a given formal specification (e.g., provided in temporal logic).
3. *Prescience*. The advisory controller must determine if the above specification may be violated ahead of time, and issues an advisory to the human operator in such a way that she has sufficient time to respond.
4. *Minimal Intervention*. The advisory controller should only invoke the human operator when it is necessary, and does so in a minimally-intervening manner (minimizing a given cost function capturing the cost of asking the human to intervene).

Li et al. [33] show how these requirements can be made precise for the case of controller synthesis from linear temporal logic.

3.2.3 Verification and Control

h-CPS are not only an application area for formal verification and control, but they have also given rise to new classes of problems.

For verification, one such problem is the verification of quantitative models of h-CPS inferred from experimental data. An example is the work on probabilistic modeling and verification of human driver behavior by Sadigh et al. [34]. Here the aim is to infer a Markov Decision Process (MDP) model for the whole closed-loop system (including human, controller, plant, and environment) from experimental data obtained from a industrial-scale car simulator. Since the model is inferred from an empirical data set that is incomplete, the model has estimation errors, e.g., in the transition probabilities. Therefore, in this case we infer a generalization of an MDP called a Convex-MDP (CMDP) [36], where the uncertainty in the values of transition probabilities is captured as a first-class component of the model. Puggelli et al. [36] show how one can extend algorithms for model checking to the CMDP model. Sadigh et al. [34] use that model to infer desired properties about human driver behavior, such as a quantitative evaluation of distracted driving.

In the setting of control of h-CPS, given that we have new kinds of specifications (as described earlier), control algorithms have to be modified to synthesize not only the autonomous controllers but also the advisory controllers. Li et al. [33] show how to modify standard algorithms for synthesis from LTL for this purpose. It is still an open problem to adapt other kinds of control algorithms to the h-CPS setting.

4 EDUCATION

The advent of massive open online courses (MOOCs) [37] promises to bring world-class education to anyone with Internet access. Moreover, it has placed a renewed focus on the development

and use of computational aids for teaching and learning. MOOCs present a range of problems to which the field of formal methods has much to contribute. These include *automatic grading*, *automated exercise generation*, and *virtual laboratory environments*. In automatic grading, a computer program verifies that a candidate solution provided by a student is "correct", i.e., that it meets certain instructor-specified criteria (the specification). In addition, and particularly when the solution is incorrect, the automatic grader (henceforth, *autograder*) should provide feedback to the student as to where he/she went wrong. Automatic exercise generation is the process of synthesizing problems (with associated solutions) that test students' understanding of course material, often starting from instructor-provided sample problems. Finally, for courses involving laboratory assignments, a virtual laboratory (henceforth, lab) seeks to provide the remote student with an experience similar to that provided in a real, on-campus lab.

In this section we briefly describe two applications of formal methods to education. Both applications have been to an undergraduate course at UC Berkeley, *Introduction to Embedded Systems* [38]. In this course, students learn theoretical content on modeling, design, and analysis [39], and also perform lab assignments on programming an embedded platform interfaced to a mobile robot [40]. Thus, this course provides a suitable setting to investigate the range of problems associated with MOOCs that are mentioned the preceding paragraph.

4.1 Exercise Generation

Consider first the task of automatic exercise generation. It is unrealistic and also somewhat undesirable to completely remove the instructor from the problem generation process, since this is a creative process that requires the instructor's input to emphasize the right concepts. However, some aspects of problem generation can be tedious for an instructor, and moreover, generating customized problems for students in a MOOC is impossible without some degree of automation. Additionally, creating many different versions of a problem can be effective at reducing cheating by blind copying of solutions.

Examining problems from all three parts of the Lee and Seshia textbook [39], Sadigh et al. [41] take a *template-based approach* to automatic problem generation. Specifically, several existing exercises in the book are shown to conform to a template. The template identifies common elements of these problems while representing the differentiating elements as parameters or "holes". In order to create a new problem, the template essentially must be instantiated with new parameter values. However, it is often useful to create new problems that are "similar" in difficulty to existing hand-crafted problems. To facilitate this, new problems are generated using a bounded number of *mutations* to an existing problem, under suitable constraints and pruning to ensure well-defined results. An instructor can then select results that look reasonable to him or her.

For brevity, we outline some of the main insights reported in [41] as they relate to the application of formal methods. The first insight relates to the structure of exercises After investigating the exercises from certain relevant chapters (Ch. 3,4,9,12,13) of Lee and Seshia [27], we found that more than 60% of problems fit into the model-based category, where the problem tests concepts involving relationships between models, properties and traces. Figure 6 is an illustration of the three entities, and their characteristics. At any point, given one or two of these entities, we can ask about instances of the unknown entity. Table 1 groups exercises into different classes based on what is given and what is to be found. Each group represents an interaction between models, properties and traces. The first column shows the given entity, and the second column is the unknown entity. The third column shows some of the variations of the same class of problem.

Table 2 states a solution technique for each problem category listed in Table 1. Note that major topics investigated in formal methods such as model checking, specification mining, and synthesis can be applied to various tasks in exercise generation. Moreover, since textbook problems are typically smaller than those arising in industrial use, their size is within the capacity of existing tools for synthesis and verification.

4.2 Grading and Feedback in Virtual Laboratories

Lab-based courses that are not software-only pose a particular technical challenge for MOOCs. A

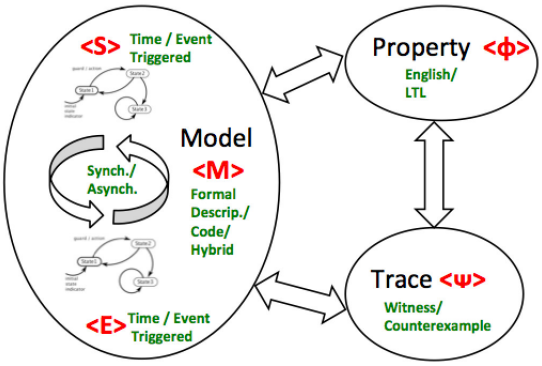


Fig. 6. Models, Properties and Traces: Entities in exercises in Lee and Seshia textbook [27] (reproduced from [41]).

Given	Find	Variations	Exercise #
$\langle\phi\rangle$	$\langle M\rangle$	(i)$\phi \in$ English or LTL (ii)use hybrid systems for M (iii)Modify pre-existing M	3.1, 3.2, 3.3, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.8, 9.4, 9.6, 13.2, 13.3
$\langle M\rangle$	$\langle\psi\rangle$	(i)reachable trace (ii)describe output	3.3, 3.5, 4.2
$\langle M\rangle$	$\langle\phi\rangle$	Models can be given in code or formal description	3.2, 12.3
$\langle M\rangle$ & $\langle\psi\rangle$	$\langle\psi\rangle$	Given input trace $\rightarrow$ find output trace	9.5
$\langle M\rangle$ & $\langle\phi\rangle$	$\langle\psi\rangle$	Find counterexample or witness trace	3.4, 4.3, 12.1

TABLE 1
Classification of Model-Based Problems in Lee and Seshia [27], First Edition, Version 1.06 (reproduced from [41])

Given	Find	Solution Technique
$\langle\phi\rangle$	$\langle M\rangle$	Constrained Synthesis or Repair
$\langle M\rangle$	$\langle\psi\rangle$	Simulation of Model
$\langle M\rangle$	$\langle\phi\rangle$	Specification Mining
$\langle M\rangle$ & $\langle\psi\rangle$	$\langle\psi\rangle$	Simulation with Guidance
$\langle M\rangle$ & $\langle\phi\rangle$	$\langle\psi\rangle$	Model Checking

TABLE 2
Techniques to Find Solutions for Model-Based Problems (reproduced from [41])

Fig. 7. Cal Climber Laboratory Platform.

key component of learning in lab-based courses is working with hardware, getting "one's hands dirty." It appears to be impossible to provide that experience online. And yet, it would be useful to provide a learning experience that approximates the real lab as well as possible. Indeed, in industrial design one often prototypes a design in a simulated environment before building the real artifact.

In an ideal world, we would provide an infrastructure where students can log in remotely to a computer which has been preconfigured with all development tools and laboratory exercises; in fact, pilot projects exploring this approach have already been undertaken (e.g., see [42]). However, in the MOOC setting, the large numbers of students makes such a remotely-accessible physical lab expensive and impractical. A virtual lab environment, driven by simulation of real-world environments, appears to be the only solution at present.

The author and colleagues have taken initial steps towards addressing this challenge for lab-based education [43], [44]. The main contribution to date is CPSGrader, which combines virtual lab software with automatic grading and feedback for courses in the areas of cyber-physical systems and robotics [44], [45], [46]. In particular, CPSGrader has been successfully used in *Introduction to Embedded Systems* at UC Berkeley [38] and its online counterpart on edX [47]. In the lab component of this course, students program the Cal Climber [40] robot (see Fig. 7) to perform certain navigation tasks like obstacle avoidance and hill climbing. Students can prototype their controller to work within a simulated environment based on the LabVIEW Robotics Environment Simulator by National Instruments (see Figure 8 and 9).

The virtual lab dynamical model is a complex, physics-based one. CPSGrader employs simulation-

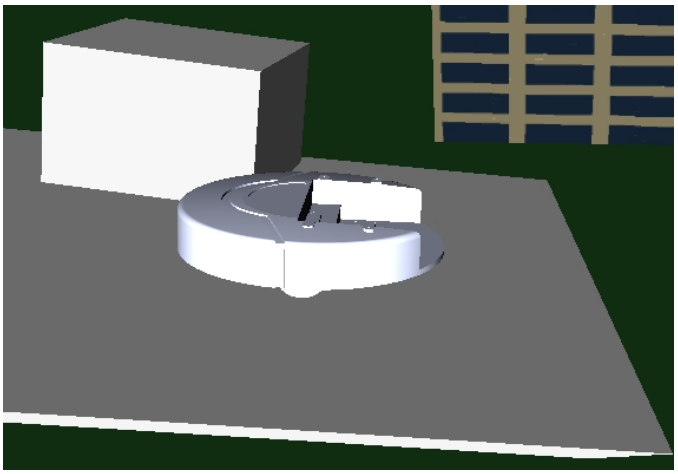


Fig. 8. Cal Climber in the LabVIEW Robotics Environment Simulator.

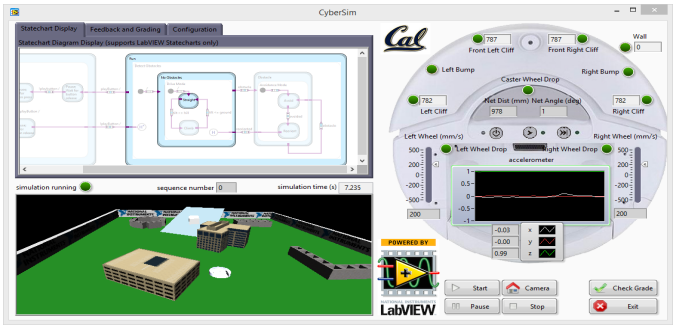


Fig. 9. Simulator with auto-grading functionality used in EECS 149.1x.

based verification, the only scalable formal approach. Correctness and the presence of certain classes of mistakes are both checked using *test benches* formalized in *Signal Temporal Logic* (STL) [48]. However, coming up with these STL properties can be tedious and error-prone, even for instructors well-versed in formal methods. Therefore, Juniwal et al. [44] show how these temporal logic testers can be generated via machine learning from solutions that have the fault (positive examples) and those that do not (negative examples). An active learning framework has also been developed to ease the burden of labeling solutions as positive or negative [45]. In machine learning terminology, this can be thought of as the *training* phase. The resulting test bench then becomes the *classifier* that determines whether a student solution is correct, and, if not, which fault is present. CPSGrader was used successfully in the edX course EECS149.1x offered in May-June 2014 [47].

There are several interesting directions for future work, including developing quantitative methods for assigning partial credit, mining temporal logic testers to capture new mistakes, and online monitoring of these testers to improve responsiveness.

5 Conclusion

In summary, in this paper we have outlined some exciting trends in formal methods and its applications. These include the combination of formal methods and inductive machine learning, and new applications to cyber-physical systems and education. Apart from these trends, there are many other exciting topics including new research in computational engines for formal methods (e.g., [49]) and emerging applications to areas such as synthetic biology [50] and computational music [51], [52].

Acknowledgments

The author thanks Prof. R. K. Shyamasundar who encouraged him to write this article. Some of the work outlined in Sec. 2 is joint with Susmit Jha. The research described in Sec. 3.1 is joint with Indranil Saha, Rattanachai Ramaithitima, Vijay Kumar, and George Pappas, that in Sec. 3.2 is joint with Wenchao Li, Dorsa Sadigh, and S. Shankar Sastry, and the work in Sec. 4 is joint with Alexandre Donzé, Mona Gupta, Jeff Jensen, Garvit Juniwal, Edward A. Lee, and Dorsa Sadigh. The work described in this paper was supported in part by an Alfred P. Sloan Research Fellowship, NSF CAREER grant #0644436, NSF Expeditions grant #1139138, and TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

[1] J. M. Wing, "A specifier's introduction to formal methods," *IEEE Computer*, vol. 23, no. 9, pp. 8–24, September 1990.

[2] E. M. Clarke and J. M. Wing, "Formal methods: State of the art and future directions," *ACM Computing Surveys (CSUR)*, vol. 28, no. 4, pp. 626–643, 1996.

[3] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching-time temporal logic," in *Logic of Programs*, 1981, pp. 52–71.

[4] J.-P. Queille and J. Sifakis, "Specification and verification of concurrent systems in CESAR," in *Symposium on Programming*, ser. LNCS, no. 137, 1982, pp. 337–351.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 2000.

[6] S. Owre, J. M. Rushby, and N. Shankar, "PVS: A prototype verification system," in *11th International Conference on Automated Deduction (CADE)*, ser. Lecture Notes in Artificial Intelligence, D. Kapur, Ed., vol. 607. Springer-Verlag, June 1992, pp. 748–752.

[7] M. Kaufmann, P. Manolios, and J. S. Moore, *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, 2000.

[8] M. J. C. Gordon and T. F. Melham, *Introduction to HOL: A Theorem Proving Environment for Higher-Order Logic*. Cambridge University Press, 1993.

[9] S. Malik and L. Zhang, "Boolean satisfiability: From theoretical hardness to practical success," *Communications of the ACM (CACM)*, vol. 52, no. 8, pp. 76–82, 2009.

[10] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, August 1986.

[11] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Satisfiability*, A. Biere, H. van Maaren, and T. Walsh, Eds. IOS Press, 2009, vol. 4, ch. 8.

[12] S. A. Seshia, "Sciduction: Combining induction, deduction, and structure for verification and synthesis," in *Proceedings of the Design Automation Conference (DAC)*, June 2012, pp. 356–365.

[13] ——, "Sciduction: Combining induction, deduction, and structure for verification and synthesis," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2011-68, May 2011.

[14] S. A. Seshia, N. Sharygina, and S. Tripakis, "Modeling for verification," in *Handbook of Model Checking*, E. M. Clarke, T. Henzinger, and H. Veith, Eds. Springer, 2014, ch. 3.

[15] P. Cousot and R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 1977, pp. 238–252.

[16] R. Kurshan, "Automata-theoretic verification of coordinating processes," in *11th International Conference on Analysis and Optimization of Systems – Discrete Event Systems*, ser. LNCS. Springer, 1994, vol. 199, pp. 16–28.

[17] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[18] D. Angluin and C. H. Smith, "Inductive inference: Theory and methods," *ACM Computing Surveys*, vol. 15, pp. 237–269, Sept. 1983.

[19] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *12th International Conference on Computer Aided Verification (CAV)*, ser. Lecture Notes in Computer Science, vol. 1855. Springer, 2000, pp. 154–169.

[20] R. Alur, R. Bodik, G. Juniwal, M. M. K. Martin, M. Raghothaman, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa, "Syntax-guided synthesis," in *Proceedings of the IEEE International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, October 2013.

[21] D. Angluin, "Queries and concept learning," *Machine Learning*, vol. 2, pp. 319–342, 1988.

[22] A. Solar-Lezama, L. Tancau, R. Bodík, S. A. Seshia, and V. Saraswat, "Combinatorial sketching for finite programs," in *ASPLOS*, 2006.

[23] A. Gupta, "Learning abstractions for model checking," Ph.D. dissertation, Computer Science Department, Carnegie Mellon University, 2006.

[24] S. A. Goldman and M. J. Kearns, "On the complexity of teaching," *Journal of Computer and System Sciences*, vol. 50, pp. 20–31, 1995.

[25] S. Jha, S. Gulwani, S. A. Seshia, and A. Tiwari, "Oracle-guided component-based program synthesis," in *Proceedings of the 32nd International Conference on Software Engineering (ICSE)*, 2010, pp. 215–224.

[26] S. Jha and S. A. Seshia, "Are there good mistakes? A theoretical analysis of CEGIS," in *3rd Workshop on Synthesis (SYNT)*, July 2014.

[27] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, First Edition. http://leeseshia.org, 2011.

[28] E. A. Lee, B. Hartmann, J. Kubiatowicz, T. S. Rosing, J. Wawrzynek, D. Wessel, J. M. Rabaey, K. Pister, A. L. Sangiovanni-Vincentelli, S. A. Seshia, D. Blaauw, P. Dutta, K. Fu, C. Guestrin, B. Taskar, R. Jafari, D. L. Jones, V. Kumar, R. Mangharam, G. J. Pappas, R. M. Murray, and A. Rowe, "The swarm at the edge of the cloud," *IEEE Design & Test*, vol. 31, no. 3, pp. 8–20, 2014.

[29] "KMel robotics." [Online]. Available: http://kmelrobotics.com/

[30] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, "Automated composition of motion primitives for multi-robot systems from safe ltl specifications," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, September 2014.

[31] Federal Aviation Administration (FAA), "The interfaces between flight crews and modern flight systems," http://www.faa.gov/avr/afs/interfac.pdf, 1995.

[32] L. T. Kohn and J. M. Corrigan and M. S. Donaldson, editors., "To err is human: Building a safer health system," A report of the Committee on Quality of Health Care in America, Institute of Medicine, Washington, DC, Tech. Rep., 2000, national Academy Press.

[33] W. Li, D. Sadigh, S. Sastry, and S. A. Seshia, "Synthesis of human-in-the-loop control systems," in *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, April 2014.

[34] D. Sadigh, K. Driggs-Campbell, A. Puggelli, W. Li, V. Shia, R. Bajcsy, A. L. Sangiovanni-Vincentelli, S. S. Sastry, and S. A. Seshia, "Data-driven probabilistic modeling and verification of human driver behavior," in *Formal Verification and Modeling in Human-Machine Systems, AAAI Spring Symposium*, March 2014.

[35] National Highway Traffic Safety Administration, "Preliminary statement of policy concerning automated vehicles," May 2013.

[36] A. Puggelli, W. Li, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Polynomial-time verification of PCTL properties of MDPs with convex uncertainties," in *Proceedings of the 25th International Conference on Computer-Aided Verification (CAV)*, July 2013.

[37] *The Year of the MOOC*, New York Times, November 2012.

[38] E. A. Lee and S. A. Seshia. EECS 149 course website. http://chess.eecs.berkeley.edu/eecs149. University of California, Berkeley.

[39] ——, *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. Berkeley, CA: LeeSeshia.org, 2011.

[40] J. C. Jensen, E. A. Lee, and S. A. Seshia, *An Introductory Lab in Embedded and Cyber-Physical Systems*. Berkeley, CA: LeeSeshia.org, 2012.

[41] D. Sadigh, S. A. Seshia, and M. Gupta, "Automating exercise generation: A step towards meeting the MOOC challenge for embedded systems," in *Proc. Workshop on Embedded Systems Education (WESE)*, October 2012.

[42] Massachusetts Institute of Technology (MIT), "The iLab Project," https://wikis.mit.edu/confluence/display/ILAB2/Home, Last accessed: February 2014.

[43] J. C. Jensen, E. A. Lee, and S. A. Seshia, "Virtualizing cyber-

physical systems: Bringing CPS to online education," in *Proc. First Workshop on CPS Education (CPS-Ed)*, April 2013.

[44] G. Juniwal, A. Donzé, J. C. Jensen, and S. A. Seshia, "CPS-Grader: Synthesizing temporal logic testers for auto-grading an embedded systems laboratory," in *Proceedings of the 14th International Conference on Embedded Software (EMSOFT)*, October 2014.

[45] G. Juniwal, "CPSGrader: Auto-grading and feedback generation for cyber-physical systems education," Master's thesis, EECS Department, University of California, Berkeley, Dec 2014. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-237.html

[46] A. Donzé, G. Juniwal, J. C. Jensen, and S. A. Seshia. CPS-Grader website. http://www.cpsgrader.org. UC Berkeley.

[47] E. A. Lee, S. A. Seshia, and J. C. Jensen. EECS149.1x Course Website on edX. https://www.edx.org/course/uc-berkeleyx/uc-berkeleyx-eecs149-1x-cyber-physical-1629. UC BerkeleyX.

[48] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *FORMATS/FTRTFT*, 2004, pp. 152–166.

[49] S. Chakraborty, D. J. Fremont, K. S. Meel, S. A. Seshia, and M. Y. Vardi, "Distribution-aware sampling and weighted model counting for sat," in *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, July 2014.

[50] S. Srivastava, J. Kotker, S. Hamilton, P. Ruan, J. Tsui, J. C. Anderson, R. Bodik, and S. A. Seshia, "Pathway synthesis using the Act ontology," in *Proceedings of the 4th International Workshop on Bio-Design Automation (IWBDA)*, June 2012.

[51] A. Donze, S. Libkind, S. A. Seshia, and D. Wessel, "Control improvisation with application to music," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-183, Nov 2013. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-183.html

[52] A. Donzé, R. Valle, I. Akkaya, S. Libkind, S. A. Seshia, and D. Wessel, "Machine improvisation with formal specifications," in *Proceedings of the 40th International Computer Music Conference (ICMC)*, September 2014.

[53] S. Jha and S. A. Seshia, "A Theory of Formal Synthesis via Inductive Learning," *ArXiv e-prints*, May 2015.