Title
Introspective Environment Modeling

ISBN
978-3-030-32078-2

Author
Seshia, Sanjit A

Publication Date
2019

DOI
10.1007/978-3-030-32079-9_2

Peer reviewed

Introspective Environment Modeling

Sanjit A. Seshia

University of California at Berkeley, Berkeley
sseshia@berkeley.edu

Abstract. Autonomous systems often operate in complex environments which can be extremely difficult to model manually at design time. The set of agents and objects in the environment can be hard to predict, let alone their behavior. We present the idea of *introspective environment modeling*, in which one algorithmically synthesizes, by introspecting on the system, assumptions on the environment under which the system can guarantee correct operation and which can be efficiently monitored at run time. We formalize the problem, illustrate it with examples, and describe an approach to solving a simplified version of the problem in the context of temporal logic planning. We conclude with an outlook to future work.

1 Introduction

Autonomous systems, especially those based on artificial intelligence (AI) and machine learning (ML), are increasingly being used in a variety of application domains including healthcare, transportation, finance, industrial automation, etc. This growing societal-scale impact has brought with it a set of risks and concerns about the dependability and safety of AI-based systems including about errors in AI software, faults, cyber-attacks, and failures of human-robot interaction. In a previous article [13], the author defined "*Verified AI*" as the goal of designing AI-based systems that have strong, ideally provable, assurances of correctness with respect to mathematically-specified requirements. That article lays out five major challenges to applying formal methods for achieving this goal, and proposes principles towards overcoming those challenges. One of those challenges is that of *modeling the environment* of an AI-based autonomous system.

The environments in which AI-based autonomous systems operate can be very complex, with considerable uncertainty even about how many and which agents are in the environment (both human and robotic), let alone about their intentions and behaviors. As an example, consider the difficulty in modeling urban traffic environments in which an autonomous car must operate. Indeed, AI/ML is often introduced into these systems precisely to deal with such complexity and uncertainty! From a formal methods perspective, this makes it very hard to create realistic environment models with respect to which one can perform verification or synthesis.

A particularly vexing problem for environment modeling is to deal with *unknown variables* of the environment. In the traditional success stories for formal verification, such as verifying cache coherence protocols or device drivers, the *interface variables* between the system S and its environment E are well-defined. The environment can only influence the system through this interface. However, for AI-based systems such

as an autonomous vehicle, it may be impossible to precisely define all the variables (features) of the environment. Even in restricted scenarios where the environment variables (agents) are known, there is a striking lack of information, especially at design time, about their behaviors. Additionally, modeling sensors such as LiDAR (Light Detection and Ranging) that represent the interface to the environment is in itself a technical challenge.

In this paper, we present *introspective environment modeling* (IEM), an idea introduced in [13] to address this challenge. The central idea in this approach is to *introspect on the system in order to model the environment*. In other words, analyze the system's behavior and its sensing interface to the environment to extract a representation of environments in which correct operation is guaranteed. A key underlying computational problem is to *algorithmically identify assumptions* that the system makes about the environment that are sufficient to guarantee the satisfaction of the specifications. In general, we want to generate the weakest set of assumptions on the environment of the AI-based autonomous system. These assumptions form critical components of an assurance case for the safety and correctness of the autonomous system, since they precisely pinpoint weak points of the system. Moreover, the assumptions identified must be *efficiently monitorable at run time*: one must be able to monitor at run time whether they are true or false, and ideally to also be able to predict whether they are likely to be violated in advance, with sufficient lead time. Additionally, in situations where human operators may be involved, one would want the assumptions to be translatable into an explanation that is *human understandable*, so that the autonomous system can "explain" to the human why it may not be able to satisfy the specification (this information can be used offline for debugging/repair or online for control). We illustrate our ideas with examples drawn from the domain of autonomous driving, and more generally, for autonomous cyber-physical systems (CPS) and robotics.

Related Work: The topic of environment modeling, also termed as "world modeling", has been much studied in the literature in formal methods, AI, and related areas. We do not attempt to cover the vast literature here, focusing instead on algorithmic methods and other closely related papers. A common approach in the AI literature is to have a probabilistic model of the world and maintain a belief distribution over possible worlds which can be updated at run time. However, the model (or model structure) is typically created manually and not algorithmically. Some world models can be monitored at run time and updated online; this has been demonstrated, e.g., in the case of autonomous vehicles [14]. In the formal methods literature, the inspiration for IEM comes from the work on automated generation of environment assumptions. Closely related work includes that of Chatterjee et al. [4] on finding minimal environment assumptions, the work of Li et al. [10] on the first counterstrategy-guided approach to inductive synthesis of environment assumptions, and the subsequent work by Alur et al. [3]. However, none of these works focused on generating environment models that could be efficiently monitored at run time. To our knowledge, our prior work [11] is the first to place this requirement and show how to mine assumptions that can be efficiently monitored at run time during the controller synthesis process. All of the above work on assumption mining is for discrete systems/models. Later Ghosh et al. [9] showed how to algorithmically repair specifications, including environment assumptions, for receding horizon

model predictive control for cyber-physical systems, but this work assumes knowledge of the structure of the environment model. In a more recent paper, Ghosh et al. [8] show how to close the gap between a high-level mathematical model of a system and its environment and a simulatable or executable model. Focusing on reach-avoid objectives, this work also assumes knowledge of the environment model structure, but shows how to adapt the environment model to account for behavioral discrepancies between the two models. Damm and Finkbeiner [5] present an approach to representing and analyzing the "perimeter" of the world model, which captures the environment variables that can be modeled and restricted via environment assumptions. They provide a notion of an optimal world model with respect to the specification and class of system strategies based on the idea of dominant strategies; such a notion of optimality could be useful in IEM as well.

The rest of the paper is organized as follows. In Section 2, we explain the idea of introspective environment modeling using an illustrative example, and formalize it. Section 3 shows how traditional controller synthesis from linear temporal logic (LTL) can be extended to perform IEM. We conclude in Section 4 with a discussion of future work.

2 Introspective Environment Modeling: The Idea

We now present the basic idea of introspective environment modeling (IEM). We start in Sec. 2.1 with a discussion of the problem, and then illustrate it with an example in Sec. 2.2. We conclude in Sec. 2.3 with a formalization of the IEM problem.

2.1 Problem Setup

Consider the standard picture of a closed system as shown in Fig. 1, where a system S interfaces to an environment E through sensors and actuators. Let x_S denote the state variables of S, x_E denote the state variables of E, y denote the inputs to S as output from its sensors, and z denote the outputs from S as generated by its actuators. The variables x_E represent the state of the agents and objects in the environment.

The challenge that IEM seeks to address is the lack of information about the environment E and its behavior. More specifically, there are three kinds of uncertainty about the environment:

1. *Uncertainty about Parameters:* The variables x_E are known and the dynamical model of E indicating how x_E changes is known too, but values of parameters in the dynamical model of E are not known.
2. *Uncertainty about Dynamics:* Variables x_E are known, but the dynamical model of E governing their evolution is not known.
3. *Uncertainty about Variables:* Even the agents in E and the variables x_E are not completely known, let alone the dynamics of E.

There are a variety of techniques available to deal with uncertainty about parameters, including using system identification or machine learning methods to estimate parameter values or ranges. The second and third type of uncertainty are much harder to handle and are the primary subject of this paper. We next introduce a simple example to illustrate the main ideas.

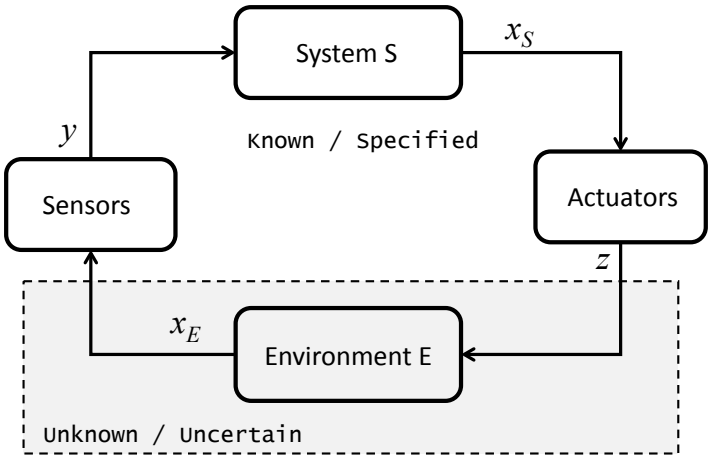


Fig. 1: System S and Environment E in a closed loop.

2.2 Illustrative Example

Consider the traffic scenario depicted in Fig. 2 on an arterial road with a top speed of 45 mph (about 20 m/s). The blue car is an autonomous vehicle (AV) travelling at 20 m/s with no vehicles or obstructions initially in the lane in front of it. On the right (slow) lane is a slow-moving line of (orange) cars. The AV is equipped with a LiDAR sensor that allows it to detect objects around it to a range of 300m [2], which is sufficient to cover the entire road scene shown in the figure. The LiDAR sensor allows the AV to estimate the position and velocity of each of the five cars A-E in the right lane; we assume for simplicity that the sensor's estimate is perfect. The challenge is that each of these five cars might suddenly decide to change to the left lane in which the AV is travelling, and the AV must avoid a collision. What assumptions on the environment (cars A-E) guarantee the safety of the AV in this scenario? Under those assumptions, what actions must the AV take to avoid a collision?

To answer these questions, first, we must formalize the notion of safety. Suppose that the true safety property is to guarantee that *the distance between the AV and any environment object is always greater than zero*. Such a property is expressible in a standard specification language such as linear temporal logic and its extensions for CPS such as metric temporal logic and signal temporal logic, as follows:

$$\mathtt{G}\,\Big[\bigwedge_{o \in Obj} dist(\mathbf{x}_{AV}, \mathbf{x}_o) > 0\Big]$$

One challenge with specifying such a property is that not all the environment objects (the set Obj) are known, and therefore, such a property cannot be guaranteed to hold at run time. Therefore, we suggest to specify a weaker property that can be monitored by available sensors. This requires the property to be based on the sensor model and be

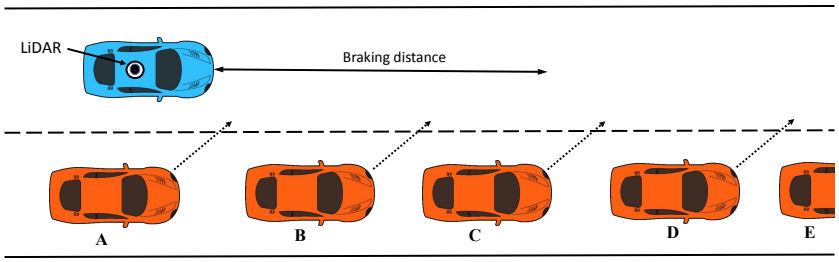


Fig. 2: Autonomous Vehicle Scenario Illustrating IEM. The blue car at the top is an autonomous vehicle travelling from left to right, while the orange cars are in a slow-moving lane from which they may possibly change lane.

time-bounded, i.e., expressed over a finite window of time over which the environment can be monitored with sufficient accuracy. For the example in Fig. 2, the absence-of-collision property will need to be revised on two counts: (1) to objects detectable by available sensors, and (ii) a finite window of time. This revised property can be formulated as follows:

$$\mathtt{G}_{[0,\tau]}\ \Big[\bigwedge_{o \in ObjSens(t)} dist(\mathbf{x}_{AV}, \mathbf{x}_o) > 0\Big]$$

where τ is a (typically short) time bound and $ObjSens(t)$ is the set of objects that are sensed by available sensors at the current time instant t. Using this alternate specification is sound only if it implies the original property over the $[0, \tau]$ interval. In this case, since we assume that the LiDAR range covers the entire scene, $ObjSens$ equals Obj.

While designing the controller for the AV, we have no knowledge of the exact number of environment agents (vehicles) or how they will behave. In the context of this example, the idea of IEM is as follows: given a strategy for the AV (based on its controller, sensors, etc.), extract an assumption on the environment E that guarantees the property of interest. As mentioned earlier, the AV is travelling at a velocity of 20 m/s. A typical braking distance from that speed on modern automobiles is about 24-48 m depending on road surfaces; we assume it to be 40 m for this example, which gives the AV 2 seconds to come to a complete stop. Typical lane widths in the United States are about 3 m [1]. Thus, assuming that the environment car starts in the middle of its lane, and vehicles are no more than 2 m wide, it would need to move with a lateral velocity of at least 1 m/s to cause a collision, provided it ends in the portion of the left lane overlapping with the braking distance.

Let us assume that the AV samples sensors periodically at (small) intervals of time Δ (of the order of a few milliseconds), and the control strategy executes instantaneously after the current sensor sample is received. Further, assume that at the current time step, the AV's strategy is to drive straight at 20 m/s and brake to stop within 2 seconds when it detects an object moving into its path. In this case, we set $\tau = 2$ to be the time bound within which the AV can come to a complete stop. Then, the AV can avoid a collision provided the following conditions hold on the environment agents: (1) Vehicle A moves

with lateral velocity v_A less than 1 m/s, and (2) Vehicle B moves with lateral velocity v_B less than 0.5 m/s. In logic, this is expressed as the following predicate:

$$\mathsf{G}_{[0,\Delta]}\ (v_A < 1) \wedge (v_B < 0.5)$$

It can avoid a collision with vehicles C, D, and E no matter their lateral velocity as they are further than the braking distance. Note that these conditions are evaluated at the current step, and must be re-evaluated at the next step Δ time units later.

The reader will note that the environment assumption specified above can be efficiently monitored provided the estimation of velocities v_A and v_B is performed efficiently over a small window of sensor samples. If the assumption is broken, mitigating actions must be taken, such as moving to a degraded mode of operation with a weaker specification guaranteed. However, we also note that the process of coming up with these assumptions involved somewhat tedious manual reasoning, even for a small example like the one in this section with perfect LiDAR. Ideally, we need algorithmic methods to generate the environment assumptions automatically. Further, it would be best to co-synthesize those assumptions along with a strategy for the system (AV) to execute. We will discuss these in Sec. 3 after formalizing the IEM problem in Sec. 2.3.

2.3 Formalization

This section formalizes the IEM problem.

Consider Fig. 1. We model the system S as a transition system $(\mathcal{X}_S, \mathcal{X}_S^0, \mathcal{Y}_S, \mathcal{Z}_S, \delta_S, \rho_S)$ where $\mathcal{X}_S$ is the set of states, $\mathcal{X}_S^0$ is the set of initial states, $\mathcal{Y}_S$ is the set of inputs to S from its sensors, $\mathcal{Z}_S$ is the set of outputs generated by S via its actuators, $\delta_S \subseteq \mathcal{X}_S \times \mathcal{Y}_S \times \mathcal{X}_S$ is the transition relation, and $\rho_S \subseteq \mathcal{X}_S \times \mathcal{Y}_S \times \mathcal{Z}_S$ is the output relation. As before, we will denote a system state by x_S; this will also denote the variables used to model a system state. We model S as a non-deterministic system rather than as a stochastic system, but the core problem formulation carries over to other formalisms. We assume that a non-zero amount of time elapses between transitions and between outputs; for convenience we will assume this to be Δ time units as in the preceding section.

So far the formal model is fairly standard. Next, we consider the environment. As discussed earlier, the environment states and model are unknown. Let us denote the unknown set of environment states by $\mathcal{X}_E$, and the variables representing an environment state by x_E. We will assume that x_E is also unknown.

The sensor model is a crucial component of the overall formal model. If $\mathcal{X}_E$ is known, the sensor model can be formalized as a non-deterministic map Σ from $\mathcal{X}_E$ to $\mathcal{Y}_S$, where Σ maps an environment state x_E to a vector of sensor values $y \in \mathcal{Y}_S$. However, if we do not know $\mathcal{X}_E$ then the sensor model captures the sequences of sensor values in $\mathcal{Y}_S$ that are feasible. In other words, in this case we define Σ as the set of all sensor value sequences, a subset of $\mathcal{Y}_S^\omega$, that can be physically generated by the sensors in some environment. We say *an environment is consistent with* Σ if it only produces sensor value sequences in Σ.

The desired specification, denoted by Φ^*, is a function of x_S and x_E. For example, this can be a temporal property indicating that an AV maintains a minimum safety

distance from any objects in the environment. However, since we do not know x_E, we instead have an alternative specification Φ which is a function of x_S, y and z. The following property must hold between these specifications:

Proposition 1. *Given specifications Φ and Φ^*, and a sensor model Σ, for all environments consistent with Σ, if a system satisfies Φ, it also satisfies Φ^*.*

We are now ready to define the introspective environment modeling problem formally.

Problem 1. Given a system S, a sensor model Σ, a specification Φ, generate an environment assumption $\Psi(x_S, y, z)$ such that S satisfies the specification $(\Psi \implies \Phi)$ in environments consistent with Σ.

Two other important aspects of the IEM problem are:

1. The environment assumptions Ψ must be *efficiently monitorable at run time*. More specifically, the environment assumption should be evaluated in sub-linear time and space (in the length of the trace), and ideally in constant time and space.
2. When Ψ does not hold and Φ is violated, the violation of Ψ should occur well in advance of the violation of Φ so that S can take mitigating actions, such as moving into a degraded mode of operation where it satisfies a weaker specification.

Note that S is not required to satisfy Φ when the environment assumption Ψ is violated. However, we want S to be "aware of its assumptions": the violation of Ψ should be detected by S and it should take actions that preserve a core (safety) specification.

In the following section, we present an approach to solving the IEM problem for a simple case where the world is modeled using propositional temporal logic and the controller for S is synthesized using standard game-theoretic approaches to reactive synthesis from temporal logic.

3 IEM for Synthesis from Temporal Logic

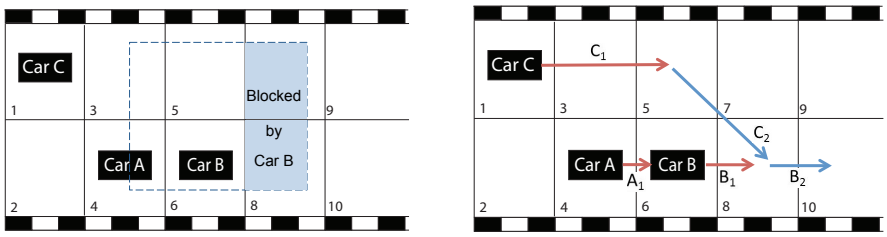
We now discuss how the IEM problem can be tackled in the context of synthesis of controllers from temporal logic specifications. The basic idea was presented in our prior work on synthesis for human-in-the-loop systems such as semi-autonomous vehicles [11]. Concretely, we consider the setting where a specification is given in linear temporal logic (LTL) in the GR(1) fragment [12], and one seeks to synthesize a controller so as to satisfy that specification in an adversarial environment. In GR(1), the specification is of the form $\varphi_a \implies \varphi_g$, where φ_a and φ_g are conjunctions of specific LTL formula types capturing safety and fairness properties, where φ_a represents the assumptions and φ_g represents guarantees. While typically the entire LTL specification is given as input, for the variant of the problem we consider, the guarantees are given, but the assumptions about the environment may be absent or only partial.

For this section, we consider S to be a finite-state transducer (FST) whose inputs $\mathcal{Y}_S$ are valuations to a set of Boolean input propositions and outputs $\mathcal{Z}_S$ are assignments to a set of Boolean output propositions. The sensor model Σ defines the sequences of input propositions that are physically feasible in some environment. The specification

Φ is a GR(1) formula of the form $\varphi_a \implies \varphi_g$, where φ_a may be $\mathbf{true}$. We wish to solve the IEM problem, i.e., to synthesize an environment assumption Ψ that implies Φ and additionally satisfies the following criteria: (1) it is efficiently monitorable at run-time; (2) it is *prescient*, meaning that S gets at least T time units to take mitigating actions before the property is violated, and (3) it is the weakest environment assumption satisfying the above properties (for some reasonable definition of "weakest").

We next present a motivating example to illustrate this variant of the IEM problem. Then we present the algorithmic approach to synthesize Ψ. Finally, we conclude by discussing some sample results. The material in this section is substantially borrowed from the author's prior work [11].

3.1 Example



(a) A's Sensing Range. (b) Failed to Follow.

Fig. 3: Controller Synthesis – Car A Following Car B

Consider the example in Figure 3. Car A is a semi-autonomous vehicle, car B and C are two other cars on the road. We assume that the road has been divided into discretized regions that encode all the legal transitions for the vehicles on the map, similar to the discretization used in LTL synthesis for robotics and CPS, such as the work on receding horizon temporal logic planning [15]. The objective of car A is to *follow* car B. Car B and C are part of the *environment*. The notion of following can be stated as follows. We assume that car A is equipped with sensors that allows it to see two squares ahead of itself if its view is not obstructed, as indicated by the enclosed region by blue dashed lines in Figure 3a. In this case, car B is blocking the view of car A, and thus car A can only see regions 3, 4, 5 and 6. Car A is said to be able to *follow* car B if it can always move to a position where it can see car B. Furthermore, we assume that at each step cars A and C can move at most 2 squares forward, but car B can move at most 1 square ahead, since otherwise car B can out-run or out-maneuver car A.

Given this objective, and additional safety rules such as cars not crashing into one another, our goal is to automatically synthesize a controller for car A such that:

- car A follows car B whenever possible;
- and in situations where the objective may not be achievable, *switches control* to the human driver while allowing *sufficient time* for the driver to respond and take control.

In general, it is not always possible to come up with a fully automatic controller that satisfies all requirements. Figure 3b illustrates such a scenario where car C blocks the view as well as the movement path of car A after two time steps. The brown arrows

indicate the movements of the three cars in the first time step, and the blue arrows indicate the movements of car B and C in the second time step. Positions of a car X at time step t is indicated by X_t. In this failure scenario, the autonomous vehicle needs to notify the human driver since it has lost track of car B.

The IEM problem, for this example, is to *identify the environment assumptions that we need to monitor* and when they may fail, notify the driver sufficiently in advance so that the driver can take mitigating actions. In the next section, we give a brief overview of how such assumptions can be co-synthesized along with a controller.

3.2 IEM for LTL Synthesis

Our approach to solve Problem 1 is based on extending the standard game-theoretic approach to LTL synthesis, where one must solve a two-player zero-sum game between S and E. We begin with the specification Φ and check whether it is realizable (i.e. a finite-state controller can be synthesized from it). If so, no environment assumptions are needed, i.e., $\Psi = \mathbf{true}$, and we are done.

The more likely case is that Φ is unrealizable. In this case, we need to synthesize assumptions so that the resulting specification becomes realizable. For this, we follow a *counterstrategy-guided approach* to environment assumption synthesis similar to that proposed first by Li et al. [10]. A counterstrategy is a winning strategy for the environment E to force violation of Φ. The approach is based on analyzing a data structure called the *counterstrategy graph* that summarizes all possible ways for the environment to force a violation of the system guarantees. It comprises the following steps:

1. *Identify Class of Assumptions:* Fix a class of environment assumptions that is efficiently monitorable. We use a class of LTL formulas of the form $\bigwedge_i(\mathtt{G}(a_i \rightarrow \neg \mathtt{X}\, b_i))$, where a_i is a Boolean formula describing a set of assignments over variables in (y, z), and b_i is a Boolean formula describing a set of assignments over variables in y. This is a property over a pair of consecutive states. The template and the approach can be extended to properties involving over a window of size k for a constant k.
2. *Transform Counterstrategy Graph:* Analyze the counterstrategy graph to find nodes that correspond to violations of safety properties, and cycles that correspond to violations of liveness properties. Transform the graph into a condensed directed acyclic graph (DAG) by contracting strongly connected components. Identify *error nodes* — nodes in this DAG that correspond to property violations. A cut in this DAG that separates nodes corresponding to start states from the error nodes corresponds to an environment assumption – a constraint one can place on the environment to eliminate the property violations.
3. *Extract Environment Assumption from Min-Cuts:* Assign weights to the edges in the graph so as to capture the penalty of reporting an environment assumption (and transferring control from the controller S to a higher level, supervisory controller such as a human operator). We consider all cuts in the graph that are at least T edges (steps) away from any error node in order to report an environment assumption violation T time steps in advance of a potential property violation. Thus, we find a min-cut in the counterstrategy graph at least T steps away from an error node. Each

edge in the cut provides one of the conjuncts in the template formula $\bigwedge_i(\mathtt{G}(a_i \rightarrow \neg \mathtt{X}\, b_i))$.

Further details of this approach are available in [11]. We demonstrate its working on the simple example in the following section.

3.3 Results

We now describe the operation of the approach on the car-following example introduced in Section 3.1. We denote the positions of cars A, B, C by p_A, p_B, p_C respectively; these variables indicate the rectangular regions where the cars are located at the current instant. Φ is of the form $\phi_S \implies \phi_E$ where each of the ϕ_i's are conjunctions of properties. We list some of these below:

- Any position can be occupied by at most one car at a time (i.e., no collisions):
$$\mathtt{G}\big(p_A = x \rightarrow (p_B \neq x \wedge p_C \neq x)\big)$$
where x denotes a position on the discretized space. The cases for B and C are similar, but they are part of ψ_E.
- Car A is required to follow car B:
$$\mathtt{G}\big((v_{AB} = \mathbf{true} \wedge p_A = x) \rightarrow \mathtt{X}(v_{AB} = \mathbf{true})\big)$$
where $v_{AB} = \mathbf{true}$ iff car A can see car B.
- Two cars cannot cross each other if they are right next to each other. For example, when $p_C = 5$, $p_A = 6$ and $p'_C = 8$ (in the next cycle), $p'_A \neq 7$. In LTL,
$$\mathtt{G}\big(((p_C = 5) \wedge (p_A = 6) \wedge (\mathtt{X}p_C = 8)) \rightarrow (\mathtt{X}(p_A \neq 7))\big)$$

The other specifications can be found in the supplementary material of Ref. [11]. Observe that car C can in fact force a violation of the system guarantees in one step under two situations – when $p_C = 5, p_B = 8$ and $p_A = 4$, or $p_C = 5, p_B = 8$ and $p_A = 6$. Both are situations where car C is blocking the view of car A, causing it to lose track of car B. The second failure scenario is illustrated in Figure 3b.

Applying our algorithm to this (unrealizable) specification with $T = 1$, we obtain the following assumption Ψ.

$$\Psi = \mathtt{G}\big(((p_A = 4) \wedge (p_B = 6) \wedge (p_C = 1)) \rightarrow \neg\mathtt{X}((p_B = 8) \wedge (p_C = 5))\big) \bigwedge$$
$$\mathtt{G}\big(((p_A = 4) \wedge (p_B = 6) \wedge (p_C = 1)) \rightarrow \neg\mathtt{X}((p_B = 6) \wedge (p_C = 3))\big) \bigwedge$$
$$\mathtt{G}\big(((p_A = 4) \wedge (p_B = 6) \wedge (p_C = 1)) \rightarrow \neg\mathtt{X}((p_B = 6) \wedge (p_C = 5))\big)$$

Note how Ψ reports a violation at least $T = 1$ time steps ahead of a potential property failure. Also, Ψ corresponds to three possible evolutions of the environment from the initial state. In general, Ψ can be a conjunction of conditions at different time steps as E and S progress.

These results indicate the feasibility of an algorithmic approach to generating environment assumptions for temporal logic based planning. However, there are also several limitations. First, it is hard to scale this explicit graph-based approach to large state

spaces and specifications. Second, it is only applicable to problems where a discretization of the state space is meaningful for planning in the real world. Recent work on repair of specifications for receding horizon control for real-time temporal logics over continuous signals [9] provides a starting point, although those methods need to be extended to handle highly adversarial environments. Third, the sensor model is highly simplified. Nevertheless, a counterstrategy-based approach provides a first step to producing environment models in the form of logical specifications (assumptions) that are usable for controller synthesis, efficiently monitorable at run time, and provide time for taking mitigating actions when the assumptions are violated.

4 Conclusion

We presented the idea of introspective environment modeling (IEM) as a way of dealing with the challenge of modeling unknown and uncertain environments of autonomous systems. The central idea is to introspect on the working of the system in order to capture a set of environment assumptions that is sufficient to guarantee correct operation and which is also efficiently monitorable at run time. We formalized the IEM problem and described an algorithmic approach to solving it for a simplified setting of temporal logic planning.

Much more remains to be done to solve the IEM problem in practice. First, the algorithmic approach presented in Sec. 3 must be extended from the discrete setting to cyber-physical systems. The scalability challenge must be addressed, moving from the explicit graph-theoretic method of Sec. 3 to one that scales to high-dimensional spaces involving both discrete and continuous variables, likely requiring symbolic methods. A particularly important problem is to devise realistic sensor models that capture the noise and errors that arise in real-world sensors; while this is challenging, we believe this is an easier modeling problem as the number of sensor types is much less than the number of possible environments. Approaches to extract the weakest environment assumptions that are also efficiently monitorable at run time must be investigated further. It would also be useful to explore formalisms to capture environment assumptions beyond temporal logic, and the use of IEM for stochastic environment models, represented, e.g., using probabilistic programming languages [7]. Finally, we believe the extracted assumptions and the sensor model could be valuable in building an assurance case for autonomous systems, especially when combined with techniques for run-time assurance (e.g., [6]).

Acknowledgments

I gratefully acknowledge the contributions of students and other collaborators in the work that this article draws from. This work was supported in part by NSF grants 1545126 (VeHICaL) and 1646208, the DARPA Assured Autonomy program, the iCyPhy center, and Berkeley Deep Drive.

References

1. Typical lane widths. `https://en.wikipedia.org/wiki/Lane#Lane_width`.

2. Velodyne Lidar: Products. https://velodynelidar.com/products.html.
3. Rajeev Alur, Salar Moarref, and Ufuk Topcu. Counter-strategy guided refinement of gr(1) temporal logic specifications. In *Proceedings of the 13th Conference on Formal Methods in Computer-Aided Design (FMCAD'13)*, pages 26–33, 2013.
4. Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Environment assumptions for synthesis. In *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR'08)*, pages 147–161. Springer, 2008.
5. Werner Damm and Bernd Finkbeiner. Does it pay to extend the perimeter of a world model? In *International Symposium on Formal Methods*, pages 12–26. Springer, 2011.
6. Ankush Desai, Shromona Ghosh, Sanjit A. Seshia, Natarajan Shankar, and Ashish Tiwari. A runtime assurance framework for programming safe robotics systems. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, June 2019.
7. Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Scenic: A language for scenario specification and scene generation. In *Proceedings of the 40th annual ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI)*, June 2019.
8. Shromona Ghosh, Somil Bansal, Alberto Sangiovanni-Vincentelli, Sanjit A. Seshia, and Claire J. Tomlin. A new simulation metric to determine safe environments and controllers for systems with unknown dynamics. In *Proceedings of the 12th International Conference on Hybrid Systems: Computation and Control (HSCC)*, pages 185–196, April 2019.
9. Shromona Ghosh, Dorsa Sadigh, Pierluigi Nuzzo, Vasumathi Raman, Alexandre Donzé, Alberto L. Sangiovanni-Vincentelli, S. Shankar Sastry, and Sanjit A. Seshia. Diagnosis and repair for synthesis from signal temporal logic specifications. In *Proceedings of the 9th International Conference on Hybrid Systems: Computation and Control (HSCC)*, April 2016.
10. Wenchao Li, Lili Dworkin, and Sanjit A. Seshia. Mining assumptions for synthesis. In *Proceedings of the Ninth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*, pages 43–50, July 2011.
11. Wenchao Li, Dorsa Sadigh, S. Shankar Sastry, and Sanjit A. Seshia. Synthesis for 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)*, pages 470–484, April 2014.
12. Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive(1) designs. In *Proceedings of the 7th Verification, Model Checking, and Abstract Interpretation (VMCAI'06)*, pages 364–380. Springer, 2006.
13. Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. Towards Verified Artificial Intelligence. *ArXiv e-prints*, July 2016.
14. Chris Urmson, Chris Baker, John Dolan, Paul Rybski, Bryan Salesky, William Whittaker, Dave Ferguson, and Michael Darms. Autonomous driving in traffic: Boss and the urban challenge. *AI Magazine*, 30(2):17–17, 2009.
15. Tichakorn Wongpiromsarn et al. Receding horizon temporal logic planning. *IEEE Transactions on Automatic Control*, 57(11):2817–2830, 2012.