Synthesis for Human-in-the-Loop Control Systems

Wenchao Li
Dorsa Sadigh
S. Shankar Sastry
Sanjit A. Seshia

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2013-134
http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-134.html

July 17, 2013

Acknowledgement

This work was supported in part by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA. This work was also supported by the NSF grants CCF-1116993 and CCF-1139138.

Synthesis for Human-in-the-Loop Control Systems

Wenchao Li, Dorsa Sadigh, S. Shankar Sastry, Sanjit A. Seshia
University of California, Berkeley {wenchaol, dsadigh, sastry, sseshia}@eecs.berkeley.edu

***Abstract*—Several control systems in safety-critical applications involve the interaction of an autonomous controller with one or more human operators. Examples include pilots interacting with an autopilot system in an aircraft, and a driver interacting with automated driver-assistance features in an automobile. The correctness of such systems depends not only on the autonomous controller, but also on the actions of the human controller. In this paper, we present a formalism for human-in-the-loop control systems. Particularly, we focus on the problem of synthesizing a semi-autonomous controller from high-level temporal specifications that expect occasional human intervention for correct operation. We present an algorithm for this problem, and demonstrate its operation on problems related to driver assistance in automobiles.**

I. Introduction

Many safety-critical systems are *interactive*, i.e., they interact with a human being, and the human operator's 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 driver assistance systems (interacting with a driver), and medical devices (interacting with a doctor, nurse, or patient). We refer to such interactive control 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., [7], [10]).

One alternative to human-in-the-loop systems is to synthesize a fully autonomous controller from a high-level mathematical specification. The specification typically captures both assumptions about the environment and correctness guarantees that the controller must provide, and can be specified in a formal language such as linear temporal logic [16]. While this correct-by-construction approach looks very attractive, the existence of a fully autonomous controller that can satisfy the specification is not always guaranteed. For example, in the absence of adequate assumptions constraining its behavior, the environment can be modeled as being overly adversarial, causing the synthesis algorithm to conclude that no controller exists. Additionally, the high-level specification might abstract away from inherent physical limitations of the system, such as insufficient range of sensors, which must be taken into account in any real implementation. Thus, while full manual control puts too high a burden on the human operator, some element of human control is desirable. However, at present, there is no systematic methodology to synthesize a combination of human and autonomous control from high-level specifications. In this paper, we address this limitation of the state of the art. Specifically, we consider the question: Can we devise a controller that is mostly automatic and requires only occasional human interaction for correct operation? We formalize this problem of human-in-the-loop (HuIL) synthesis and establish formal criteria for solving it.

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. Recognizing the safety issues and the potential benefits of vehicle automation, the National Highway Traffic Safety Administration (NHTSA) recently published a statement that provides descriptions and guidelines for the continual development of these technologies [14]. 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). In this paper, 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."* [14]

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. Based on the NHTSA statement, we identify four important criteria required for a human-in-the-loop controller to achieve this level of automation.

- *Monitoring*. The controller should be able to determine if human intervention is needed based on monitoring past and current information about the system and its environment.
- *Minimally Intervening*. The controller should only invoke the human operator when it is necessary, and does so in a minimally intervening manner.
- *Prescient*. The controller can determine if a 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.

- *Conditionally Correct*. The controller should operate correctly until the point when human intervention is deemed necessary.

We further elaborate and formally define these concepts later in Section III. In general, a human-in-the-loop controller, as shown in Figure 1 is a controller consists of three components: an automatic controller, a human operator, and an advisory control mechanism that orchestrates the switching between the auto-controller and the human operator.[1] In this setting, the auto-controller and the human operator can be viewed as two separate controllers, each capable of producing outputs based on inputs from the environment, while the key responsibility of the advisory controller is to determine precisely when the human operator should assume control, while giving her enough time to respond.

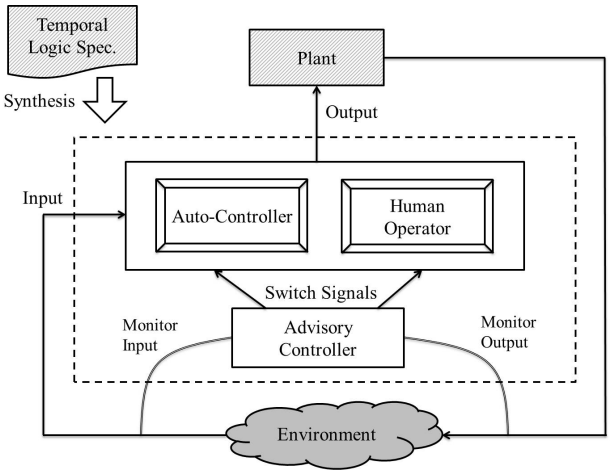


Fig. 1: Human-in-the-Loop Controller: Component Overview and Synthesis from Specification

In this paper, we study the construction of such controller in the context of *reactive synthesis* from temporal logic. *Reactive synthesis* is the process of automatically synthesizing a discrete controller (finite-state transducer) that reacts to environment changes in such a way that the given specification (in temporal logic) is satisfied. There has been growing interest recently in the control and robotics communities (e.g., [22], [9]) to apply this approach to automatically generate embedded control software.

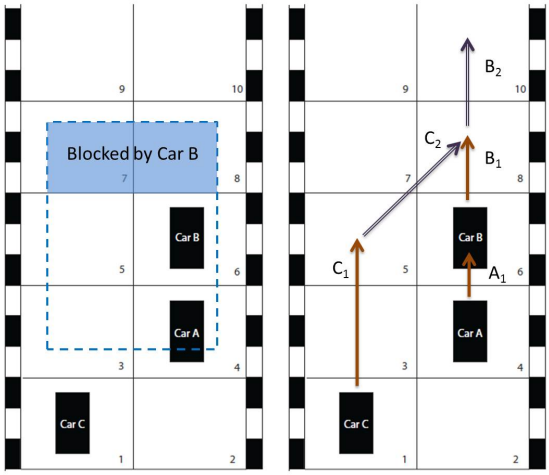
In summary, the main contributions of this paper are:

- A formalization of human-in-the-loop control systems and the problem of synthesizing such controllers from high-level specifications, including four key criteria these controllers must satisfy.
- An algorithm for synthesizing human-in-the-loop controllers that satisfy the afore-mentioned criteria.
- An application of the proposed technique to examples motivated by driver-assistance systems for automobiles.

[1]In this paper, we do not consider explicit dynamics of the plant. Therefore it can be considered as part of the environment also.

The paper is organized as follows. Section II describes an motivating example discussing a *car following* example. Section III provides a formalism and characterization of the human-in-the-loop controller synthesis problem. Section IV reviews material on reactive controller synthesis from temporal logic. Section V describes our algorithm for the problem. We then present case studies of safety critical driving scenarios in Section VI. Finally, we discuss related work in Section VII and conclude in Section VIII.

II. Motivating Example



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

Fig. 2: Controller Synthesis – Car A Following Car B

Consider the example in Figure 2. In this example, car A is the 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 setup used in receding horizon temporal logic planning [23]. The objective of car A is to *follow* car B. Note that car B and C are part of the *environment* and cannot be controlled. 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 2a. 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 car 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

1) car A follows car B whenever possible;
2) and in situations where the objective may not be achievable, *switches control* to the human driver but also

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 2b 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 purple arrows indicate the movements of car B and C in the second time step. Positions of a car X at time 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.

Hence, a human-in-the-loop synthesis approach is tasked with producing an autonomous controller along with advisories for the human driver in situations where her attention is required. Our challenge, however, is to *identify the conditions that we need to monitor and notify the driver when they may fail*. In the next section, we discuss how human constraints such as response time can be simultaneously considered in the solution, and mechanisms for switching control between the auto-controller and the human driver.

III. Formal Model of HuIL Controller

A. Preliminaries

Consider a Booleanized space over the input and output alphabet $\mathcal{I} = 2^I$ and $\mathcal{O} = 2^O$, where I and O are the variables for input and output respectively, a trace τ is an infinite sequence $(i, o)^\omega = (i_0, o_0)(i_1, o_1)\ldots$ such that $i_k \in \mathcal{I}$ and $o_k \in \mathcal{O}$ for all $k \geq 0$. To characterize the correctness of a trace, we assume we are given a function $\mathcal{F}$ that labels a trace to $\{\texttt{true}, \texttt{false}\}$. We say a trace τ is a *failure* trace if $\mathcal{F}(\tau) = \texttt{true}$. In this paper, we model a discrete controller as a finite-state transducer. A finite-state (Moore) transducer is a tuple $M = (\tilde{Q}, \tilde{q}_0, \mathcal{I}, \mathcal{O}, \delta, \theta)$, where $\tilde{Q}$ is the set of states, $\tilde{q}_0 \in \tilde{Q}$ is the initial state, $\delta : \tilde{Q} \times \mathcal{I} \to \tilde{Q}$ is the transition function, and $\theta : \tilde{Q} \to \mathcal{O}$ is the state output function. Given a word $\bar{i} = i_0 i_1 \ldots$, a run of M is the sequence $\bar{q} = q_0 q_1 \ldots$ of states such that $\bar{q}_0 = q_0$, and $q_{k+1} = \delta(q_k, i_k)$ for all $k \geq 0$. The run $\bar{q}$ on $\bar{i}$ produces the word $M(\bar{i}) = \theta(q_0)\theta(q_1)\ldots$. The language of M is then denoted by the set $\mathcal{L}(M) = \{(i, o)^\omega \mid M(\bar{i}) = \bar{o}\}$. Intuitively, if the language of a controller M does not contain any failure trace, then the controller functions correctly. For convenience, we also say a state q is *failure-prone* if a run of M containing q given some $\bar{i}$ produces a failure trace.

B. Agents as Automata

We model each of the three agents in a human-in-the-loop controller – the human operator $\mathcal{HC}$, the automatic controller $\mathcal{AC}$, and the advisory controller $\mathcal{VC}$, as finite-state transducers (FSTs). The overall controller $\mathcal{HIL}$ is a composition of the models of $\mathcal{HC}$, $\mathcal{AC}$ and $\mathcal{VC}$, and is also modeled as a finite-state transducer $\mathcal{HIL}$.

We represent the four FSTs as follows:

$$\begin{aligned}
\mathcal{HC} &= (\mathcal{I}^{\mathcal{HC}}, \mathcal{O}^{\mathcal{HC}}, \tilde{Q}^{\mathcal{HC}}, \tilde{q}_0^{\mathcal{HC}}, \delta^{\mathcal{HC}}, \theta^{\mathcal{HC}}) \\
\mathcal{AC} &= (\mathcal{I}^{\mathcal{AC}}, \mathcal{O}^{\mathcal{AC}}, \tilde{Q}^{\mathcal{AC}}, \tilde{q}_0^{\mathcal{AC}}, \delta^{\mathcal{AC}}, \theta^{\mathcal{AC}}) \\
\mathcal{VC} &= (\mathcal{I}^{\mathcal{VC}}, \mathcal{O}^{\mathcal{VC}}, \tilde{Q}^{\mathcal{VC}}, \tilde{q}_0^{\mathcal{VC}}, \delta^{\mathcal{VC}}, \theta^{\mathcal{VC}}) \\
\mathcal{HIL} &= (\mathcal{I}^{\mathcal{HIL}}, \mathcal{O}^{\mathcal{HIL}}, \tilde{Q}^{\mathcal{HIL}}, \tilde{q}_0^{\mathcal{HIL}}, \delta^{\mathcal{HIL}}, \theta^{\mathcal{HIL}})
\end{aligned}$$

where $\mathcal{I}^{\mathcal{HIL}} = \mathcal{I}$, $\mathcal{O}^{\mathcal{HIL}} = \mathcal{O}^{\mathcal{HC}} = \mathcal{O}^{\mathcal{AC}} = \mathcal{O}$. We use the binary variable $auto$ to denote the internal advisory signal that $\mathcal{VC}$ sends to both $\mathcal{AC}$ and $\mathcal{HC}$. Hence, $\mathcal{I}^{\mathcal{HC}} = \mathcal{I}^{\mathcal{AC}} = \mathcal{I} \cup \{auto\}$, and $\mathcal{O}^{\mathcal{VC}} = \{auto\}$. When $auto = \texttt{false}$, it means the advisory controller is requiring the human operator to take over control, and the auto-controller can have control otherwise. $\mathcal{HC}$ can be viewed as a hierarchical state machine as shown in Figure 3.

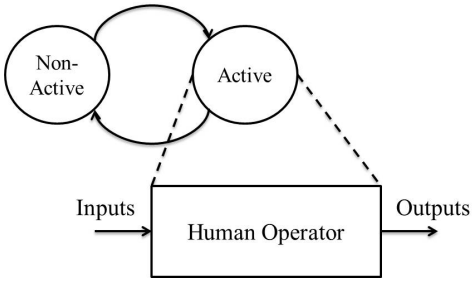


Fig. 3: Human Controller as a Hierarchical State Machine

We assume that the human operator (driver behind the wheel) can take control at any time by transitioning from the "Non-Active" state to the "Active" state, e.g. by hitting a button on the dashboard or simply pressing down the gas pedal or the brake. When $\mathcal{HC}$ is in the active state, the human operator essentially acts as the automaton that produces outputs to the plant (e.g. a car) based on environment inputs. We use a binary variable $active$ to denote if $\mathcal{HC}$ is in the "Active" state. When $active = \texttt{true}$, the output of $\mathcal{HC}$ can overwrite the output of $\mathcal{AC}$. Similarly, when $active = \texttt{false}$, the output of $\mathcal{HIL}$ is the output of $\mathcal{AC}$. Note that even though the human operator is modeled as a FST here, since we do not have direct control of the human operator, it can in fact be any arbitrary relation mapping $\mathcal{I}$ to $\mathcal{O}$.

C. Criteria for Human-in-the-loop Controllers

One key distinguishing factor of a human-in-the-loop controller from traditional controller is the involvement of a human operator. Hence, human factors such as response time cannot be disregarded. In addtion, we would like to minimize the need to engage the human operator. Based on the NHSTA statement, we derive four criteria for any effective human-in-the-loop controller, as described below.

1. *Monitoring.* An advisory $auto$ is issued to the human operator under specific conditions. These conditions in turn need to be determined at runtime. Hence, a HuIL controller must be able to monitor the input and output signals needed to evaluate these conditions, i.e. $\mathcal{I}^{\mathcal{VC}} = \mathcal{I} \cup \mathcal{O}$.

2. *Minimally intervening.* Our mode of interaction requires only selective human intervention. An intervention occurs when $\mathcal{HC}$ transitions from the "Non-Active" state to the "Active" state (we discuss mechanisms for suggesting a transition from "Active" to "Non-Active" in Section V-C, after prompted by the advisory signal $auto$ (being `false`). However, frequent transfer of control would mean constant attention is required from the human operator, thus nullifying the benefits of having the auto-controller. In order to reduce the overhead of human participation, we want to minimize a joint objective function $\mathcal{C}$ that combines two elements: (i) the *probability* that when $auto$ is set to `false`, the environment will eventually force $\mathcal{AC}$ into a failure scenario, and (ii) the *cost* of having the human operator taking control. We formalize this objective function in Sec. V-A. The condition under which $auto$ remains `true` is termed as the environment assumption and denoted by φ_{env}.
3. *Prescient.* It may be too late to seek the human operator's attention when failure is imminent. We also need to allow extra time for the human to respond and study the situation. Hence, an advisory should be issued ahead of any failure scenario. In the discrete setting, we parameterize the controller by a non-negative integer T, which is minimum number of transitions it takes to get to a state that is *failure-prone*.
4. *Conditionally-Correct.* The auto-controller is responsible for correct operation until the human operator takes over control (assuming that she does so within T time steps after $auto$ bcomes `false`). We formalize this as the following guarantees provided by the auto-controller:
   - For any trace τ that starts from a valid initial state and satisfies the environment assumption φ_{env} ($auto$ is always true), $\mathcal{HIL}$ satisfies the specification (i.e., $\mathcal{F}(\tau) = \texttt{false}$).
   - For any trace τ, if φ_{env} is violated at some point in the trace, then the safety component of the specification (formalized in Sec. IV) remains `true` for the next T time steps (until the human controller takes over).

   While other criteria may be desirable, we believe that at least the above four are necessary.

Now we are ready to state the controller synthesis problem. *HuIL controller Synthesis Problem:* Given a model of the system and its specification expressed in a formal language, synthesize a HuIL controller $\mathcal{HIL}$ that is, by construction, *monitoring, minimally intervening, presicent,* and *conditionally correct*.

In this paper, we study the synthesis of a HuIL controller in the setting of synthesis of reactive systems from linear temporal logic (LTL). We give background on this setting in Section IV, and propose an algorithm for solving the HuIL controller synthesis problem in Section V.

IV. Synthesis from Temporal Logic

The idea of temporal logic synthesis is to automatically construct an implementation that is guaranteed to satisfy a behavioral description of the system expressed in temporal logic [16]. First formulated by Church in 1962 [5], it has been used to synthesize designs in a variety of applications, such as digital circuits [3] and embedded software [22]. In this section, we give an overview on synthesizing reactive modules from a specification given in Linear Temporal Logic (LTL) [17]. In general, the problem can be viewed as a two-player game between the system sys and the environment env.

A. Linear Temporal Logic

Given a finite set of propositional (Boolean) variables P, which is the disjoint union $I \dot{\cup} O$, formulas in linear temporal logic (LTL) are constructed as follows.

$$\psi ::= p \,|\, \neg\psi \,|\, \psi \vee \psi \,|\, \mathbf{X}\,\psi \,|\, \psi_1 \mathbf{U}\, \psi_2$$

where $p \in P$ is a propositional variable, $\mathbf{X}$ is the temporal operator *next* (used to indicate that a property ψ holds from the next state in a trace) and $\mathbf{U}$ is the temporal operator *until* (used to indicate that some property ψ_2 holds in a trace eventually, while another ψ_1 holds until ψ_2 becomes true). Other temporal operators can be derived using these two temporal operators and Boolean operators: the formula "eventually ψ holds" is written as $\mathbf{F}\psi = \texttt{true}\mathbf{U}\psi$, and the formula "$\psi$ holds globally (at all time points)" is written as $\mathbf{G}\,\psi = \neg\mathbf{F}\neg\psi$. LTL formulas are usually interpreted over infinite words (traces) $w \in \Sigma^\omega$, where $\Sigma = 2^P$. Then the language of a LTL formula ψ is the set of infinite words that satisfy ψ, given by $\mathcal{L}(\psi) = \{w \in \Sigma^\omega \,|\, w \models \psi\}$. One classic example is the LTL formula $\mathbf{G}(p \rightarrow \mathbf{F}\, q)$, which means every occurrence of p in a trace must be followed by some q in the future.

Properties are usually classified as being *safety* or *liveness* properties. Informally, safety properties are those that specify that "nothing bad happens in the trace" and their violation can be demonstrated on a finite-length trace. Liveness properties on the other hand specify that "something good happens in the future" and their violation can only be shown with an infinite-length trace. As stated in the preceding section, the *conditional-correctness* guarantees state that, when the environment assumption φ_{env} does not hold, we require that the auto-controller guarantees that the safety component of the specification is satisfied for T steps until the human controller takes over.

B. Satisfiability and Realizability

A LTL formula ψ is *satisfiable* if there exists an infinite word that satisfies ψ, i.e., $\exists w \in (2^P)^\omega$ such that $w \models \psi$. A Moore transducer M *satisfies* a LTL formula ψ if $\mathcal{L}(M) \subseteq \mathcal{L}(\psi)$. We write this as $M \models \psi$. Then *realizability* is the problem of checking whether there exists a Moore transducer M that satisfies the LTL specification ψ. Informally, satisfiability means that there exists *some* behavior of the environment that makes it satisfy the property ψ, whereas

realizability means that M satisfies ψ *for all* environment behaviors.

C. Synthesis Complexity and GR(1) Specification

The complexity of LTL synthesis can be prohibitively high (2EXPTIME-complete [18]). However, Piterman et. al [15] describe an approach for synthesizing a subclass of LTL properties, known as Generalized Reactivity (1) [GR(1)] formulas. Their algorithm runs in time $O(N^3)$ where N is the size of the system to be synthesized. A GR(1) formula has the form $\psi = \psi_{env} \rightarrow \psi_{sys}$, where ψ_{env} is the *environment assumption* and ψ_{sys} is the *system guarantee*. The syntax of GR(1) formulas is given as follows. We require ψ_l for $l \in \{env, sys\}$ to be *rewritten* as a conjunction of subformulas in the following forms:

- ψ_l^i: a Boolean formula that characterizes the initial states
- ψ_l^t: a LTL formula that characterizes the transition, in the form $\mathbf{G}\ B$, where B is a Boolean combination of variables in $I \cup O$ and expression $\mathbf{X}\ u$ where $u \in I$ if $l = env$ and $u \in I \cup O$ if $l = sys$.
- ψ_l^f: a LTL formula that characterizes fairness, in the form $\mathbf{G}\,\mathbf{F}\ B$, where B is a Boolean formula over variables in $I \cup O$.

In this paper, we consider (unrealizable) specifications given in the GR(1) subclass. For certain properties, even if they are not originally syntactically in the GR(1) subclass described above, it is easy to rewrite them into this form; see [15] for more details.

D. Games and Strategies

A finite-state two-player game is defined by its *game graph*, represented by the tuple $\mathcal{G} = (Q, \Sigma, T, q_0, Win)$, where $\Sigma = \mathcal{I} \times \mathcal{O}$, $T : Q \times \Sigma \rightarrow Q$ is a deterministic and complete transition function, $q_0 \in Q$ is the initial state, and $Win : Q^\omega \rightarrow \{\texttt{false}, \texttt{true}\}$ is the winning condition. We refer the readers to [8] for the construction of $\mathcal{G}$ from GR(1) specifications. Note that each $q \in Q$ can be marked by the last tuple (i, o) (for $i \in \mathcal{I}$ and $o \in \mathcal{O}$) that lead to q (with the exception of q_0, which is marked by a set of (i, o)s that satisfy the initial conditions of the specification). A play π of $\mathcal{G}$ is an infinite sequence $\pi = q_0 q_1 \ldots \in Q^\omega$ of states with $q_{i+1} = T(q_i, \sigma_i)$ for all $i \geq 0$. In each step of the play, the letter $\sigma_i = (x_i, y_i)$ is chosen in such a way that env first chooses $x_i \in \mathcal{I}$ and then sys chooses $y_i \in \mathcal{O}$. A play is won by the system iff $Win(\pi) = \texttt{true}$.

A finite-memory strategy for env in $\mathcal{G}$ is a tuple $\mathcal{S} = (\Gamma, \gamma_0, \rho)$, where Γ is a finite set representing the memory, $\gamma_0 \in \Gamma$ is in the initial memory content, and $\rho \subseteq Q \times \Gamma \times \mathcal{I} \times \Gamma$ is a relation mapping a state in $\mathcal{G}$ and the memory content to a set of possible next inputs and an updated memory content. A play π conforms to a strategy ρ if and only if there exists sequences $(x_0, y_0)(x_1, y_1) \ldots \in \Sigma^\omega$ and $\gamma_0 \gamma_1 \ldots \in \Gamma^\omega$ such that, for all $i \geq 0$, $(q_i, \gamma_i, x_i, \gamma_{i+1}) \in \rho$ and $q_{i+1} = T(q_i, (x_i, y_i))$. Note that here $\mathcal{S}$ is the strategy for env, so the environment chooses only x_i. $\mathcal{S}$ is winning for env from a state q if all plays starting in q and conforming to $\mathcal{S}$ are won by env. $W^{env} \subseteq Q$ is called the winning region for env, denoting the set of states from which such a winning strategy exists for env. Dually, the winning region for sys is $W^{sys} = Q \setminus W^{env}$. The winning strategy for the environment is called a *counterstrategy* and it exists if $q_0 \in W^{env}$. Könighofer et al. [8] show that a counterstrategy for the environment can be derived from intermediate results in the computation of the winning region W^{env} for GR(1) specifications.

E. Illustrative Example

In this section, we give a simple example of synthesis from GR(1) specifications. We will use the same example to illustrate our HuIL controller synthesis algorithm in Section V as well. Consider the following specification with $I = \{r\}$ and $O = \{g\}$:

- $\psi_0 = \mathbf{G}\,\mathbf{F}\ (r = 0)$ (Environment fairness)
- $\psi_1 = \mathbf{G}\ \big((r = 1) \leftrightarrow (g = 1)\big)$ (System transition)
- $\psi_2 = \mathbf{F}\ (g = 1)$ (System reachability)

where ψ_2 can be rewritten in GR(1) format. The environment assumption ψ_{env} is comprised of ψ_0, while the system guarantees ψ_{sys} includes ψ_1 and ψ_2. It is easy to see that the specification is satisfiable; simply consider an environment that toggles r between 1 and 0. However, it is not realizable because an adversarial environment can always force violation of ψ_{sys} by setting r to 0 at all times, and the system cannot both satisfy ψ_1 and ψ_2.

V. HuIL Controller Synthesis

Given an unrealizable specification, a counterstrategy exists for env. In fact, the counterstrategy summarizes all the moves by env such that it can force a violation of the system guarantees. Our algorithm for synthesizing a HuIL controller thus relies on the insight that we can synthesize an advisory controller that monitors these moves and prompts the human operator with sufficient time ahead of any danger. Jointly, we can also obtain an auto-controller that is correct if the environment turns out to be not as adversarial and not make these moves. The challenge, however, is to decide when an advisory should be sent to the human operator, in a way that it is also *minimally intervening* to the human operator.

A. Weighted Counterstrategy Graph

We consider a representation of the counterstrategy $\mathcal{S}$ as a transition system $\mathcal{V} = (S, S_0, \theta)$. $S = Q \times \Gamma \times \mathcal{I}$ defines the state space, $S_0 \subseteq S$ is the set of initial states, $\theta : S \times \mathcal{O} \rightarrow 2^S$ is the (nondeterministic) transition relation mapping a state in $\mathcal{V}$ given a system output o to a set of possible next states. Intuitively, the states in $\mathcal{V}$ corresponds to a move by the environment given a current state in $\mathcal{G}$, and the transitions correspond to all the possible ways that the system can respond to this move without violating any system guarantees *yet*. Figure 4 shows $\mathcal{V}$ for the GR(1) specifications described in Section IV-E. For convenience, we use assignments to primed variables to represent the moves that env makes next, and do not show the memory content Γ. For a state $s = (q, \gamma, i)$,

we use c_s to denote the number of possible next-inputs that the environment can choose without violating ψ_{env}, including i. In this example, both states have two possible next-inputs (while the counterstrategy only allows one).

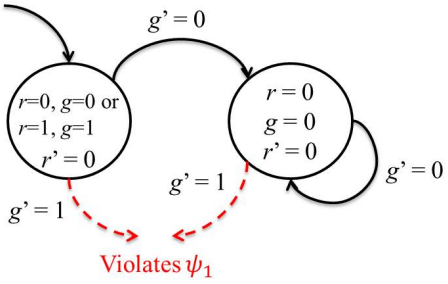


Fig. 4: Summarization of Counterstrategy for the GR(1) Specifications in Section IV-E

$\mathcal{V}$ is a directed graph. We call a node s *failure-imminent* if $\theta(s, o) = \emptyset$ for all $o \in \mathcal{O}$, or s is a part of a strongly connected component (SCC)[2] in $\mathcal{V}$. Intuitively, if there is no outgoing transition from s, it means that sys cannot make a move without violating any ψ^t_{sys} (transition guarantee). If s is part of a SCC, then env is able to force violation of some ψ^f_{sys} (fairness guarantee). In this example, observe that starting at the left state, it takes one time step to reach a *failure-imminent* state, assuming env adheres to the counterstrategy. We use T, as described in Section III-C, to parameterize the minimum number of time steps for a state to reach a failure-imminent state.

In practice, it is not always the case that the environment will behave in the most adversarial way. For example, a car in front may yield if it is blocking our path. Hence, even though the specification is not realizable, it is still important to assess, at any given state, whether it will actually lead to a violation. For simplicity, we assume that the environment will adhere to the counterstrategy once it enters a state belonging to one of the SCCs. Hence, we can convert $\mathcal{V}$ to a directed acyclic graph $\mathcal{V}' = (S', S'_0, \theta')$ by collapsing each SCC to a single node, i.e. $\mathcal{V}'$ is the condensation of $\mathcal{V}$. In this case, S' is the set of SCCs in $\mathcal{V}$, and we label a node in S' *failure-imminent* if the corresponding SCC in $\mathcal{V}$ contains an failure-imminent node. We further augment $\mathcal{V}'$ with a function $\varpi : S' \to \mathbb{Q}$ to encode how likely the state $s \in S'$ will lead to a state that is *failure-imminent*.

Recall from Section III-C that the notion of *minimally-intervening* involves minimizing $\mathcal{C}$, which involves the *probability* that $auto$ is set to `false`, Thus far, we have not associated any probabilities with transitions taken by the system or environment. While our approach can be adapted to work with any assignment of probabilities, for ease of presentation, we make a particular choice in this paper. Specifically, we assume that at any step, the environment can pick an action uniformly at random from the set of possible *legal* actions — those that do not violating any environment assumptions. In our running example, this means that it is equally likely for env to choose $r' = 0$ or $r' = 1$. The reason that $r' = 1$ is absent in the counterstrategy graph shown in Figure 4 is that sys would be able to choose $g' = 1$ to satisfy the system guarantees. In addition, we need to take into account of the cost of having the human operator performing the maneuver instead of the auto-controller. In general, this cost increases with increasing human engagement. Based on these two notions, we define ϖ recursively as follows.

$$\varpi(s) = \begin{cases} 1 & \text{if } s \text{ is } \textit{failure-imminent} \\ \frac{p_s(l_s+1)}{c_s} & \text{Otherwise} \end{cases}$$

where $p_s \in \mathbb{Q}$ is a user-defined penalty parameter, and l_s is the length (number of edges) of the shortest path from s to any failure-imminent node in the DAG. Intuitively, a state far away from any failure-imminent state is less likely to cause an error since the environment would have to make multiple consecutive moves all in an adversarial way. However, if we transfer control at this state, the human operator will have to spend more time controlling the plant. Hence, the penalty term $p_s(l_s + 1)$ is used to reflect this potential overhead. Typically, p_s is chosen in such a way that $\varpi(s) \leq 1$. In the next section, we describe how to use this node-weighted DAG representation of a counterstrategy to derive a HuIL controller that satisfies the criteria established in Section III-C.

B. Counterstrategy-Guided Synthesis

Given an unrealizable GR(1) specification, we can obtain a modified counterstrategy graph $\mathcal{V}'$, which summarizes all possible ways for the environment to force a violation of the system guarantees (reaching a *failure-imminent* node). Hence, if we can eliminate this counterstrategy, then we will obtain an auto-controller that is correct-by-construction. However, since we cannot control how the environment behaves, we would need to actively monitor its changes, and notify the human operator in case the environment behaves according to the counterstrategy. This responsiblity is delegated to an advisory controller. In this section, we describe in detail how we make use of $\mathcal{V}'$ to synthesize the auto-controller, the advisory controller, as well as their interaction with the human operator.

Consider a non-failure-imminent node s in S', s encodes a particular condition where the environment makes a next-move given some last move made by the environment and the system. Assume that some of these next-moves by the environment are disallowed, such that none of the failure-imminent nodes are reachable from any initial state (or source node), then we have effectively eliminated the counterstrategy. This means that if we assert the negation of the corresponding conditions as additional ψ^t_{env} (environment transition assumptions), then we can obtain a realizable specification. Formally, we mine assumptions of the form $\psi_a = \mathbf{G}\,(B_1 \to \neg\mathbf{X}\,B_2)$, where B_1 is a Boolean formula describing a set of assignments over $I \cup O$, and B_2 is a Boolean formula describing a set of

[2] We consider SCCs that contain more than one nodes or one node with an edge to itself.

assignments over I, such that $(\varphi_{env} \wedge \psi_{env}) \rightarrow \psi_{sys}$ is *realizable*, where φ_{env} is a conjunction of ψ_as. Intuitively, contingent upon env not making any of the moves characterized by φ_{env} (in addition to ψ_{env}), we can automatically synthesize an auto-controller that satisfies the system guarantees ψ_{sys}. In addition, computing φ_{env} is *equivalent to* finding a set of nodes in $\mathcal{V}'$ such that, if these nodes are removed from $\mathcal{V}'$, then none of the failure-imminent nodes is reachable from any initial state (source node). We denote such set of nodes as $S^* \subseteq S'$, and the corresponding assumption for each node $s \in S^*$ as ψ_a^s[3].

Recall that we also require a HuIL controller to be *prescient* and *minimally intervening*. Hence, we want to find nodes that reflect these criteria as well. The notion of *prescient* essentially requires that none of the failure-imminent nodes is reachable from any $s \in S^*$ with less than T steps (edges). We use the weight assignment function ϖ to characterize the cost of any failing condition resulting in the advisory controller prompting the human operator to take over control by setting $auto$ to `false`. Formally, we seek S^* such that $\sum_{s \in S^*} \varpi(s)$ is minimized. We can formulate this problem as a *s-t* min-cut problem for directed acyclic graphs. We first compute the subset of nodes S^- in S' that are within T edges of some of the failure-imminent nodes (including the failure-imminent nodes themselves). Then we remove any node in S^- from $\mathcal{V}'$ to form a new DAG $\mathcal{V}^*$. After that, we modify $\mathcal{V}^*$ by adding a new source node that has an outgoing edge to all the source nodes in $\mathcal{V}^*$, and a new terminal node that has an incoming edge from all the sink nodes in $\mathcal{V}^*$. The problem now becomes the *s-t* min-cut problem for node-weighted directed acyclic graph, which can be solved by standard techniques [6]. Figure 5 illustrates this algorithm, and the result of applying it to the running example is shown in Figure 6. The overall approach is summarized in Algorithm 1.

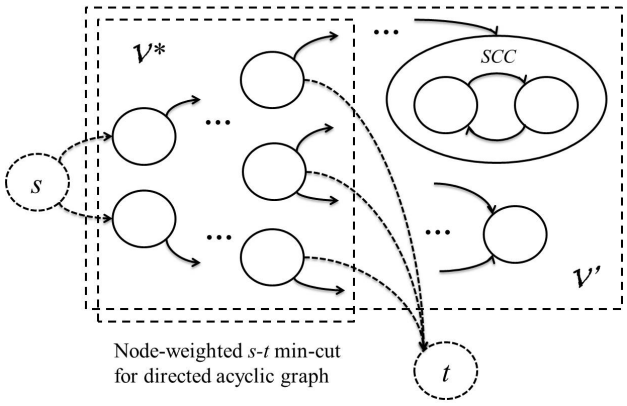


Fig. 5: Advisory Synthesis Formulated as *s-t* Min-Cut in $\mathcal{V}^*$

Theorem 5.1: Given a GR(1) specification ψ and a response time parameter T, Algorithm 1 is guaranteed to either produce a fully autonomous controller satisfying ψ, or a HuIL controller, modeled as a composition of an auto-controller, a human operator and an advisory controller, that is *monitoring*, *prescient* (with parameter T), *minimally intervening*, and *conditionally correct.*[4]

[3]Each node can correspond to one ψ_a or a conjunction of ψ_as.

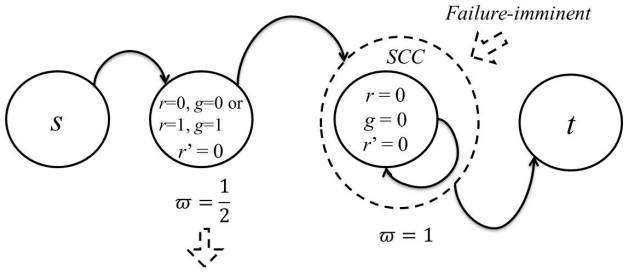


Fig. 6: Illustration of Algorithm for the Running Example

Algorithm 1 Algorithm for Counterstrategy-Guided HuIL Controller Synthesis

Input: GR(1) temporal logic specification ψ.
Input: T : parameter for minimum human response time.
Output: $\mathcal{HIL}$.
if ψ is *realizable* **then**
  Synthesize transducer $M \models \psi$ (using standard GR(1) synthesis);
  $\mathcal{HIL} := M$ (fully autonomous).
else
  Generate counterstrategy $\mathcal{V}$ from ψ;
  Convert $\mathcal{V}$ to $\mathcal{V}^*$ by removing S^- — nodes that are within T steps of failure-imminent nodes;
  Compute node-weighted min-cut S^* in $\mathcal{V}^*$;
  Add assumptions φ_{env} corresponding to S^* to ψ generating new specification $\psi' = \varphi_{env} \rightarrow \psi$;
  Synthesize $\mathcal{AC}$ so that $M \models \psi'$;
  Synthesize $\mathcal{VC}$ as a program that outputs $auto =$ `false` iff the assumption φ_{env} is violated
  $\mathcal{HIL}$ is then a composition of $\mathcal{AC}$, $\mathcal{VC}$ and the human operator as described in Section III-B.
end if

Proof: (Sketch) When ψ is realizable, a fully autonomous controller is synthesized that unconditionally satisfies ψ.

When ψ is not realizable, Algorithm 1 produces a HuIL controller that is *conditionally correct* since the algorithm produces an assumption φ_{env} and an auto-controller that satisfies $\psi' \doteq \varphi_{env} \rightarrow \psi$ and the additional assumptions φ_{env} guarantee that there does not exist a path from any valid initial state to a *failure-imminent* state.

The HuIL controller is *monitoring* as φ_{env} only comprises a set of transitions — expressions involving input and output variables that can be evaluated at run time based on input and

[4]We assume that all failure-imminent nodes are at least T steps away from any initial node. This condition can be checked easily and if it is not satisfied, human should take control immediately.

output values.[5]

It is *prescient* by construction as the $auto$ flag advising the human operator to take over control is set to `false` at least T time steps ahead of any potential failure.

Finally, the HuIL controller is *minimally-intervening*, since $auto$ is set to `false` only when φ_{env} is violated. φ_{env} in turn is constructed based on the minimum node-weighted cut in the counterstrategy graph, which minimizes the cost of switching control from $\mathcal{AC}$ to $\mathcal{HC}$, i.e. $\sum_{s \in S^*} \varpi(s)$ is minimized and $\varphi_{env} = \bigwedge_{s \in S^*} \psi_a^s$.

□

C. Switching from Human Operator to Auto-Controller

Once control has been transferred to the human operator, when should the human yield control to the autonomous controller again? We outline here an approach one can take to address this question, while noting that alternative approaches may exist.

The basic idea in our approach is for the HuIL controller to continue to monitor the environment after the human operator has taken control, checking if a state is reached from which the auto-controller can ensure that it satisfies the specification under assumption φ_{env}, and then take back control.

Recall that the original temporal logic specification ψ is of the form $\psi_{env} \rightarrow \psi_{sys}$ where ψ_{env} is the original set of assumptions on the environment, including on initial states. Algorithm 1 augments ψ_{env} with an additional assumption φ_{env}, to obtain the combined environment assumption $\varphi_{env} \wedge \psi_{env}$. Under this combined assumption, the GR(1) synthesis algorithm is able to extract a winning strategy for the system, which forms the auto-controller. While extracting the winning strategy, we record the set of all states from which the "system" (autonomous controller) can win the game no matter what the environment does for the next T steps (where T is the human response time, as before). This set can be recorded in terms of its characteristic Boolean function, denoted W.

Thus, the HuIL controller continues to monitor the environment and check whether W becomes `true`. If so, the auto-controller notifies the human operator that it is ready to take back control. As long as W remains `true`, the human operator then may return control to the autonomous system, and the auto-controller executes as before.

VI. Experimental Results

In this section, we present several scenarios in the context of autonomous driving and demonstrate the usefulness of our approach in synthesizing human-in-the-loop controllers that satisfy the criteria established in Section III-C. Details of the LTL specifications for each scenario can be found in the Appendices. Our algoritm is implemented on top of the GR(1) synthesis tool RATSY [2].

[5]We assume that the controller can monitor the next assignment of the input variables that the environment is about to make.

A. Car-Following

Recall the car-following example shown in Section II. We describe some of the more interesting specifications below and their corresponding LTL formulas. p_A, p_B, p_C are used to denote the positions of car A, B and C respectively.

- Any position can be occupied by at most one car at a time (no crashing):

$$\mathbf{G}\left(p_A = x \rightarrow (p_B \neq x \wedge p_C \neq x)\right)$$

  where x denotes a position on the discretized space. The cases for B and C are similar, but they are part of ψ_{env}.
- Car A is required to follow car B:

$$\mathbf{G}\left((v_{AB} = \texttt{true} \wedge p_A = x) \rightarrow \mathbf{X}\left(v_{AB} = \texttt{true}\right)\right)$$

  where $v_{AB} = \texttt{true}$ 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,

$$\mathbf{G}\big(((p_C = 5) \wedge (p_A = 6) \wedge (\mathbf{X} p_C = 8)) \rightarrow (\mathbf{X}(p_A \neq 7))\big)$$

  The same requirement for the other positions are similarly specified.

The other specifications can be found in the link described at the beginning of this section. 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 2b.

Applying our algorithm to this (unrealizable) specification with $T = 1$, we obtain the following φ_{env}.

$$\varphi_{env} = \mathbf{G}\left(((p_A = 4) \wedge (p_B = 6) \wedge (p_c = 1)) \rightarrow \mathbf{X}\left((p_B \neq 8) \wedge (p_C \neq 5)\right)\right) \bigwedge$$

$$\mathbf{G}\left(((p_A = 4) \wedge (p_B = 6) \wedge (p_c = 1)) \rightarrow \mathbf{X}\left((p_B \neq 6) \wedge (p_C \neq 3)\right)\right) \bigwedge$$

$$\mathbf{G}\left(((p_A = 4) \wedge (p_B = 6) \wedge (p_c = 1)) \rightarrow \mathbf{X}\left((p_B \neq 6) \wedge (p_C \neq 5)\right)\right) \bigwedge$$

In fact, φ_{env} corresponds to three possible evolutions of the environment from the initial state. In general, φ_{env} can be a conjunction of conditions at different time steps as env and sys progress. The advantage of our approach is that it can produce φ_{env} such that we can synthesize an auto-controller that is guaranteed to satisfy the specification if φ_{env} is not violated, together with an advisory controller that prompts the driver (at least) T (T = 1 in this case) time step ahead of a potential failure when φ_{env} is violated.

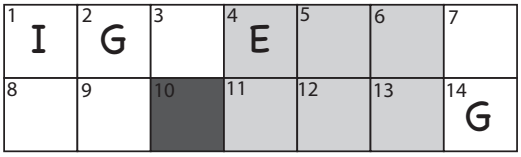


Fig. 7: Gridworld Hallway Example.

B. Gridworld Hallway

This example is a modified version of the grid world hallway example used in [12]. The controllable vehicle (car A) starts from position I as shown in Figure 7. We use p_A the position of car A. We would like to guarantee that car A eventually visits the two goal states G.

$$\mathbf{F}\,(p_A = 2) \wedge \mathbf{F}\,(p_A = 14)$$

We restrict car A to only move in one direction to the right side of the grid, so as it approaches to the right, it won't be able to back up. For example if car A is at position 4, it cannot backup to position 3:

$$\mathbf{G}\,((p_A = 4) \rightarrow \mathbf{X}\,(p_A = 4 \vee p_A = 5 \vee p_A = 11))$$

The uncontrollable vehicle (car B) can only move in the shaded area (regions $\{4, 5, 6, 11, 12, 13\}$) and must leave position E infinitely often. We use p_B to denote position of car B.

$$\mathbf{G}\,(\mathbf{F}\,(p_B \neq 4))$$

There is also a fixed obstacle at position 10. The controllable vehicle cannot synthesize a controller for this example as it has to pass the shaded region to visit its goal at position 14, and the uncontrollable vehicle can mirror the controllable vehicle's movement. For instance, if car A moves into the shaded area at position 4, car B already located at 5 can block its path by staying at 5. If car A moves to 11, then car B can move to 12. The different scenarios in φ_{env} are depicted in Figure 8. Notice that car B can stay in 6 to block car A's path later as car A moves closer to the goal.

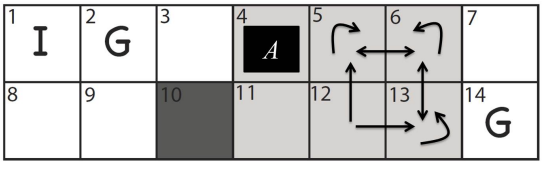


Fig. 8: Illustration of φ_{env}: arrows indicate all possible movements of car B from the current position to the next position.

VII. Related Work

In this paper, we consider the problem of "reactive synthesis" from temporal logic specifications [17], [15] —- generating autonomous machines that satisfy temporal logic specifications for desired output and any given environment input. A major obstacle to adoption of synthesis techniques arises from uncertain or unspecified environment assumptions. In this paper, we consider the problem of detecting unrealizability of a specification and leveraging the presence of a human operator to design a controller that requests human intervention for correct operation. We use assumption mining approaches to extract environment assumptions that make the system realizable and we notify the human user whenever such assumptions are violated. To our knowledge, our work is the first to consider the problem of synthesizing interventions to combine an autonomous controller with a human operator.

Kress-Gazit et al. [9] study synthesis of fully autonomous hybrid controllers that guarantee a desired high level task given a class of admissible environments. In this work, the authors synthesize a discrete controller that satisfies a Linear Temporal Logic formula, and they use the synthesized discrete controller to guide their sensor based controller that results in a hybrid system satisfying the specification.

Livingston et al. [12] and [13] have also studied the synthesis problem for hybrid systems with exploiting the notion of locality that allows "patching" a nominal solution. They update the local parts of the strategy as new data accumulates. This approach allows incremental synthesis and prevents global resynthesis if the nominal plan fails. The patching algorithm considers changes (addition and deletion of edges) in the game graph, and identifies the affected nodes for each system goal and modifies the game graph locally. In our approach, we do not start with a synthesized game graph, and we mine assumptions from counter-strategies and counter-traces instead. Also, the uncertainties we consider are not limited to the topological changes in the system's environment.

In the area of assumption mining, Chatterjee et al. [4] construct a minimal environment assumption based on removing edges from the game graph to ensure safety assumptions and then compute liveness assumptions to put additional fairness constraints on the remaining edges. Finding this minimal set can be shown to be NP-Hard. Our approach in mining environment assumptions is different and is based on removing edges from the counter-strategy. These assumptions are easier to mine and more practical since they can simply be mapped to monitoring real sensors. We also have the freedom of choosing which edges to be removed from the counter-strategy. This flexibility gives us a measure of how conservative we would like to be in our control. Removing edges closer to the source node of the counter-strategy graph prevents any violations in the further future moves and removing edges towards the sink node is less conservative.

Li et al. [11] focus on mining environment assumptions for GR(1) specifications. We base our work on this paper, and use a similar approach for human-in-the-loop control systems. In addition to using the same approach — counter-strategy guided assumption mining — we guarantee to choose the minimal environment assumption for every counter-strategy and we provide formal guarantees on a reasonable advance

warning to the human user about potential failures.

In recent years, there has been an increasing interest in human-in-the-loop systems in the control systems community. The current human-in-the-loop control paradigm studies and learns human models and takes action whenever it concludes that the human user is not capable of controlling the system. Anderson et al. [1] study obstacle avoidance and lane keeping for semiautonomous cars, which is a common example of human-in-the-loop control. The control input of the semiautonomous vehicle is a weighted sum of control input of driver and control input of the autonomous system with weights representing threat functions that only depend on sideslip angles of the vehicle. The autonomous control in this work is a model predictive control which is an iterative, finite-horizon optimization of the plant model. Our approach, unlike this one, seeks to provide correctness guarantees in the form of temporal logic properties.

Verma et al. [20], [21] consider safety and collision avoidance in multivehicle systems, where a human driven vehicle is not communicating with a fully autonomous system. In this scenario, the autonomous vehicle finds the least restrictive safe control action by modeling the human driven car as a hidden mode hybrid system (HMHS) and estimating the state of the human driver. The control of the autonomous vehicle is less conservative and least restrictive since this model does not assume an adversarial human driven car. This approach finds a safe control for a fully autonomous vehicle in a multivehicle situation with non-communicating human driven cars, which is a different situation from the HuIL control in this work.

Vasudevan et al. [19] focus on learning and predicting a human model based on prior observations (past states of the driver, past vehicle trajectories and past generated optimal vehicle trajectories) and current state of the environment. The learned human model is then used in a "vehicle intervention function". Based on the measured level of threat, the controller intervenes and overwrites the driver's input. However, we believe that this paradigm, where the autonomous controller is capable of overriding the human inputs is unsafe in scenarios where the environment is not fully modeled. For that reason, we propose a different paradigm where we allow the human to take control of the vehicle if the autonomous system predicts failure, and for the autonomous vehicle to request back control when it is certain of safe operation.

VIII. Conclusions

In this paper, we propose a synthesis approach for designing human-in-the-loop controllers. We consider a mode of interaction where the controller is mostly autonomous but requires occasional intervention by a human operator, and study important criteria for devising such controllers. Further, we study the problem in the context of controller syntehsis from (unrealizable) temporal-logic specifications. We propose an algorithm based on mining monitorable conditions from the counterstrategy of the unrealizable specifications. Preliminary results on applying this approach to driver assistance in automobiles are encouraging, where useful conditions are generated automatically to ensure the autonomous controller interacts with the human driver in a safe and timely manner. One limitation of the current approach is the use of an explicit counterstrategy graph, while GR(1) synthesis(if the specification is realizable) can be done symbolically. We plan to improve the efficiency of our algorithms for the HuIL controller synthesis problem in the future.

Acknowledgement. This work was supported in part by the TerraSwarm Research Center, one of six centers supported by the STARnet phase of the Focus Center Research Program (FCRP) a Semiconductor Research Corporation program sponsored by MARCO and DARPA. This work was also supported by the NSF grants CCF-1116993 and CCF-1139138.

References

[1] S. J. Anderson, S. C. Peters, T. E. Pilutti, and K. Iagnemma. An optimal-control-based framework for trajectory planning, threat assessment, and semi-autonomous control of passenger vehicles in hazard avoidance scenarios. *International Journal of Vehicle Autonomous Systems*, 8(2):190–216, 2010.

[2] R. Bloem, A. Cimatti, K. Greimel, G. Hofferek, R. Knighofer, M. Roveri, V. Schuppan, and R. Seeber. Ratsy - a new requirements analysis tool with synthesis. In *CAV'10*, pages 425–429, 2010.

[3] R. Bloem, S. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Automatic hardware synthesis from specifications: A case study. In *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, pages 1 –6, Apr. 2007.

[4] K. Chatterjee, T. A. Henzinger, and B. Jobstmann. Environment assumptions for synthesis. In *Proceedings of the 19th international conference on Concurrency Theory*, CONCUR '08, pages 147–161, Berlin, Heidelberg, 2008. Springer-Verlag.

[5] A. Church. Logic, arithmetic and automata. In *Proceedings of the 1962 International Congress of Mathematicians*.

[6] M.-C. Costa, L. Léocart, and F. Roupin. Minimal multicut and maximal integer multiflow: A survey. *European Journal of Operational Research*, 162(1):55 – 69, 2005.

[7] F. A. A. (FAA). The interfaces between flight crews and modern flight systems. http://www.faa.gov/avr/afs/interfac.pdf, 1995.

[8] R. Könighofer, G. Hofferek, and R. Bloem. Debugging formal specifications using simple counterstrategies. In *Formal Methods in Computer-Aided Design, 2009. FMCAD 2009*, pages 152 –159, Nov. 2009.

[9] H. Kress-Gazit, G. Fainekos, and G. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370 –1381, dec. 2009.

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

[11] W. Li, L. Dworkin, and S. Seshia. Mining assumptions for synthesis. In *Formal Methods and Models for Codesign (MEMOCODE), 2011 9th IEEE/ACM International Conference on*, pages 43–50, 2011.

[12] S. C. Livingston, R. M. Murray, and J. W. Burdick. Backtracking temporal logic synthesis for uncertain environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5163–5170. IEEE, 2012.

[13] S. C. Livingston, P. Prabhakar, A. B. Jose, and R. M. Murray. Patching task-level robot controllers based on a local μ-calculus formula. 2012.

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

[15] N. Piterman and A. Pnueli. Synthesis of reactive(1) designs. In *In Proc. Verification, Model Checking, and Abstract Interpretation (VMCAI06*, pages 364–380. Springer, 2006.

[16] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 46–57, 1977.

[17] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '89, pages 179–190, New York, NY, USA, 1989. ACM.

[18] R. Rosner. Modular synthesis of reactive systems. *Ph.D. dissertation, Weizmann Institute of Science*, 1992.
[19] R. Vasudevan, V. Shia, Y. Gao, R. Cervera-Navarro, R. Bajcsy, and F. Borrelli. Safe semi-autonomous control with enhanced driver modeling. In *American Control Conference (ACC), 2012*, pages 2896 –2903, june 2012.
[20] R. Verma and D. Del Vecchio. Semiautonomous multivehicle safety. *IEEE Robotics Automation Magazine*, 18(3):44–54, 2011.
[21] R. Verma and D. Del Vecchio. Safety control of hidden mode hybrid systems. *IEEE Transactions on Automatic Control*, 57(1):62 –77, jan. 2012.
[22] T. Wongpiromsarn, U. Topcu, and R. Murray. Receding horizon temporal logic planning for dynamical systems. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 5997–6004, Dec. 2009.
[23] T. Wongpiromsarn, U. Topcu, and R. Murray. Receding horizon temporal logic planning. *IEEE Transactions on Automatic Control*, 57(11):2817–2830, 2012.

APPENDIX A
CAR FOLLOWING

We reproduce the discretized regions with the starting positions of car A, B and C in Figure 9. Recall that the task of the controller is to enable car A to continuously follow car B, even if car C intervenes.

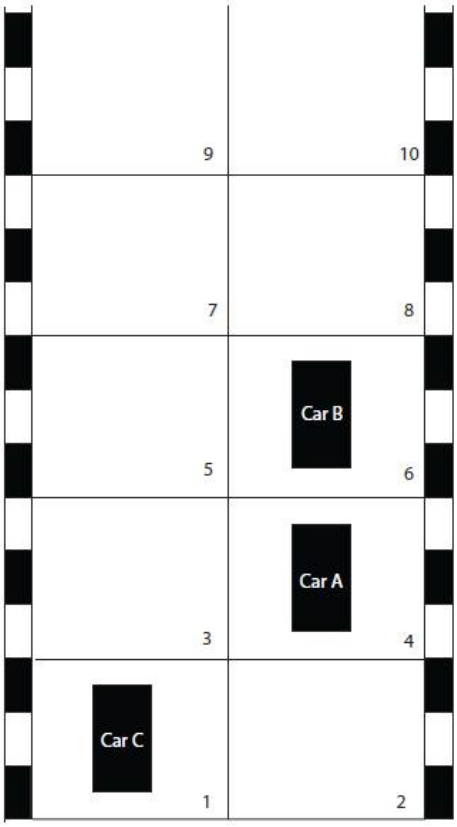


Fig. 9: Car Following Example

A. Input Variables:

1) Position of car B: $p_B \in \{1 \dots 10\}$.
2) Position of car C: $p_C \in \{1 \dots 10\}$.

B. Output Variables:

1) Position of car A: $p_A \in \{1 \dots 10\}$.
2) Visibility of car A: $v_A \subseteq \{1 \dots 10\}$.
3) $follow = \texttt{true}$ iff car A can see the region where car B is in.

C. Environment Assumptions:

1) Initially, car B is in region 6 and car C is in region 1.

$$p_B = 6 \wedge p_C = 1$$

2) Car B can only move at most one square up at each time step. For example, starting at 6, car B can move to 7 or 8, or stay at 6.

$$\mathbf{G}\left((p_A = 6 \rightarrow \mathbf{X}\left(p_A = 6 \vee p_A = 7 \vee p_A = 8\right)\right)$$

3) Car C can move at most two squares up at each time step. For example, starting at 1, car C can move to 3, 4 or 5, or stay at 1.

$$\mathbf{G}\left((p_C = 1 \rightarrow \mathbf{X}\left(p_C = 3 \vee\ p_C = 4 \vee\right.\right.$$
$$\left.\left.p_C = 5 \vee p_C = 1\right)\right)$$

4) Car C does not purposely collide into car A. For example, if car A is at 4, then car C should not move to 4 in the next cycle.

$$\mathbf{G}\left((p_A = 4 \rightarrow \mathbf{X}\left(p_C \neq 4\right)\right)$$

D. System Guarantees:

1) Initially, car A is in region 4.

$$p_A = 4$$

2) Car A must not collide with car B or C.

$$\mathbf{G}\left(p_A \neq p_B \wedge p_A \neq p_C\right)$$

3) Car A can move at most two squares up at each time step. For example, starting at 4, car A can move to 5, 6 or 8, or stay at 4.

$$\mathbf{G}\left((p_A = 4 \rightarrow \mathbf{X}\left(p_A = 5 \vee\ p_A = 6 \vee\right.\right.$$
$$\left.\left.p_A = 8 \vee p_A = 4\right)\right)$$

4) Constraints on the visibility regions of car A. When the view of car A is not obstructed by another vehicle directly in front, car A can see squares ahead include the current position on both lanes. This specification simulates the potential limitation on the sensing capabilities on the vehicle. For example, starting at 4, car A is supposed to be able to see 3, 4, 5, 6, 7 and 8, but due to car B being at 6 (thus partially blocking the view of A), car A can only see regions 3, 4, 5 and 6.

$$\mathbf{G}\left((p_A = 4 \wedge (p_B = 6 \vee p_C = 6)) \rightarrow\right.$$
$$\left.(v_A = \{3, 4, 5, 6\})\right)$$

5) Constraints on $follow$. Basically, car A is said to be able to "follow" car B if it can always move into a position where it can see where car B is at.

$$\mathbf{G}\left((track = \texttt{true}) \leftrightarrow (p_B \in v_A)\right)$$

We further require that $\mathbf{G}\,(follow = \texttt{true})$.

6) Car A cannot change lane if another car parallel to it is changing lane as well. For example, if car A is at 4 and car C is at 3, and car C is moving to 6, then car A cannot move to 5.

$$\mathbf{G}\left((p_A = 4 \wedge p_C = 3 \wedge (\mathbf{X}\, p_C = 6)) \rightarrow (\mathbf{X}\, p_A \neq 5)\right)$$

APPENDIX B
GRIDWORLD HALLWAY

Here we discuss the details of the LTL specifications of the Gridworld Hallway problem described in Section VI. Recall that car A is the controllable vehicle and car B is the uncontrollable vehicle. The goal of this example is for car A to start at position *I* in Figure 7, and eventually visit the two goal states *G* while car B can only move in the shaded area, and has to leave *E* infinitely often.

A. Input Variables:

1) Position of car B: $p_B \in \{4, 5, 6, 11, 12, 13\}$.
2) Position of the obstacle O: $p_O \in \{1 \ldots 14\}$.

B. Output Variables:

1) Position of car A: $p_A \in \{1 \ldots 14\}$.

C. Environment Assumptions:

1) The obstacle O is at a fixed position.

$$\mathbf{G}\,(p_O = 10)$$

2) Car B can only move at most one square in each direction in the shaded area. For example, starting at 12, car B can move to 11, 13 or 5, or stay at 12.

$$\mathbf{G}\left((p_A = 12 \rightarrow \mathbf{X}\,(p_A = 12 \vee p_A = 11 \vee p_A = 13 \vee p_A = 5)\right)$$

3) Car B must leave position *E* infinitely often.

$$\mathbf{G}\,(\mathbf{F}\,(p_B \neq 4))$$

D. System Guarantees:

1) Initially, car A is in region 1.

$$p_A = 1$$

2) Car A must eventually visit the goal states.

$$\mathbf{F}\,(p_A = 2) \wedge \mathbf{F}\,(p_A = 14)$$

3) Car A must not collide with car B.

$$\mathbf{G}\,(p_A \neq p_B)$$

4) Car A must not collide with the obstacle O.

$$\mathbf{G}\,(p_A \neq p_O)$$

5) Car A can move at most one squares to the right side of the grid at each time step, or move one square up or down. For example, given that car A is at 5, it can move to 6, 12, or stay at 5.

$$\mathbf{G}\left((p_A = 5 \rightarrow \mathbf{X}\,(p_A = 5 \vee p_A = 6 \vee p_A = 12)\right)$$