Synthesis for Human-in-the-Loop Control Systems

Wenchao Li[1,⋆], Dorsa Sadigh[2], S. Shankar Sastry[2], and Sanjit A. Seshia[2]

[1] SRI International, Menlo Park, USA
li@csl.sri.com
[2] University of California, Berkeley, USA
{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 (HuIL) 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.

1 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., [1,7]).

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 (LTL) [15]. 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,

⋆ This work was performed when the first author was at UC Berkeley.

E. Ábrahám and K. Havelund (Eds.): TACAS 2014, LNCS 8413, pp. 470–484, 2014.

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 following 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 [13]. 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."* [13]

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

1. *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.
2. *Minimally Intervening*. The controller should only invoke the human operator when it is necessary, and does so in a minimally intervening manner.
3. *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.
4. *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 3. 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 advisory controller is responsible for determining precisely when the human operator should assume control while giving her enough time to respond.

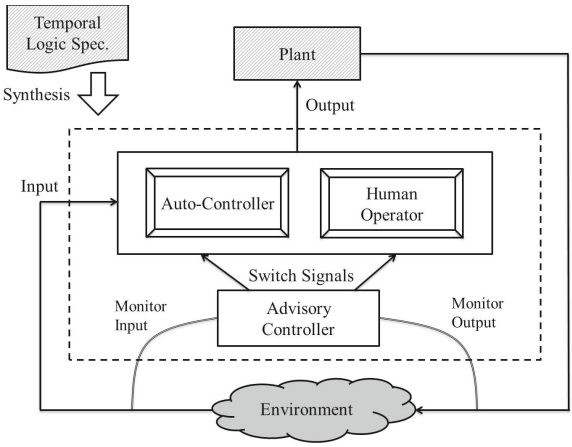


Fig. 1. Human-in-the-Loop Controller: Component Overview

In this paper, we study the construction of such controller in the context of *reactive synthesis* from LTL specifications. *Reactive synthesis* is the process of automatically synthesizing a discrete system (e.g., a finite-state Mealy transducer) that reacts to environment changes in such a way that the given specification (e.g., a LTL formula) is satisfied. There has been growing interest recently in the control and robotics communities (e.g., [20,9]) to apply this approach to automatically generate embedded control software. In summary, the main contributions of this paper are:

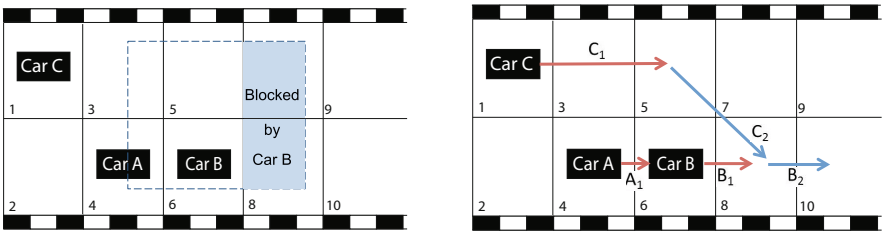
- A formalization of human-in-the-loop control systems and the problem of synthesizing such controllers from high-level specifications, including four key criteria these controllers must satisfy.
- An algorithm for synthesizing human-in-the-loop controllers that satisfy the aforementioned criteria.
-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

The paper is organized as follows. Section 2 describes an motivating example discussing a *car following* example. Section 3 provides a formalism and characterization of the human-in-the-loop controller synthesis problem. Section 4 reviews material on reactive controller synthesis from temporal logic. Section 5 describes our algorithm for the problem. We then present case studies of safety critical driving scenarios in Section 6. Finally, we discuss related work in Section 7 and conclude in Section 8.

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

2 Motivating Example

Consider the example in Figure 2. 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 [21]. 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.



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

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

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 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, human-in-the-loop synthesis 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.

3 Formal Model of HuIL Controller

3.1 Preliminaries

Consider a Booleanized space over the input and output alphabet $\mathcal{X} = 2^X$ and $\mathcal{Y} = 2^Y$, where X and Y are two disjoint sets of variables representing inputs and outputs respectively, we model a discrete controller as a finite-state transducer. A finite-state (Mealy) transducer (FST) is a tuple $M = (Q, q_0, \mathcal{X}, \mathcal{Y}, \rho, \delta)$, where Q is the set of states, $q_0 \in Q$ is the initial state, $\rho : Q \times \mathcal{X} \to Q$ is the transition function, and $\delta : Q \times \mathcal{X} \to \mathcal{Y}$ is the output function. Given an input sequence $\boldsymbol{x} = x_0 x_1 \ldots$, a run of M is the infinite sequence $\boldsymbol{q} = q_0 q_1 \ldots$ of states such that $q_{k+1} = \rho(q_k, i_k)$ for all $k \geq 0$. The run $\boldsymbol{q}$ on $\boldsymbol{x}$ produces the word $M(\boldsymbol{x}) = \delta(q_0, x_0)\delta(q_1, x_1) \ldots$. The language of M is then denoted by the set $\mathcal{L}(M) = \{(x, y)^\omega \mid M(\boldsymbol{x}) = \boldsymbol{y}\}$.

To characterize correctness of M, we assume that we can label if a state is *unsafe* or not, by using a function $\mathcal{F} : Q \to \{\texttt{true}, \texttt{false}\}$, i.e. a state q is failure-prone if $\mathcal{F}(q) = \texttt{true}$. We elaborate on $\mathcal{F}$ later in Section 5.1.

3.2 Agents as Automata

We model two of the three agents in a human-in-the-loop controller, the automatic controller $\mathcal{AC}$ and the advisory controller $\mathcal{VC}$, as finite-state transducers (FSTs). The human operator can be viewed as another FST $\mathcal{HC}$ that uses the same input and output interface as the auto-controller. The overall controller $\mathcal{HUIL}$ is then a composition of the models of $\mathcal{HC}$, $\mathcal{AC}$ and $\mathcal{VC}$.

We use a binary variable $auto$ to denote the internal advisory signal that $\mathcal{VC}$ sends to both $\mathcal{AC}$ and $\mathcal{HC}$. Hence, $\mathcal{X}^{\mathcal{HC}} = \mathcal{X}^{\mathcal{AC}} = \mathcal{X} \cup \{auto\}$, and $\mathcal{Y}^{\mathcal{VC}} = \{auto\}$. 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.

We assume that the human operator (e.g., driver behind the wheel) can take control at any time by transitioning from a "non-active" state to an "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 $\mathcal{HC}$ overwrites the output of $\mathcal{AC}$, i.e. the output of $\mathcal{HC}$ is the output of $\mathcal{HUIL}$. The "overwrite" action happens when a sensor senses the human operator is in control, e.g., putting her hands on the wheel. Similarly, when $active = \texttt{false}$, the output of $\mathcal{HUIL}$ 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 $\mathcal{X}$ to $\mathcal{Y}$. Considering more complex human driver models is left as a future direction [17].

3.3 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 addition, we would like to minimize the need to engage the human operator. Based on the NHTSA statement, we derive four criteria for any effective human-in-the-loop controller, as stated below.

1. *Monitoring.* An advisory $auto$ is issued to the human operator under specific conditions. These conditions in turn need to be determined unambiguously *at runtime*, potentially based on history information but not predictions. In a reactive setting, this means we can use trace information only up to the point when the environment provides a next input from the current state.
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 5.3, 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. 5.1.
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. Thus, we require an advisory to be issued ahead of any failure scenario. In the discrete setting, we assume we are given a positive integer T representing human response time (which can be driver-specific), and require that $auto$ is set to `false` at least T number of transitions ahead of a state (in $\mathcal{AC}$) that is *unsafe*.
4. *Conditionally-Correct.* The auto-controller is responsible for correct operation as long as $auto$ is set to `true`. Formally, if $auto$ = `true` when $\mathcal{AC}$ is at a state q, then $\mathcal{F}(q) =$ `false`. Additionally, when $auto$ is set to `false`, the auto-controller should still maintain correct operation in the next $T-1$ time steps, during or after which we assume the human operator take over control. Formally, if $auto$ changes from `true` to `false` when $\mathcal{AC}$ is at a state q, let $R_T(q)$ be the set of states reachable from q within $T-1$ transitions, then $\mathcal{F}(q') =$ `false`, $\forall q' \in R_T(q)$.

Now we are ready to state the *HuIL Controller Synthesis Problem: Given a model of the system and its specification expressed in a formal language, synthesize a HuIL controller $\mathcal{HuIL}$ that is, by construction, monitoring, minimally intervening, prescient, 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) specifications. We give background on this setting in Section 4, and propose an algorithm for solving the HuIL controller synthesis problem in Section 5.

4 Synthesis from Temporal Logic

4.1 Linear Temporal Logic

An LTL formula is built from atomic propositions AP, Boolean connectives (i.e. negations, conjunctions and disjunctions), and temporal operators $\mathbf{X}$ (*next*) and $\mathbf{U}$ (*until*). In this paper, we consider $AP = X \cup Y$.

LTL formulas are usually interpreted over infinite words (traces) $w \in \Sigma^\omega$, where $\Sigma = 2^{AP}$. The language of an LTL formula ψ is the set of infinite words that satisfy ψ, given by $\mathcal{L}(\psi) = \{w \in \Sigma^\omega \mid w \models \psi\}$. 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.

An LTL formula ψ is *satisfiable* if there exists an infinite word that satisfies ψ, i.e., $\exists w \in (2^{AP})^\omega$ such that $w \models \psi$. A transducer M *satisfies* an LTL formula ψ if $\mathcal{L}(M) \subseteq \mathcal{L}(\psi)$. We write this as $M \models \psi$. *Realizability* is the problem of determining whether there exists a transducer M with input alphabet $\mathcal{X} = 2^X$ and output alphabet $\mathcal{Y} = 2^Y$ such that $M \models \psi$.

4.2 Synthesis from GR(1) Specification

Synthesis is the process of automatically finding an implementation that satisfies a given specification. However, the complexity of deciding the realizability of an LTL formula can be prohibitively high (2EXPTIME-complete [16]). Piterman et al. [14] describe a more efficient algorithm for synthesizing a subclass of LTL properties, known as Generalized Reactivity (1) [GR(1)] formulas. In this paper, we consider (unrealizable) specifications given in the GR(1) subclass. A GR(1) formula has the form $\psi = \psi^{env} \rightarrow \psi^{sys}$, where ψ^{env} represents the *environment assumptions* and ψ^{sys} represents the *system guarantees*. The syntax of GR(1) formulas is given as follows. We require ψ^l for $l \in \{env, sys\}$ to be a conjunction of sub-formulas in the following forms:

- ψ_i^l: a Boolean formula that characterizes the *initial states*.
- ψ_t^l: an LTL formula that characterizes the *transition*, in the form $\mathbf{G}\,B$, where B is a Boolean combination of variables in $X \cup Y$ and expression $\mathbf{X}\,u$ where $u \in X$ if $l = env$ and $u \in X \cup Y$ if $l = sys$.
- ψ_f^l: an LTL formula that characterizes *fairness*, in the form $\mathbf{G}\,\mathbf{F}\,B$, where B is a Boolean formula over variables in $X \cup Y$.

4.3 Games and Strategies

In general, the synthesis problem can be viewed as a two-player game between the system sys and the environment env. Following [14], a finite-state two-player game is defined by its *game graph*, represented by the tuple $\mathcal{G} = (Q^g, \theta^g, \rho^{env}, \rho^{sys}, Win)$ for input variables X controlled by the environment env and output variables Y controlled by the system sys, where $Q^g \subseteq 2^{X \cup Y}$ is the state space of the game, θ^g is a Boolean formula over $X \cup Y$ that specifies the initial states of the game structure, $\rho^{env} \subseteq Q^g \times 2^X$ is the environment transition relation relating a present state in Q^g to the possible next inputs the environment can pick in 2^X, $\rho^{sys} \subseteq Q^g \times 2^X \times 2^Y$ is the system transition relation relating a present state in Q^g and a next input in 2^X picked by the environment to the possible next outputs the system can pick in 2^Y, and Win is the winning condition. Given a set of GR(1) specifications, i.e. $\psi_i^{env}, \psi_i^{sys}, \psi_t^{env}, \psi_t^{sys}, \psi_f^{env}, \psi_f^{sys}$, we can define a game structure $\mathcal{G}$ by setting $\theta^g = \psi_i^{env} \wedge \psi_i^{sys}$, $\rho^{env} = \psi_t^{env}$ with all occurrences of $\mathbf{X}\,u$ replaced by u'[2], $\rho^{sys} = \psi_t^{sys}$ with all occurrences of $\mathbf{X}\,u$ replaced by u', and Win as $\psi_f^{env} \rightarrow \psi_f^{sys}$. A play π of $\mathcal{G}$ is a maximal sequence of states

[2] We use the primed copies u' of u to denote the next input/output variables.

$\pi = q_0 q_1 \ldots$ of states such that $q_0 \models \theta^g$ and $(q_i, q_{i+1}) \in \rho^{env} \wedge \rho^{sys}$ for all $i \geq 0$. A play π is winning for the system iff it is infinite and $\pi \models Win$. Otherwise, π is winning for the environment. The set of states from which there exists a winning strategy for the environment is called the winning region for env.

A finite-memory strategy for env in $\mathcal{G}$ is a tuple $\mathcal{S}^{env} = (\Gamma^{env}, \gamma^{env_0}, \eta^{env})$, where Γ^{env} is a finite set representing the memory, $\gamma^{env_0} \in \Gamma^{env}$ is the initial memory content, and $\eta^{env} \subseteq Q^g \times \Gamma^{env} \times \mathcal{X} \times \Gamma^{env}$ is a relation mapping a state in $\mathcal{G}$ and some memory content $\gamma^{env} \in \Gamma^{env}$ to the possible next inputs the environment can pick and an updated memory content. A strategy $\mathcal{S}^{env}$ is winning for env from a state q if all plays starting in q and conforming to $\mathcal{S}^{env}$ are won by env. Following the terminology used in [8], if a strategy $\mathcal{S}^{env}$ is winning from an initial state q satisfying θ^g, then it is called a *counterstrategy* for env. The existence of a counterstrategy is equivalent to the specification being unrealizable. We refer the readers to [8] for details on how a counterstrategy can be extracted from intermediate results of the fix-point computation for the winning region for env. On the other hand, a winning strategy for the system can be turned into an implementation, e.g., a sequential circuit with $|X|$ inputs, $|X| + |Y|$ state-holding elements (flip-flops), and $|Y|$ outputs that satisfies the given GR(1) specification. In this paper, the synthesized implementation is effectively the auto-controller in the proposed HuIL framework, and can be viewed as a Mealy machine with state space $Q \subseteq 2^{X \cup Y}$. We refer the readers to [14] for details of this synthesis process.

4.4 Counterstrategy Graph

The counterstrategy can be conveniently viewed as a transition system. A *counterstrategy graph* G^c is a discrete transition system $G^c = (Q^c, Q_0^c \subseteq Q^c, \rho^c \subseteq Q^c \times Q^c)$, where $Q^c \subseteq Q^g \times \Gamma^{env}$ is the state space, $Q_0^c = Q_0^g \times \gamma^{env_0}$ is the set of initial states, and $\rho^c = \eta^{env} \wedge \rho^{sys}$ is the transition relation. In a nutshell, G^c describes evolutions of the game state where env adheres to η^{env} and sys adheres to ρ^{sys}. For convenience, we use a function $\theta^c : Q^c \to 2^{X \cup Y}$ to denote the game state (an assignment to X and Y) associated with a state $q^c \in Q^c$. A run π^c of G^c is a maximal sequence of states $\pi^c = q_0^c q_1^c \ldots$ of states such that $q_0^c \in Q_0^c$ and $(q_i^c, q_{i+1}^c) \in \rho^c$ for all $i \geq 0$.

We can also view G^c as a directed graph, where each state in Q^c is given its own node, and there is an edge from node q_i^c to node q_j^c if given the current state at q_i^c, there exists a *next* input picked from the counterstrategy for which the system can produce a legal next output so that the game proceeds to a new state at q_j^c.

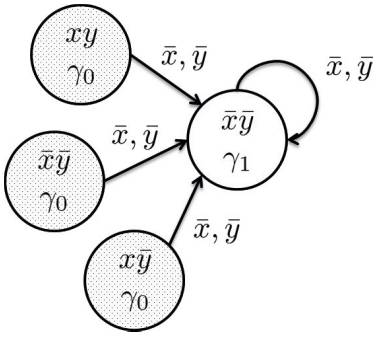
5 HuIL Controller Synthesis

Given an unrealizable specification, a counterstrategy $\mathcal{S}^{env}$ exists for env which describes moves by env such that it can force a violation of the system guarantees. The key insight of our approach for synthesizing a HuIL controller is that we can synthesize an advisory controller that monitors these moves and prompts the human operator with sufficient time ahead of any danger. These moves are essentially assumptions on the environment under which the system guarantees can be ensured. When these assumptions are not violated (the environment may behave in a benign way in reality), the auto-controller fulfills the objective of the controller. On the other hand, if any of the

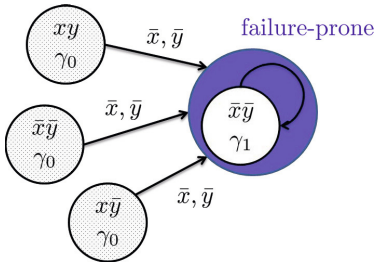
assumptions is violated, as flagged by the advisory controller, then the control is safely switched to the human operator in a way that she can have sufficient time to respond. The challenge, however, is to decide when an advisory should be sent to the human operator, in a way that it is also *minimally intervening* to the human operator. We use the following example to illustrate our algorithm.

Example 1. Consider $X = \{x\}, Y = \{y\}$ and the following GR(1) sub-formulas which together form $\psi = \psi^{env} \rightarrow \psi^{sys}$.

1. $\psi_f^{env} = \mathbf{G}\ (\mathbf{F}\ \neg x)$
2. $\psi_t^{sys} = \mathbf{G}\ (\neg x \rightarrow \neg y)$
3. $\psi_f^{sys} = \mathbf{G}\ (\mathbf{F}\ y)$



(a) Counterstrategy graph G^c for unrealizable specification ψ.



(b) Condensed graph $\hat{G}^c$ for G^c after contracting the SCC.

Specification ψ is not realizable. Figure 3a shows the computed counterstrategy graph G^c. The literal $\bar{x}$ ($\bar{y}$) denotes the negation of the propositional variable x (y). The memory content is denoted by γ_i with γ_0 being the initial memory content. The three shaded states on the left are the initial states. The literals on the edges indicate that the environment first chooses $\bar{x}$ and then the system chooses $\bar{y}$. (the system is forced to pick $\bar{y}$ due to ψ_t^{sys}). Observe that, according the counterstrategy, the system will be forced to pick $\bar{y}$ perpetually. Hence, the other system guarantee ψ_f^{sys} cannot be satisfied.

5.1 Weighted Counterstrategy Graph

Recall that a counterstrategy can be viewed as a discrete transition system or a directed graph G^c. We consider two types of *imminent* failures (violation of some system guarantee specification) described by G^c.

- **Safety violation.** For a node (state) $q_1^c \in Q^c$, if there does not exist a node q_2^c such that $(q_1^c, q_2^c) \in \rho^c$, then we say q_1^c is *failure-imminent*. In this scenario, after env picks a next input according to the counterstrategy, sys cannot find a next output such that all of the (safety) guarantees are satisfied (some ψ_i^{sys} or ψ_t^{sys} is violated).
- **Fairness violation.** If a node q^c is part of a strongly connected component (SCC) scc in Q^c, then we say q^c is *failure-doomed*. For example, the node $(\bar{x}, \bar{y}, \gamma_1)$ in Figure 3a is a failure-doomed node. Starting from q^c, env can always pick inputs in such a way that the play is forced to get stuck in scc. Clearly, all other states in scc are also *failure-doomed*.

Now we make the connection of the labeling function $\mathcal{F}$ for a controller M to the counterstrategy graph G^c which describes behaviors that M should not exhibit. Consider an auto-controller M and a state q (represented by the assignment xy) in M. $\mathcal{F}(q) = \texttt{true}$ if and only if there exist some $q^c \in Q^c$ such that $\theta^c(q^c) = xy$ and q^c is either *failure-imminent* or *failure-doomed*. 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 *failure-doomed* state.

We can convert G^c to its directed acyclic graph (DAG) embedding $\hat{G}^c = (\hat{Q}^c, \hat{Q}^c_0, \hat{\rho}^c)$ by contracting each SCC in G^c to a single node. Figure 3b shows the condensed graph $\hat{G}^c$ of G^c shown in Figure 3a. We use a surjective function $\hat{f} : Q^c \to \hat{Q}^c$ to describe the mapping of nodes from G^c to $\hat{G}^c$. We say a node $\hat{q} \in \hat{Q}^c$ is *failure-prone* if a node $q^c \in Q^c$ is either *failure-imminent* or *failure-doomed* and $\hat{f}(q^c) = \hat{q}$.

Recall from Section 3.3 that the notion of *minimally-intervening* requires the minimization of a cost function $\mathcal{C}$, which involves the *probability* that $auto$ is set to `false`, Thus far, we have not associated any probabilities with transitions taken by the environment or the system. 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 each step, the environment picks a next-input uniformly at random from the set of possible *legal* actions (next-inputs) obtained from η^{env} given the current state. In Example 1 and correspondingly Figure 3a, this means that it is equally likely for env to choose $\bar{x}$ or x from any of the states. We use $c(q)$ to denote the total number of legal actions that the environment can take from a state q.

In addition, we take into account of the cost of having the human operator perform the maneuver instead of the auto-controller. In general, this cost increases with longer human engagement. Based on these two notions, we define ϖ, which assigns a weight to an edge $e \in \hat{Q}^c \times \hat{Q}^c$ in $\hat{G}^c$, recursively as follows. For an edge between $\hat{q}_i$ and $\hat{q}_j$,

$$\varpi(\hat{q}_i, \hat{q}_j) = \begin{cases} 1 & \text{if } \hat{q}_j \text{ is } \textit{failure-prone} \\ \frac{pen(\hat{q}_i) \times len(\hat{q}_i)}{c(\hat{q}_i)} & \text{Otherwise} \end{cases}$$

where $pen : \hat{Q}^c \to \mathbb{Q}^+$ is a user-defined penalty parameter[3], and $len : \hat{Q}^c \to \mathbb{Z}^+$ is the length (number of edges) of the shortest path from a node $\hat{q}_i$ to any failure-prone node in $\hat{G}^c$. Intuitively, a state far away from any failure-prone state is less likely to cause a failure since the environment would need 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 in control, which is not desirable for a HuIL controller. Next, we describe how to use this edge-weighted DAG representation of a counterstrategy graph to derive a HuIL controller that satisfies the criteria established earlier.

5.2 Counterstrategy-Guided Synthesis

Suppose we have a counterstrategy graph G^c that summarizes all possible ways for the environment to force a violation of the system guarantees. Consider an outgoing edge

[3] $pen(\hat{q}_i)$ should be chosen such that $\varpi(\hat{q}_i, \hat{q}_j) < 1$.

from a non-failure-prone node $\hat{q}$ in $\hat{G}^c$ (condensed graph of G^c), this edge encodes a particular condition where the environment makes a next-move given some last move made by the environment and the system. If some of these next-moves by the environment are disallowed, such that none of the failure-prone nodes are reachable from any initial state, 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 $\phi = \bigwedge_i(\mathbf{G}\ (a_i \rightarrow \neg\mathbf{X}\ b_i))$, where a_i is a Boolean formula describing a set of assignments over variables in $X \cup Y$, and b_i is a Boolean formula describing a set of assignments over variables in X.

Under the assumption ϕ, if $(\phi \wedge \psi^{env}) \rightarrow \psi^{sys}$ is realizable, then we can automatically synthesize an auto-controller that satisfies ψ. In addition, the key observation here is that mining ϕ is *equivalent to* finding a set of edges in $\hat{G}^c$ such that, if these edges are removed from $\hat{G}^c$, then none of the failure-prone nodes is reachable from any initial state. We denote such set of edges as E^ϕ, where each edge $e \in E^\phi$ corresponds to a conjunct in ϕ. For example, if we remove the three outgoing edges from the source nodes in Figure 3b, then the failure-prone node is not reachable. Removing these three edges correspond to adding the following environment assumption, which can be monitored at runtime.

$$(\mathbf{G}\ ((x \wedge y) \rightarrow \neg\mathbf{X}\ \bar{x})) \wedge (\mathbf{G}\ ((\bar{x} \wedge \bar{y}) \rightarrow \neg\mathbf{X}\ \bar{x})) \wedge (\mathbf{G}\ ((x \wedge \bar{y}) \rightarrow \neg\mathbf{X}\ \bar{x}))$$

Human factors play an important role in the design of a HuIL controller. The criteria established for a HuIL controller in Section 3.3 also require it to be *prescient* and *minimally intervening*. Hence, we want to mine assumptions that reflect these criteria as well. The notion of *prescient* essentially requires that none of the failure-prone nodes is reachable from a non-failure-prone node with less than T steps (edges). The weight function ϖ introduced earlier can be used to characterize the cost of a failing assumption resulting in the advisory controller prompting the human operator to take over control (by setting $auto$ to `false`). Formally, we seek E^ϕ such that the total cost of switching control $\sum_{e \in E^\phi} \varpi(e)$ is minimized.

We can formulate this problem as a *s-t* min-cut problem for directed acyclic graphs. Given $\hat{G}^c$, we first compute the subset of nodes $\hat{Q}_T^c \subseteq \hat{Q}^c$ that are backward reachable within $T-1$ steps from the set of failure-prone nodes (when $T = 1$, $\hat{Q}_T^c$ is the set of failure-prone node). We assume that $\hat{Q}_0^c \cap \hat{Q}_T^c = \emptyset$. Next, we remove the set of nodes $\hat{Q}_T^c$ from $\hat{G}^c$ and obtain a new graph $\hat{G}_T^c$. Since $\hat{G}_T^c$ is again a DAG, we have a set of source nodes and a set of terminal nodes. Thus, we can formulate a *s-t* min-cut problem by adding a new source node that has an outgoing edge (with a sufficiently large weight) to each of the source nodes and a new terminal node that has an incoming edge (with a sufficiently large weight) from each of the terminal nodes. This *s-t* min-cut problem can be easily solved by standard techniques [6]. The overall approach is summarized in Algorithm 1.

Theorem 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 $\mathcal{AC}$, a human operator and an advisory controller $\mathcal{VC}$, that is* monitoring, prescient *with parameter*

Algorithm 1. Counterstrategy-Guided HuIL Controller Synthesis

Input: GR(1) specification $\psi = \psi^{env} \rightarrow \psi^{sys}$.
Input: T : parameter for minimum human response time.
Output: $\mathcal{AC}$ and $\mathcal{VC}$. $\mathcal{HuIL}$ is then a composition of $\mathcal{AC}$, $\mathcal{VC}$ and $\mathcal{HC}$.
if ψ is *realizable* **then**
Synthesize transducer $M \models \psi$ (using standard GR(1) synthesis);
$\mathcal{HuIL} := M$ (fully autonomous).
else
Generate G^c from ψ (assume a single G^c; otherwise the algorithm is performed iteratively);
Generate the DAG embedded $\hat{G}^c$ from G^c.
Reduce $\hat{G}^c$ to $\hat{G}^c_T$;
Assign weights to $\hat{G}^c$ using φ; by removing $\hat{Q}^c_T$ – nodes that are within $T-1$ steps of any failure-prone node;
Formulate a *s-t* min-cut problem with $\hat{G}^c_T$;
Solve the *s-t* min-cut problem to obtain E^ϕ;
Add assumptions ϕ to ψ to obtain the new specification $\psi_{new} := (\phi \wedge \psi^{env}) \rightarrow \psi^{sys}$;
Synthesize $\mathcal{AC}$ so that $M \models \psi_{new}$;
Synthesize $\mathcal{VC}$ as a (stateless) monitor that outputs $auto = \texttt{false}$ if ϕ is violated.
end if

T, minimally intervening[4] *with respect to the cost function* $f_C = \sum_{e \in E^\phi} \varpi(e)$, *and* conditionally correct[5].

Proof. (Sketch) When ψ is realizable, a fully autonomous controller is synthesized and unconditionally satisfies ψ. Now consider that case when ψ is not realizable.

The HuIL controller is *monitoring* as ϕ only comprises a set of environment transitions up to the next environment input.

It is *prescient* by construction. The $auto$ flag advising the human operator to take over control is set to `false` *precisely* when ϕ is violated. When ϕ is violated, it corresponds to the environment making a next-move from the current state q according to some edge $e = (\hat{q}_i, \hat{q}_j) \in E^\phi$. Consider any $q^c \in Q^c$ such that $\hat{f}(q^c) = \hat{q}_i$, $\theta^c(q^c) = q$. Since $\hat{q}_i \notin \hat{Q}^c_T$ by the construction of $\hat{G}^c_T$, $\hat{q}_i$ is at least T transitions away from any *failure-prone* state in $\hat{G}^c$. This means q^c must also be at least T transitions away from any *failure-imminent* state or *failure-doomed* state in Q^c. Hence, by the definition of $\mathcal{F}$ with respect to a *failure-doomed* or *failure-doomed* state in Section 5.1, q is (and $auto$ is set) at least T transitions ahead of any state that is *unsafe*.

The HuIL controller is also *conditionally correct*. By the same reasoning as above, for any state $q' \in R_T(q)$, $\mathcal{F}(q') = \texttt{false}$, i.e. q' is *safe*.

Finally, since $auto$ is set to `false` precisely when ϕ is violated, and ϕ in turn is constructed based on the set of edges E^ϕ, which minimizes the cost function $f_C = \sum_{e \in E^\phi} \varpi(e)$, the HuIL controller is *minimally-intervening* with respect to the cost function f_C.

[4] We assume the counterstrategy we use to mine the assumptions is an optimal one – it forces a violation of the system guarantees as quickly as possible.

[5] We assume that all failure-prone nodes are at least T steps away from any initial node.

5.3 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? One idea is for the HuIL controller to continually 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), and then the advisory controller can signal a driver telling her that the auto-controller is ready to take back control. We note that alternative approaches may exist and we plan to investigate this further in future work.

6 Experimental Results

Our algorithm is implemented as an extension to the GR(1) synthesis tool RATSY [4]. Due to space constraint, we discuss only the car-following example (as shown in Section 2) here and refer the readers to `http://verifun.eecs.berkeley.edu/tacas14/` for other examples.

Recall the car-following example shown in Section 2. 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:
$$\mathbf{G}\left((v_{AB} = \texttt{true} \wedge p_A = x) \rightarrow \mathbf{X}\,(v_{AB} = \texttt{true})\right)$$
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,
$$\mathbf{G}\left(((p_C = 5) \wedge (p_A = 6) \wedge (\mathbf{X}\,p_C = 8)) \rightarrow (\mathbf{X}\,(p_A \neq 7))\right)$$

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 assumption ϕ.

$$\begin{aligned}\phi = \mathbf{G}\left(((p_A = 4) \wedge (p_B = 6) \wedge (p_C = 1)) \rightarrow \neg\mathbf{X}\left((p_B = 8) \wedge (p_C = 5)\right)\right) \bigwedge \\ \mathbf{G}\left(((p_A = 4) \wedge (p_B = 6) \wedge (p_C = 1)) \rightarrow \neg\mathbf{X}\left((p_B = 6) \wedge (p_C = 3)\right)\right) \bigwedge \\ \mathbf{G}\left(((p_A = 4) \wedge (p_B = 6) \wedge (p_C = 1)) \rightarrow \neg\mathbf{X}\left((p_B = 6) \wedge (p_C = 5)\right)\right)\end{aligned}$$

In fact, ϕ corresponds to three possible evolutions of the environment from the initial state. In general, ϕ 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 such that we can synthesize an auto-controller that is guaranteed to satisfy the specification if ϕ is not violated, together with an advisory controller that prompts the driver (at least) T (T = 1 in this case) time steps ahead of a potential failure when ϕ is violated.

7 Related Work

Similar to [9], we synthesize a discrete controller from temporal logic specifications. Wongpiromsarn et al. [21] consider a receding horizon framework to reduce the synthesis problem to a set of simpler problems for a short horizon. Livingston et al. [11,12] exploit the notion of locality that allows "patching" a nominal solution. They update the local parts of the strategy as new data accumulates allowing incremental synthesis. The key innovation in this paper is that we consider synthesizing interventions to combine an autonomous controller with a human operator.

Our work is inspired by the recent works on assumption mining. Chatterjee et al. [5] construct a minimal environment assumption by removing edges from the game graph to ensure safety assumptions, then compute liveness assumptions to put additional fairness constraints on the remaining edges. Li et al. [10] and later Alur et al. [2] use a counterstrategy-guided approach to mine environment assumptions for GR(1) specifications. We adapt this approach to the synthesis of human-in-the-loop control systems.

In recent years, there has been an increasing interest in human-in-the-loop systems in the control systems community. Anderson et al. [3] study obstacle avoidance and lane keeping for semiautonomous cars. They use a model predictive control for their autonomous control. Our approach, unlike this one, seeks to provide correctness guarantees in the form of temporal logic properties. Vasudevan et al. [19] focus on learning and predicting a human model based on prior observations. Based on the measured level of threat, the controller intervenes and overwrites the driver's input. However, we believe that allowing an auto-controller to override the human inputs is unsafe especially since it is hard to fully model the environment. We propose a different paradigm where we allow the human to take control if the autonomous system predicts failure. Finally, human's reaction time while driving is an important consideration in this paper. The value of reaction time can range from 1 to 2.5 seconds for different tasks and drivers [18].

8 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 synthesis 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. One limitation of the current approach is the use of an explicit counterstrategy graph (due to weight assignment). We plan to explore symbolic algorithms in the future.

Acknowledgment. This work was supported in part by TerraSwarm, one of six centers of STARnet, 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. Federal Aviation Administration. The interfaces between flight crews and modern flight systems (1995)
2. Alur, R., et al.: Counter-strategy guided refinement of gr(1) temporal logic specifications. In: The Conference on Formal Methods in Computer-Aided Design, pp. 26–33 (2013)
3. Anderson, S.J., et al.: 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)
4. Bloem, R., Cimatti, A., Greimel, K., Hofferek, G., Könighofer, R., Roveri, M., Schuppan, V., Seeber, R.: RATSY – A new requirements analysis tool with synthesis. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 425–429. Springer, Heidelberg (2010)
5. Chatterjee, K., Henzinger, T.A., Jobstmann, B.: Environment assumptions for synthesis. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 147–161. Springer, Heidelberg (2008)
6. Costa, M.-C., et al.: Minimal multicut and maximal integer multiflow: A survey. European Journal of Operational Research 162(1), 55–69 (2005)
7. Kohn, L.T., et al.: 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 (2000)
8. Könighofer, R., et al.: Debugging formal specifications using simple counterstrategies. In: Conference on Formal Methods in Computer-Aided Design, pp. 152–159 (2009)
9. Kress-Gazit, H., et al.: Temporal-logic-based reactive mission and motion planning. IEEE Transactions on Robotics 25(6), 1370–1381 (2009)
10. Li, W., et al.: Mining assumptions for synthesis. In: Conference on Formal Methods and Models for Codesign, pp. 43–50 (2011)
11. Livingston, S.C., et al.: Backtracking temporal logic synthesis for uncertain environments. In: Conference on Robotics and Automation, pp. 5163–5170 (2012)
12. Livingston, S.C., et al.: Patching task-level robot controllers based on a local μ-calculus formula. In: Conference on Robotics and Automation, pp. 4588–4595 (2013)
13. National Highway Traffic Safety Administration. Preliminary statement of policy concerning automated vehicles (May 2013)
14. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive(1) designs. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 364–380. Springer, Heidelberg (2006)
15. Pnueli, A.: The temporal logic of programs. In: Annual Symposium on Foundations of Computer Science, pp. 46–57 (1977)
16. Rosner, R.: Modular synthesis of reactive systems. Ph.D. dissertation, Weizmann Institute of Science (1992)
17. Sadigh, D., et al.: Data-driven probabilistic modeling and verification of human driver behavior. In: Formal Verification and Modeling in Human-Machine Systems (2014)
18. Triggs, T.J., et al.: Reaction time of drivers to road stimuli (1982)
19. Vasudevan, R., et al.: Safe semi-autonomous control with enhanced driver modeling. In: American Control Conference, pp. 2896–2903 (2012)
20. Wongpiromsarn, T., et al.: Receding horizon temporal logic planning for dynamical systems. In: Conference on Decision and Control, pp. 5997–6004 (2009)
21. Wongpiromsarn, T., et al.: Receding horizon temporal logic planning. IEEE Transactions on Automatic Control 57(11), 2817–2830 (2012)