Diagnosis and Repair for Synthesis from Signal Temporal Logic Specifications

Shromona Ghosh[§]   Dorsa Sadigh[§]   Pierluigi Nuzzo[§]
Vasumathi Raman[†]   Alexandre Donzé[§]   Alberto Sangiovanni-Vincentelli[§]
S. Shankar Sastry[§]   Sanjit A. Seshia[§]
[§]Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA
[†]United Technologies Research Center, Berkeley, CA

ABSTRACT

We address the problem of diagnosing and repairing specifications for hybrid systems formalized in signal temporal logic (STL). Our focus is on the setting of automatic synthesis of controllers in a model predictive control (MPC) framework. We build on recent approaches that reduce the controller synthesis problem to solving one or more mixed integer linear programs (MILPs), where infeasibility of a MILP usually indicates unrealizability of the controller synthesis problem. Given an infeasible STL synthesis problem, we present algorithms that provide feedback on the reasons for unrealizability, and suggestions for making it realizable. Our algorithms are sound and complete, i.e., they provide a correct diagnosis, and always terminate with a non-trivial specification that is feasible using the chosen synthesis method, when such a solution exists. We demonstrate the effectiveness of our approach on the synthesis of controllers for various cyber-physical systems, including an autonomous driving application and an aircraft electric power system.

1. INTRODUCTION

The automatic synthesis of controllers for hybrid systems from expressive high-level specification languages allows raising the level of abstraction for the designer while ensuring correctness of the resulting controller. In particular, several controller synthesis methods have been proposed for expressive temporal logics and a variety of system dynamics. However, a major challenge for the adoption of these methods in practice is the difficulty of writing correctly formal specifications. Specifications that are poorly stated, incomplete, or inconsistent can produce synthesis problems that are unrealizable (no controller exists for the provided specification), intractable (synthesis is computationally too hard), or lead to solutions that fail to capture the designer's intent. In this paper, we present an algorithmic approach to reduce the specification burden for controller synthesis from temporal logic specifications, focusing on the case when the original specification is unrealizable.

Logical specifications can be provided in multiple ways. One approach is to provide *monolithic* specifications, combining within a single formula constraints on the environment with desired properties of the system under control. In many cases, a system specification can be conveniently provided as a contract to emphasize what are the responsibilities of the system under control (guarantees) versus the assumptions on the external, possibly adversarial, environment [18, 17]. In such a scenario, besides *"weakening" the guarantees*, realizability of a controller can also be achieved by *"tightening" the assumptions*. Indeed, when the specification is unrealizable, it could be either because the environment assumptions are too weak, or the requirements are too strong, or a combination of both. Finding the "problem" with the specification manually can be a tedious and time-consuming process, nullifying the benefits of automatic synthesis. Further, in the *reactive* setting, when the environment is adversarial, finding the right assumptions a priori can be difficult. Thus, given an unrealizable logical specification, there is a need for tools that localize the cause of unrealizability to (hopefully small) parts of the formula, and provide suggestions for repairing the formula in an "optimal" manner.

The problem of diagnosing and repairing formal requirements has received its share of attention in the formal methods community. Ferrère et al. perform diagnosis on faulty executions of systems with specifications expressed in linear temporal logic (LTL) and Metric Temporal Logic (MTL) [9]. They identify the cause of unsatisfiability of these properties in the form of prime implicants, which are conjunctions of literals, and map the failure of a specification to the failure of these prime implicants. Similar syntax tree based definitions of unsatisfiable cores for LTL were presented in [22]. In the context of synthesis from LTL, Raman et al. [20] address the problem of categorizing the causes of unrealizability, and how to detect them in high-level robot control specifications. The use of counter-strategies to derive new environment assumptions for synthesis has also been much studied over the past few years [12, 2, 13]. Our approach, based on exploiting information from optimization solvers, is similar to that taken by Nuzzo et al. [16] to extract unsatisfiable cores for satisfiability modulo theories (SMT) solving.

In this paper, we address the problem of diagnosing and repairing specifications formalized in *signal temporal logic (STL)* [14], a specification language that is well-suited for hybrid systems. Our work is conducted in the setting of

automated synthesis from STL using optimization methods in a model predictive control (MPC) framework [21, 19]. In this approach to synthesis, both the system dynamics and the STL requirements on the system are encoded as mixed integer linear constraints on variables modeling the dynamics of the system and its environment. Controller synthesis is then formulated as an optimization problem to be solved subject to these constraints [21]. In the reactive setting, this approach proceeds by iteratively solving a combination of optimization problems using a *counterexample-guided inductive synthesis* (CEGIS) scheme [19]. In this context, an unrealizable STL specification leads to an infeasible optimization problem. We leverage the ability of existing mixed integer linear programming (MILP) solvers to localize the cause of infeasibility to so-called *irreducibly inconsistent systems* (IIS). Our algorithms use the IIS to localize the cause of unrealizability to the relevant parts of the STL specification. Additionally, we give a method for generating a *minimal set of repairs* to the STL specification such that, after applying those repairs, the resulting specification is realizable. The set of repairs is drawn from a suitably defined space that ensures that we rule out vacuous and other unreasonable adjustments to the specification. Specifically, in this paper, we focus on the numerical parameters in a formula since their specification is often the most tedious and error-prone part. Our algorithms are sound and complete, i.e., they provide a correct diagnosis, and always terminate with a reasonable specification that is realizable using the chosen synthesis method, when such a repair exists in the space of possible repairs.

The problem of infeasibility in constrained predictive control schemes has also been widely addressed in the literature, e.g., by adopting robust MPC approaches, soft constraints, and penalty functions [11, 23, 4]. Rather than tackling general infeasibility issues in MPC, our focus is on providing tools to help debug the controller specification at design time. However, the deployment of robust or soft-constrained MPC approaches can also benefit from our techniques. Our use of MILP does not restrict our method to linear dynamical systems; indeed, we can handle constrained linear and piecewise affine systems, mixed logical dynamical (MLD) systems [3], and certain differentially flat systems. We demonstrate the effectiveness of our approach on the synthesis of controllers for a number of cyber-physical systems, including an autonomous driving application and an aircraft electric power system.

The paper is organized as follows. We begin in Sec. 2 and 3 with preliminaries and a running example. We formally define the diagnosis and repair problems in Sec. 4 and describe our algorithms for both monolithic and contract specifications in Sec. 5 and 6. In Sec. 7 we illustrate our approach on the case studies, and finally conclude in Sec. 8.

2. PRELIMINARIES

In this section, we introduce preliminaries on hybrid dynamical systems, the specification language *Signal Temporal Logic*, and the *Model Predictive Control* framework.

2.1 Hybrid Dynamical Systems

We consider a continuous-time hybrid dynamical system:

$$\dot{x}_t = f(x_t, u_t, w_t)$$
$$y_t = g(x_t, u_t, w_t), \tag{1}$$

where $x_t \in \mathcal{X} \subseteq (\mathbb{R}^{n_c} \times \{0,1\}^{n_l})$ represent the hybrid (continuous and logical) states at time t, $u_t \in \mathcal{U} \subseteq (\mathbb{R}^{m_c} \times \{0,1\}^{m_l})$ are the hybrid control inputs, $y_t \in \mathcal{Y} \subseteq (\mathbb{R}^{p_c} \times \{0,1\}^{p_l})$ are the outputs, and $w_t \in \mathcal{W} \subseteq (\mathbb{R}^{e_c} \times \{0,1\}^{e_l})$ are the hybrid external inputs, including disturbances and other adversarial inputs from the environment. Using a sampling period $\Delta t > 0$, the continuous-time system (1) lends itself to the following discrete-time approximation:

$$x_{k+1} = f_d(x_k, u_k, w_k)$$
$$y_k = g_d(x_k, u_k, w_k), \tag{2}$$

where states and outputs evolve according to time steps $k \in \mathbb{N}$, where $x_k = x(\lfloor t/\Delta t \rfloor) \in \mathcal{X}$. Given that the system starts at an initial state $x_0 \in \mathcal{X}$, a *run* of the system can be expressed as:

$$\xi = (x_0, y_0, u_0, w_0), (x_1, y_1, u_1, w_1), (x_2, y_2, u_2, w_2), \ldots \tag{3}$$

i.e., as a sequence of assignments over the system variables $V = (x, y, u, w)$. A run is, therefore, a *discrete-time signal*. We denote $\xi_k = (x_k, y_k, u_k, w_k)$.

Given an initial state x_0, a finite horizon input sequence $\mathbf{u}^H = u_0, u_1, \ldots, u_{H-1}$, and a finite horizon environment sequence $\mathbf{w}^H = w_0, w_1, \ldots, w_{H-1}$, the finite horizon run of the system modeled by the system dynamics in (2) is uniquely expressed as:

$$\xi^H(x_0, \mathbf{u}^H, \mathbf{w}^H) =$$
$$(x_0, y_0, u_0, w_0), \ldots, (x_{H-1}, y_{H-1}, u_{H-1}, w_{H-1}), \tag{4}$$

where $x_1, \ldots, x_{H-1}$, $y_0, \ldots, y_{H-1}$ are computed using (2). We finally define a finite-horizon cost function $J(\xi^H)$, mapping H-horizon trajectories $\xi^H \in \Xi$ to costs in $\mathbb{R}^+$.

2.2 Signal Temporal Logic

Signal Temporal Logic (STL) was first introduced as an extension of *Metric Interval Temporal Logic (MITL)* to reason about the behavior of real-valued dense-time signals [14]. STL has been largely applied to specify and monitor real-time properties of hybrid systems [8]. Moreover, it offers a robust, quantitative interpretation for the satisfaction of a temporal formula [7, 6], as further detailed below.

An STL formula φ is evaluated on a signal ξ at some time t. We say $(\xi, t) \models \varphi$ when φ evaluates to true for ξ at time t. We instead write $\xi \models \varphi$, if ξ satifies φ at time 0. The atomic predicates of STL are defined by inequalities of the form $\mu(\xi(t)) > 0$, where μ is some function of signal ξ at time t. Specifically, μ is used to denote both the function of $\xi(t)$ and the predicate. Any STL formula φ consists of Boolean and temporal operations on such predicates. The syntax of STL formulae is defined recursively as follows:

$$\varphi ::= \mu \mid \neg\mu \mid \varphi \wedge \psi \mid \mathbf{G}_{[a,b]}\psi \mid \mathbf{F}_{[a,b]}\psi \mid \varphi\,\mathbf{U}_{[a,b]}\psi, \tag{5}$$

where ψ and φ are STL formulae, $\mathbf{G}$ is the *globally* operator, $\mathbf{F}$ is the *finally* operator and $\mathbf{U}$ is the *until* operator. Intuitively, $\xi \models \mathbf{G}_{[a,b]}\psi$ specifies that ψ must hold for signal ξ at all times of the given interval, i.e., $t \in [a, b]$. Similarly

$\xi \models \mathbf{F}_{[a,b]}\psi$ specifies that ψ must hold at some time t' of the given interval. Finally, $\xi \models \varphi\,\mathbf{U}_{[a,b]}\psi$ specifies that φ must hold starting from the current time until a specific time $t \in [a,b]$ at which ψ becomes true. Formally, the satisfaction of a formula φ for a signal ξ at time t is defined as:

$$
\begin{array}{lll}
(\xi,t) \models \mu & \Leftrightarrow & \mu(\xi(t)) > 0 \\
(\xi,t) \models \neg\mu & \Leftrightarrow & \neg((\xi,t) \models \mu) \\
(\xi,t) \models \varphi \wedge \psi & \Leftrightarrow & (\xi,t) \models \varphi \wedge (\xi,t) \models \psi \\
(\xi,t) \models \mathbf{F}_{[a,b]}\varphi & \Leftrightarrow & \exists t' \in [t+a, t+b], (\xi, t') \models \varphi \\
(\xi,t) \models \mathbf{G}_{[a,b]}\varphi & \Leftrightarrow & \forall t' \in [t+a, t+b], (\xi, t') \models \varphi \\
(\xi,t) \models \varphi\,\mathbf{U}_{[a,b]}\,\psi & \Leftrightarrow & \exists t' \in [t+a, t+b] \text{ s.t. } (\xi, t') \models \psi \\
& & \wedge \forall t'' \in [t, t'], (\xi, t'') \models \varphi.
\end{array}
\tag{6}
$$

The *bound* of an STL formula is defined as the maximum over the sums of all nested upper bounds on the temporal operators of the STL formula. For instance, given $\psi = \mathbf{G}_{[0,20]}\mathbf{F}_{[1,6]}\varphi_1 \wedge \mathbf{F}_{[2,25]}\varphi_2$, the *bound* can be calculated as $N = \max(6 + 20, 25) = 26$. An STL formula φ is *bounded-time* if it contains no unbounded operators.

Robust Satisfaction. A *quantitative* or *robust semantics* is defined for an STL formula φ by associating it with a real-valued function ρ^φ of the signal ξ and time t, which provides a "measure" of the margin by which φ is satisfied. Specifically, we require $(\xi,t) \models \varphi$ if and only if $\rho^\varphi(\xi,t) > 0$. The magnitude of $\rho^\varphi(\xi,t)$ can then be interpreted as an estimate of the "distance" of a signal ξ from the set of trajectories satisfying or violating φ.

Formally, the quantitative semantics is defined as follows:

$$
\begin{array}{lll}
\rho^\mu(\xi,t) & = & \mu(\xi(t)) \\
\rho^{\neg\mu}(\xi,t) & = & -\mu(\xi(t)) \\
\rho^{\varphi \wedge \psi}(\xi,t) & = & \min(\rho^\varphi(\xi,t), \rho^\psi(\xi,t)) \\
\rho^{\mathbf{G}_{[a,b]}\varphi}(\xi,t) & = & \min_{t' \in [t+a, t+b]} \rho^\varphi(\xi,t') \\
\rho^{\mathbf{F}_{[a,b]}\varphi}(\xi,t) & = & \max_{t' \in [t+a, t+b]} \rho^\varphi(\xi,t') \\
\rho^{\varphi\mathbf{U}_{[a,b]}\psi}(\xi,t) & = & \max_{t' \in [t+a, t+b]}(\min(\rho^\psi(\xi,t'), \\
& & \qquad \min_{t'' \in [t, t']} \rho^\varphi(\xi,t'')).
\end{array}
\tag{7}
$$

Using the definitions above, the robustness value can then be computed recursively for any STL formula.

2.3   Model Predictive Control

Model Predictive Control (MPC), or *Receding Horizon Control* (RHC), is a well studied hybrid system control method [15, 10]. In RHC, at any time step, the state of the system is observed and an optimization is solved over a finite time horizon H, given a set of constraints and a cost function J. When f, as defined in (2), is nonlinear, we assume optimization is performed at each MPC step after locally linearizing the system dynamics. For example, at time $t = k$, the linearized dynamics around the current state and time are used to compute an optimal strategy $\mathbf{u}_*^H$ over the time interval $[k, k+H-1]$. Only the first component of $\mathbf{u}_*^H$ is, however, applied to the system, while a similar optimization problem is solved at time $k+1$ to compute a new optimal control sequence along the interval $[k+1, k+H]$ for the model linearized around $t = k+1$. While the global optimality of MPC is not guaranteed, the technique is frequently used and performs well in practice.

In this paper, we use STL to express temporal constraints on the environment and system runs for MPC. We then translate an STL specification into a set of mixed integer linear constraints, as further detailed below [21, 19]. Given a formula φ to be satisfied over a finite horizon H, the associated optimization problem has the form:

$$
\begin{array}{ll}
\underset{\mathbf{u}^H}{\text{minimize}} & J(\xi^H(x_0, \mathbf{u}^H)) \\
\text{subject to} & \xi^H(x_0, \mathbf{u}^H) \models \varphi,
\end{array}
\tag{8}
$$

which extracts a control strategy $\mathbf{u}^H$ that minimizes the cost function $J(\xi^H)$ over the finite-horizon trajectory ξ^H, while satisfying the STL formula φ at time step 0. In a closed-loop setting, we compute a fresh $\mathbf{u}^H$ at every time step $i \in \mathbb{N}$, replacing x_0 with x_i in (8) [21, 19].

While (8) applies to systems without uncontrolled inputs, a more general formulation can be provided to account for an uncontrolled disturbance input $\mathbf{w}^H$ that can act, in general, adversarially. To provide this formulation, we assume that the specification is given in the form of an STL *assume-guarantee (A/G) contract* [18, 17] $\mathcal{C} = (V, \varphi_e, \varphi \equiv \varphi_e \rightarrow \varphi_s)$, where V is the set of variables, φ_e captures the assumptions (admitted behaviors) over the (uncontrolled) environment inputs w, and φ_s describes the guarantees (promised behaviors) over all the system variables. A game-theoretic formulation of the controller synthesis problem can then be represented as a *minimax* optimization problem:

$$
\begin{array}{ll}
\underset{\mathbf{u}^H}{\text{minimize}} \quad \underset{\mathbf{w}^H \in \mathcal{W}^e}{\text{maximize}} & J(\xi^H(x_0, \mathbf{u}^H, \mathbf{w}^H)) \\
\text{subject to} \quad \forall \mathbf{w}^H \in \mathcal{W}^e & \xi^H(x_0, \mathbf{u}^H, \mathbf{w}^H) \models \varphi,
\end{array}
\tag{9}
$$

where we aim to find a strategy $\mathbf{u}^H$ that minimizes the worst case cost $J(\xi^H)$ over the finite horizon trajectory, under the assumption that the disturbance signal $\mathbf{w}^H$ acts adversarially. We use $\mathcal{W}^e$ in (9) to denote the set of disturbances that satisfy the environment specification φ_e, i.e., $\mathcal{W}^e = \{\mathbf{w} \in \mathcal{W}^H | \mathbf{w} \models \varphi_e\}$.

Mixed Integer Linear Program Formulation. To solve the control problems in (8) and (9) the STL formula φ can be translated into a set of mixed integer constraints, thus reducing the optimization problem to a *Mixed Integer Program* (MIP). Specifically, in this paper, we consider control problems that can be encoded as *Mixed Integer Linear Programs* (MILP).

The MILP constraints are constructed recursively on the structure of the STL specification, and express the robust satisfaction value of the formula. We see from Section 2.2 that the robustness value of formulae with temporal and Boolean operators is expressed as the *min* or *max* of the robustness values of the operands over time. We then demonstrate the encoding of the *min* operator. Given $min(\rho^{\varphi_1}, \ldots, \rho^{\varphi_n})$, we introduce Boolean variables z^{φ_i} for $i \in \{1, \ldots, n\}$ and a continuous variable p. The resulting MILP constraints are:

$$
p \le \rho^{\varphi_i}, \quad \sum_{i=1 \ldots n} z^{\varphi_i} \ge 1
\tag{10}
$$
$$
\rho^{\varphi_i} - (1 - z^{\varphi_i})M \le p \le \rho^{\varphi_i} + (1 - z^{\varphi_i})M
$$

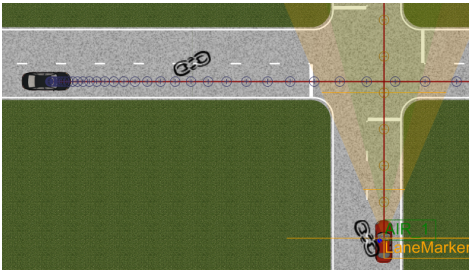where M is a constant selected to be much larger than $|\rho^{\varphi_i}|$

Figure 1: Vehicles crossing an intersection. The red car is the *ego* vehicle, while the black car is part of the environment.

for all i, and $i \in \{1, \ldots, n\}$. The above constraints ensure that p takes the value of the minimum robustness and $z^{\varphi_i} = 1$ if ρ^{φ_i} is the minimum. To get the constraints for *max*, we replace $\leq$ by $\geq$ in (10).

We solve the MILP with an off-the-shelf solver. If the receding horizon scheme is feasible, then the controller synthesis problem is *realizable*, i.e., the algorithm returns a controller that satisfies the specification and optimizes the objective. However, if the MILP is infeasible, the synthesis problem is *unrealizable*. In this case, the failure to synthesize a controller may well be attributed to just a portion of the STL specification. In the rest of the paper we discuss how infeasibility of the MILP constraints can be used to infer the "cause" of failure and, consequently, diagnose and repair the original STL specification.

3. A RUNNING EXAMPLE

To illustrate our approach, we introduce a running example from the autonomous driving domain. As shown in Fig. 1, we consider a scenario in which two moving vehicles approach an intersection. The red car, labeled the *ego* vehicle, is the vehicle under control, while the black car is part of the external environment and may behave, in general, adversarially. The state of the system includes the position and velocity of each vehicle, the control input is the acceleration of the *ego* vehicle, and the environment input is the acceleration of the other vehicle, i.e.,

$$\tilde{x}_t = (x_t^{\mathrm{ego}}, y_t^{\mathrm{ego}}, v_t^{\mathrm{ego}}, x_t^{\mathrm{adv}}, y_t^{\mathrm{adv}}, v_t^{\mathrm{adv}})$$
$$u_t = a_t^{\mathrm{ego}} \quad w_t = a_t^{\mathrm{adv}}. \tag{11}$$

We also assume the dynamics of the system is given by a simple double integrator for each vehicle, e.g.,

$$\begin{bmatrix} \dot{x}^{\mathrm{ego}} \\ \dot{y}^{\mathrm{ego}} \\ \dot{v}^{\mathrm{ego}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x^{\mathrm{ego}} \\ y^{\mathrm{ego}} \\ v^{\mathrm{ego}} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u. \tag{12}$$

A similar equation holds for the environment vehicle which is, however, constrained to move along the horizontal axis. We assume the *ego* vehicle is initialized at the coordinates $(0, -1)$ and the other vehicle is initialized at $(-1, 0)$. We further assume all the units in this example follow the metric system. We would like to design a controller for the *ego* vehicle to satisfy an STL specification under some assumptions on the external environment, and provide diagnosis and feedback if the specification is infeasible. We discuss the following three scenarios.

EXAMPLE 1 (COLLISION AVOIDANCE). *We want to avoid a collision between the* ego *and the adversary vehicle.*

In this example, we assume the environment vehicle's acceleration is fixed at all times, i.e., $a_t^{\mathrm{adv}} = 2$, while the initial velocities are $v_0^{\mathrm{adv}} = 0$ and $v_0^{\mathrm{ego}} = 0$. We encode our requirements using the formula $\varphi := \varphi_1 \wedge \varphi_2$, where φ_1 and φ_2 are defined as follows:

$$\varphi_1 = \mathbf{G}_{[0,\infty)} \neg \big((-0.5 \leq y_t^{\mathrm{ego}} \leq 0.5) \wedge (-0.5 \leq x_t^{\mathrm{adv}} \leq 0.5) \big),$$
$$\varphi_2 = \mathbf{G}_{[0,\infty)} \big(1.5 \leq a_t^{\mathrm{ego}} \leq 2.5 \big). \tag{13}$$

We prescribe bounds on the system acceleration, and state that both cars should never be confined together within a box of width 1 around the intersection $(0,0)$ to avoid a collision.

EXAMPLE 2 (NON-ADVERSARIAL RACE). *We discuss a race scenario, in which the* ego *vehicle must increase its velocity to exceed 0.5 whenever the adversary's initial velocity exceeds 0.5. We then formalize our requirement as a contract $(\psi_e, \psi_e \to \psi_s)$, where ψ_e are the assumptions made on the environment and ψ_s are the guarantees of the system provided the environment satisfies the assumptions. Specifically:*

$$\psi_e = (v_0^{\mathrm{adv}} \geq 0.5),$$
$$\psi_s = \mathbf{G}_{[0,\infty)} (-1 \leq a_t^{\mathrm{ego}} \leq 1) \wedge (v_t^{\mathrm{ego}} \geq 0.5). \tag{14}$$

The initial velocities are $v_0^{\mathrm{adv}} = 0.55$ and $v_0^{\mathrm{ego}} = 0$, while the environment vehicle's acceleration is $a_t^{\mathrm{adv}} = 1$ at all times. We also require the acceleration to be bounded by 1.

EXAMPLE 3 (ADVERSARIAL RACE). *We discuss another race scenario, in which the environment vehicle acceleration a_t^{adv} is no longer fixed, but can vary up to a maximum value of 2. Initially, $v_0^{\mathrm{adv}} = 0$ and $v_0^{\mathrm{ego}} = 0$ hold. Under these assumptions, we would like to guarantee that the velocity of the* ego *vehicle exceeds 0.5 if the speed of the adversary vehicle exceeds 0.5, while maintaining an acceleration in the $[-1, 1]$ range. Altogether, we capture the requirements above via a contract $(\phi_w, \phi_w \to \phi_s)$, where:*

$$\phi_w = \mathbf{G}_{[0,\infty)} \big(0 \leq a_t^{\mathrm{adv}} \leq 2 \big),$$
$$\phi_s = \mathbf{G}_{[0,\infty)} \big((v_t^{\mathrm{adv}} > 0.5) \to (v_t^{\mathrm{ego}} > 0.5) \big) \wedge \big(|a_t^{\mathrm{ego}}| \leq 1 \big). \tag{15}$$

4. PROBLEM STATEMENT

In this section, we define the problems of specification diagnosis and repair in the context of controller synthesis from STL. We assume the discretized system dynamics f_d and g_d, the initial state x_0, the STL specification φ, and a cost function J are given. Then, the *controller synthesis* problem denoted as $\mathcal{P} = (f_d, g_d, x_0, \varphi, J)$ translates into solving (8) (when φ is a monolithic specification of the desired system behaviors) or (9) (when φ represents a contract between the system and the environment).

If synthesis fails, the *diagnosis* problem is, intuitively, to return an explanation in the form of a subset of the original problem constraints that are already infeasible when taken alone. The *repair* problem is to return a "minimal" set of changes to the specification that would render the resulting controller synthesis problem feasible. To diagnose and repair an STL formula, we focus on its sets of atomic predicates

and time intervals of the temporal operators. We then start by providing a definition of the *support* of its atomic predicates, i.e., the set of times at which the value of a predicate affects satisfiability of the formula, and a notion for the set of repairs that we allow.

DEFINITION 1 (SUPPORT). *The* support *of a predicate μ in an STL formula φ is the set of times t such that $\mu(\xi(t))$ appears in φ.*

For example, given $\varphi = \mathbf{G}_{[6,10]}(x_t > 0.2)$, the support of predicate $\mu = (x_t > 0.2)$ is the time interval $[6, 10]$.

DEFINITION 2 (ALLOWED REPAIRS). *Let Φ denote the set of all possible STL formulae. A* repair action *is a relation $\gamma : \Phi \to \Phi$ consisting of the union of the following:*

- *A* predicate repair *returns the original formula after modifying one of its atomic predicates μ to μ^*. We denote this sort of repair by $\varphi[\mu \mapsto \mu^*] \in \gamma(\varphi)$;*

- *A* time interval repair *returns the original formula after replacing the interval of a temporal operator. This is denoted $\varphi[\Delta_{[a,b]} \mapsto \Delta_{[a^*,b^*]}] \in \gamma(\varphi)$ where $\Delta \in \{\mathbf{G}, \mathbf{F}, \mathbf{U}\}$.*

Repair actions can be composed to get a *sequence of repairs* $\Gamma = \gamma_n(\gamma_{n-1}(\dots(\gamma_1(\varphi))\dots))$. Given an STL formula φ, we denote as $\mathtt{REPAIR}(\varphi)$ the set of all possible formulae obtained through compositions of allowed repair actions on φ. Moreover, given a set of atomic predicates $\mathcal{D}$ and a set of time intervals $\mathcal{T}$, we use $\mathtt{REPAIR}_{\mathcal{T},\mathcal{D}}(\varphi) \subseteq \mathtt{REPAIR}(\varphi)$ to denote the set of repair actions that act only on predicates in $\mathcal{D}$ or time intervals in $\mathcal{T}$. We are now ready to provide the formulation of the problems addressed in the paper, both in terms of diagnosis and repair of a *monolithic* specification φ (*general diagnosis and repair*) and an A/G contract $(\varphi_e, \varphi_e \to \varphi_s)$ (*contract diagnosis and repair*).

PROBLEM 1 (GENERAL DIAGNOSIS AND REPAIR).
Given a controller synthesis problem $\mathcal{P} = (f_d, g_d, x_0, \varphi, J)$ such that (8) is infeasible, find:

- *A set of atomic predicates $\mathcal{D} = \{\mu_1, \dots, \mu_d\}$ or time intervals $\mathcal{T} = \{\tau_1, \dots, \tau_d\}$ of the original formula φ,*

- $\varphi' \in \mathtt{REPAIR}_{\mathcal{T},\mathcal{D}}(\varphi)$,

such that $\mathcal{P}' = (f_d, g_d, x_0, \varphi', J)$ is feasible, and the following minimality conditions hold:

- *(predicate minimality) if φ' is obtained by predicate repair[1], $s_i = \mu_i^* - \mu_i$ for $i \in \{1, \dots, d\}$, $s_{\mathcal{D}} = (s_1, \dots, s_d)$, and $\|\cdot\|$ is a norm on $\mathbb{R}^d$, then*

$$\nexists (\mathcal{D}', s_{\mathcal{D}'}) \quad \text{s.t.} \quad \|s_{\mathcal{D}'}\| \leq \|s_{\mathcal{D}}\| \tag{16}$$

and $\mathcal{P}'' = (f_d, g_d, x_0, \varphi'', J)$ is feasible, with $\varphi'' \in \mathtt{REPAIR}_{\mathcal{D}'}(\varphi)$.

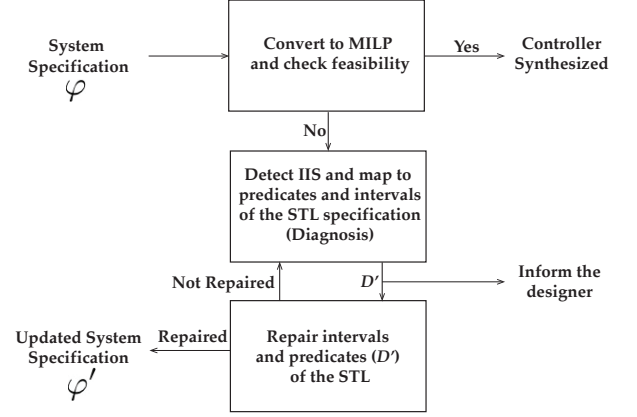[1] For technical reasons, our minimality conditions are predicated on a single type of repair being applied to obtain φ'.

Figure 2: Diagnosis and repair flow diagram.

- *(time interval minimality) if φ' is obtained by time interval repair, $\mathcal{T}^* = \{\tau_1^*, \dots, \tau_l^*\}$ are the non-empty repaired intervals, and $\|\tau\|$ is the length of interval τ:*

$$\nexists \mathcal{T}' = \{\tau_1', \dots, \tau_l'\}, \text{ s.t. } \exists i \in \{1, \dots, l\}, \|\tau_i^*\| \leq \|\tau_i'\| \tag{17}$$

and $\mathcal{P}'' = (f_d, g_d, x_0, \varphi'', J)$ is feasible, with $\varphi'' \in \mathtt{REPAIR}_{\mathcal{T}'}(\varphi)$.

PROBLEM 2 (CONTRACT DIAGNOSIS AND REPAIR).
Given a controller synthesis problem $\mathcal{P} = (f_d, g_d, x_0, \varphi \equiv \varphi_e \to \varphi_s, J)$ such that (9) is infeasible, find:

- *Sets of atomic predicates $\mathcal{D}_e = \{\mu_1^e, \dots, \mu_d^e\}$, $\mathcal{D}_s = \{\mu_1^s, \dots, \mu_{\bar{d}}^s\}$ or sets of time intervals $\mathcal{T}_e = \{\tau_1^e, \dots, \tau_l^e\}, \mathcal{T}_s = \{\tau_1^s, \dots, \tau_{\bar{l}}^s\}$, respectively, of the original formulas φ_e and φ_s,*

- $\varphi_e' \in \mathtt{REPAIR}_{\mathcal{T}_e,\mathcal{D}_e}(\varphi_e)$, $\varphi_s' \in \mathtt{REPAIR}_{\mathcal{T}_s,\mathcal{D}_s}(\varphi_s)$.

such that $\mathcal{P}' = (f_d, g_d, x_0, \varphi', J)$ is feasible, and $\mathcal{D} = \mathcal{D}_e \cup \mathcal{D}_s$, $\mathcal{T} = \mathcal{T}_e \cup \mathcal{T}_s$, and φ' satisfy the minimality conditions of Problem (1).

In the following sections, we discuss our solutions to the above problems.

5. MONOLITHIC SPECIFICATIONS

The scheme adopted to diagnose inconsistencies in the specification and provide constructive feedback to the designer is pictorially represented in Fig. 2. In this section we find a solution for Problem 1, as summarized in Algorithm 1. Given a problem $\mathcal{P}$, defined as in Section 4, $\mathtt{GenMILP}$ reformulates (8) in terms of the following MILP:

$$\begin{aligned} \underset{\mathbf{u}^H}{\text{minimize}} \quad & J(\xi^H) \\ \text{subject to} \quad & f_i^{\mathrm{dyn}} \leq 0 \qquad i \in \{1, \dots, m_d\} \\ & f_k^{\mathrm{stl}} \leq 0 \qquad k \in \{1, \dots, m_s\}, \end{aligned} \tag{18}$$

where f^{dyn} and f^{stl} are mixed integer linear constraint functions over the states, outputs, and inputs of the finite horizon trajectory ξ^H associated, respectively, with the system dynamics and the STL specification φ. We let (J, C) represent this MILP, where J is the objective, and C is the set of

Algorithm 1 DiagnoseRepair

1: **procedure** DiagnoseRepair
2:     **Input:** $\mathcal{P}$
3:     **Output:** $\mathbf{u}^H$, $\mathcal{D}$, $repaired$, φ'
4:     $(J, C) \leftarrow$ GenMILP($\mathcal{P}$), $repaired \leftarrow 0$
5:     $\mathbf{u}^H \leftarrow$ Solve(J, C)
6:     **if** $\mathbf{u}^H = \emptyset$ **then**
7:         $\mathcal{D} \leftarrow \emptyset$, $\mathcal{S} \leftarrow \emptyset$, $I \leftarrow \emptyset$, $\mathcal{M} \leftarrow (0, C)$
8:         **while** $repaired = 0$ **do**
9:             $(\mathcal{D}', \mathcal{S}', I') \leftarrow$ Diagnosis($\mathcal{M}, \mathcal{P}$)
10:             $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}'$, $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}'$, $I \leftarrow I \cup I'$
11:             $options \leftarrow$ UserInput($\mathcal{D}'$)
12:             $\lambda \leftarrow$ ModifyConstraints(I', $options$)
13:             $(repaired, \mathcal{M}, \varphi') \leftarrow$ Repair($\mathcal{M}, I', \lambda, \mathcal{S}, \varphi$)
14:         $\mathbf{u}^H \leftarrow$ Solve($J, \ \mathcal{M}.C$)

Algorithm 2 Diagnosis

1: **procedure** Diagnosis($\mathcal{M}, \mathcal{P}$)
2:     **Input:** $\mathcal{M}, \mathcal{P}$
3:     **Output:** $\mathcal{D}, \mathcal{S}, I'$
4:     $I_C \leftarrow$ IIS($\mathcal{M}$)
5:     $(\mathcal{D}, \mathcal{S}) \leftarrow$ ExtractPredicates($I_C, \mathcal{P}$)
6:     $I' \leftarrow$ ExtractConstraints($\mathcal{M}, \mathcal{D}$)

---

constraints. If problem (18) is infeasible, we iterate between diagnosis and repair phases until the repaired feasible specification φ' is obtained. We let $\mathcal{D}$ and I denote, respectively, the set of predicates returned by the diagnosis procedure, and the constraints corresponding to those predicates.

Optionally, we support an interactive repair mechanism, where the designer provides a set of *options* that prioritize which predicates to modify (UserInput procedure) and get converted into a set of weights λ (ModifyConstraints routine). The designer can then leverage this weighted-cost variant of the problem to define "soft" and "hard" constraints in the controller synthesis problem. In the following, we detail the operation of the Diagnosis and Repair routines.

5.1 Diagnosis

Our diagnosis procedure is summarized in Algorithm 2. Diagnosis receives as inputs the controller synthesis problem $\mathcal{P}$ and an associated MILP formulation $\mathcal{M}$. $\mathcal{M}$ can either be the *feasibility problem* associated with the original problem (18), or a relaxation of it. This feasibility problem has the same, possibly relaxed, constraints as (18) but zero cost. Formally, we provide the following definition of relaxed constraint and relaxed optimization problem.

DEFINITION 3 (RELAXED PROBLEM). *We say that a constraint $f' \leq 0$ is a* relaxed version *of $f \leq 0$ if there exists a slack variable $s \in \mathbb{R}^+$ such that $f' = (f - s)$. In this case, we also say that $f \leq 0$ is relaxed into $f' \leq 0$. Then, an optimization problem $\mathcal{O}'$ is a* relaxed version *of another optimization problem $\mathcal{O}$ if it is obtained from $\mathcal{O}$ by relaxing at least one of its constraints.*

When $\mathcal{M}$ is infeasible, we rely on the capability of state-of-the-art MILP solvers to provide an *Irreducibly Inconsistent System* (IIS) [1, 5] of constraints I_C, defined as follows.

DEFINITION 4 (IRREDUCIBLY INCONSISTENT SYSTEM). *Given a feasibility problem $\mathcal{M}$ with constraint set C, an* Irreducibly Inconsistent System I_C *is a subset of constraints $I_C \subseteq C$ such that: (i) the optimization problem $(0, I_C)$ is infeasible; (ii) $\forall\, c \in I_C$, problem $(0, I_C \setminus \{c\})$ is feasible.*

In other words, an IIS is an infeasible subset of constraints that becomes feasible if any single constraint is removed. For each constraint in I_C, ExtractPredicates traces back the STL predicate(s) originating it, which will be used to construct the set $\mathcal{D} = \{\mu_1, \ldots, \mu_d\}$ of STL atomic predicates in Problem 1, and the corresponding set of support intervals $\mathcal{S} = \{\sigma_1, \ldots, \sigma_d\}$ (adequately truncated to the current horizon H), as obtained from the STL syntax tree. $\mathcal{D}$ will be used to produce a relaxed version of $\mathcal{M}$ as further detailed in Section 5.2. For this purpose, the procedure also returns the subset I of all the constraints in $\mathcal{M}$ that are associated with the predicates in $\mathcal{D}$.

5.2 Repair

The diagnosis procedure isolates a set of STL atomic predicates that jointly produce a reason of infeasibility for the synthesis problem. For repair, we are instead interested in how to modify the original formula to make the problem feasible. The repair procedure is summarized in Algorithm 3. We formulate relaxed versions of the feasibility problem $\mathcal{M}$ associated with problem (18) by using *slack variables*.

Let f_i, $i \in \{1, \ldots, m\}$ denote both of the categories of constraints f^{dyn} and f^{stl} in the feasibility problem $\mathcal{M}$. We reformulate $\mathcal{M}$ into the following *slack feasibility problem*:

$$
\begin{aligned}
\underset{\mathbf{s} \in \mathbb{R}^{|I|}}{\text{minimize}} \quad & ||\mathbf{s}|| \\
\text{subject to} \quad & f_i - s_i \leq 0 && i \in \{1, \ldots, |I|\} \\
& f_i \leq 0 && i \in \{|I| + 1, \ldots, m\} \\
& s_i \geq 0 && i \in \{1, \ldots, |I|\},
\end{aligned}
\tag{19}
$$

where $\mathbf{s} = s_1 ... s_{|I|}$ is a vector of slack variables added to the subset of optimization constraints I, as obtained after the latest call of Diagnosis, to make the problem feasible. Not all the constraints in the original optimization problem (18) can be modified. For instance, the designer will not be able to arbitrarily modify constraints that can directly affect the dynamics of the system, i.e., constraints encoded in f^{dyn}. Solving problem (19) is equivalent to looking for a set of slacks that make the original control problem feasible while minimizing a suitable norm $|| \cdot ||$ of the slack vector. In most of our application examples, we choose the l_1-norm, which tends to provide sparser solutions for $\mathbf{s}$, i.e., nonzero slacks for a smaller number of constraints. However, other norms can also be used, including weighted norms based on the set of weights λ. If problem (19) is feasible, ExtractFeedback uses the solution $\mathbf{s}^*$ to repair the original infeasible specification φ. Otherwise, the infeasible problem is returned for another round of diagnosis to retrieve further constraints to relax. In what follows, we provide details on the implementation of ExtractFeedback.

Algorithm 3 Repair

1: **procedure** Repair
2:     **Input:** $\mathcal{M}$, I, λ, $\mathcal{S}$, φ
3:     **Output:** $repaired$, $\mathcal{M}$, φ
4:     $\mathcal{M}.J \leftarrow \mathcal{M}.J + \lambda^\top s_I$
5:     **for** c in I **do**
6:         **if** $\lambda(c) > 0$ **then**
7:             $\mathcal{M}.C(c) \leftarrow \mathcal{M}.C(c) + s_c$
8:     $(repaired, \mathbf{s}^*) \leftarrow$ Solve$(\mathcal{M}.J, \ \mathcal{M}.C)$
9:     **if** $repaired = 1$ **then**
10:        $\varphi \leftarrow$ ExtractFeedback$(\mathbf{s}^*, \mathcal{S}, \varphi)$

If a minimum norm solution $\mathbf{s}^*$ can be found, then the slack variables $\mathbf{s}^*$ can be mapped to a set of *predicate repairs* $s_\mathcal{D}$, as defined in Problem 1, as follows. The slack vector $\mathbf{s}^*$ in Algorithm 3 includes the set of slack variables $\{s^*_{\mu_i,t}\}$, where $s^*_{\mu_i,t}$ is the variable added to the optimization constraint associated with an atomic predicate $\mu_i \in \mathcal{D}$ at time t, $i \in \{1, \ldots, d\}$. We then set

$$\forall \, i \in \{1, \ldots, d\} \ \ s_i = \mu_i^* - \mu_i = \max_{t \in \{1, \cdots, H\}} s^*_{\mu_i,t}, \qquad (20)$$

H being the time horizon for (18), and $s_\mathcal{D} = \{s_1, \ldots, s_d\}$.

To find a set of *time-interval repairs*, we proceed, instead, as follows:

1. The slack vector $\mathbf{s}^*$ in Algorithm 3 includes the set of slack variables $\{s^*_{\mu_i,t}\}$, where $s^*_{\mu_i,t}$ is the variable added to the optimization constraint associated with an atomic predicate $\mu_i \in \mathcal{D}$ at time t. For each $\mu_i \in \mathcal{D}$, with support interval σ_i, we search for the largest time interval $\sigma_i' \subseteq \sigma_i$ such that the slack variables $s^*_{\mu_i,t}$ for $t \in \sigma_i'$ are 0. If $\mu_i \notin \mathcal{D}$, then we set $\sigma_i' = \sigma_i$.

2. We convert every temporal operator in φ into a combination of $\mathbf{G}$ (timed or untimed) and untimed $\mathbf{U}$ by using the following transformations:

$$\mathbf{F}_{[a,b]}\psi = \neg\mathbf{G}_{[a,b]}\neg\psi,$$

$$\psi_1\mathbf{U}_{[a,b]}\psi_2 = \mathbf{G}_{[0,a]}(\psi_1 \mathbf{U} \ \psi_2) \wedge \mathbf{F}_{[a,b]}\psi_2,$$

where $\mathbf{U}$ is the untimed (unbounded) *until* operator. Let $\hat{\varphi}$ be the new formula obtained from φ after applying these transformations[2].

3. The nodes of the parse tree of $\hat{\varphi}$ can then be partitioned into three subsets, ν, κ, and δ, respectively associated with the atomic *predicates*, *Boolean operators*, and *temporal operators* ($\mathbf{G}, \mathbf{U}$) in $\hat{\varphi}$. We traverse this parse tree from the leaves (atomic predicates) to the root and recursively define for each node i a new support interval σ_i^* as follows:

$$\sigma_i^* = \begin{cases} \sigma_i' & \text{if } i \in \nu \\ \bigcap_{j \in C(i)} \sigma_j^* & \text{if } i \in \kappa \cup \delta_\mathbf{U} \\ \sigma_{C(i)}^* & \text{if } i \in \delta_\mathbf{G} \end{cases} \qquad (21)$$

---

[2]While the second transformation introduces a new interval $[0,a]$, its parameters are directly linked to the ones of the original interval $[a,b]$ (now inherited by the $\mathbf{F}$ operator) and will be accordingly processed by the repair routine.

time	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8
s_{l1}	0	0	0	0	0	-0.26	0	0	0	0
s_{u2}	0	0	0	0	0	0	-0.07	0	0	0

Table 1: Slack variables for horizon, with $\Delta t = 0.2$, and $H = 10$.

where $C(i)$ denotes the children of node i, while $\delta_\mathbf{G}$ and $\delta_\mathbf{U}$ are, respectively, the subsets of nodes associated with the $\mathbf{G}$ and $\mathbf{U}$ operators. We observe that the children set of a $\mathbf{G}$ operator node is a singleton. Therefore, with some abuse of notation, we also use $C(i)$ in (21) to denote a single node in the parse tree.

4. We define the interval repair $\hat{\tau}_j$ for each (timed) temporal operator node j in the parse tree of $\hat{\varphi}$ as $\hat{\tau}_j = \sigma_j^*$. If $\hat{\tau}_j$ is empty for any j, no time-interval repair is possible. Otherwise, we map back the set of intervals $\{\hat{\tau}_j\}$ into a set of interval repairs $\mathcal{T}^*$ for the original formula φ according to the transformations in step 2 and return $\mathcal{T}^*$.

We provide an example of predicate repair below, while time interval repair is exemplified in Section 6.1.

EXAMPLE 4 (COLLISION AVOIDANCE). *We diagnose the specifications introduced in Example 1. To formulate the synthesis problem, we assume a horizon $H = 10$ and a discretization step $\Delta t = 0.2$. The system is found infeasible at the first MPC run, and* Diagnosis *detects the infeasibility of $\varphi_1 \wedge \varphi_2$ at time $t = 6$. Intuitively, given the allowed range of accelerations for* ego*, both the cars end up with entering the forbidden box at some time. Algorithm 1 chooses to repair φ_1 by adding slacks to all of its predicates, such that $\varphi_1' = (-0.5 - s_{l1} \leq y_t^{\text{ego}} \leq 0.5 + s_{u1}) \wedge (-0.5 - s_{l2} \leq x^{\text{adv}} \leq 0.5 + s_{u2})$. Table 1 shows the optimal slack values at each t, while s_{u1} and s_{l2} are set to zero at all t. We can then conclude that the specification replacing φ_1 with φ_1'*

$$\varphi_1' = \mathbf{G}_{[0,\infty)}\neg\big((-0.24 \leq y_t^{\text{ego}} \leq 0.5) \wedge (-0.5 \leq x_t^{\text{adv}} \leq 0.43)\big) \tag{22}$$

is feasible, i.e., the cars will not collide, but the original requirement was overly demanding.

Alternatively, the user can choose to run the repair procedure on φ_2 and change its predicate as $(1.5 - s_l \leq a_t^{\text{ego}} \leq 2.5 + s_u)$. In this case, we decide to stick with the original requirement on collision avoidance, and tune, instead, the control "effort" to satisfy it. Under the assumption of constant acceleration (and bounds), the slacks will be the same at all t. We then obtain $[s_l, s_u] = [0.82, 0]$, which ultimately turns into $\varphi_2' = \mathbf{G}_{[0,\infty)}\big(0.68 \leq a_t^{\text{ego}} \leq 2.5\big)$. The ego vehicle should then slow down to prevent entering the forbidden box at the same time as the other car. This latter solution is, however, suboptimal with respect to the l_1-norm selected in this example.

Our algorithm offers the following guarantees, for which a proof is reported below.

THEOREM 1 (SOUNDNESS). *Given a controller synthesis problem $\mathcal{P} = (f_d, g_d, x_0, \varphi, J)$, such that (8) is infeasible at time t, let $\varphi' \in$ REPAIR$_{\mathcal{D},\mathcal{T}}(\varphi)$ be the repaired formula returned from Algorithm 1 without human intervention, for a given set of predicates $\mathcal{D}$ or time interval $\mathcal{T}$.*

Then, $\mathcal{P}' = (f_d, g_d, x_0, \varphi', J)$ is feasible at time t and $(\varphi', \mathcal{D}, \mathcal{T})$ satisfy the minimality conditions in Problem 1.

PROOF (THEOREM 1). Suppose $\mathcal{M}$ is the MILP encoding of $\mathcal{P}$ as defined in (18), φ' is the repaired formula, and $\mathcal{D}$ the set of diagnosed predicates, as returned by Algorithm 1. We start by discussing the case of predicate repair. We let $\mathcal{M}'$ be the MILP encoding of $\mathcal{P}'$ and $\mathcal{D}^* \subseteq \mathcal{D}$ be the set of predicates that are fixed to provide φ', i.e., such that $s = (\mu^* - \mu) \neq 0$, with $\mu \in \mathcal{D}$. Algorithm 1 modifies $\mathcal{M}$ by introducing a slack variable $s_{\mu,t}$ into each constraint associated with an atomic predicate μ in $\mathcal{D}$ at time t. Such a transformation leads to a feasible MILP $\mathcal{M}''$ and an optimal slack set $\{s^*_{\mu,t} | \mu \in \mathcal{D}, t \in \{1, \ldots, H\}\}$. We now observe that $\mathcal{M}'$ and $\mathcal{M}''$ are both a relaxed version of $\mathcal{M}$. In fact, we can view $\mathcal{M}'$ as a version of $\mathcal{M}$ in which only the constraints associated with the atomic predicates in $\mathcal{D}^*$ are relaxed. Therefore, each constraint having a nonzero slack variable in $\mathcal{M}''$ is also relaxed in $\mathcal{M}'$. Moreover, by (20), the relaxed constraints in $\mathcal{M}'$ are offset by the largest slack value over the horizon H. Then, because $\mathcal{M}''$ is feasible, $\mathcal{M}'$, and subsequently $\mathcal{P}'$, are feasible.

We now prove that $(\varphi', \mathcal{D})$ satisfy the predicate minimality condition of Problem 1. Let $\tilde{\varphi}$ be any formula obtained from φ after repairing a set of predicates $\tilde{\mathcal{D}}$ such that the resulting problem $\tilde{\mathcal{P}}$ is feasible. We recall that, by Definition 4, at least one predicate in $\mathcal{D}$ generates a conflicting constraint and must be repaired for $\mathcal{M}$ to become feasible. Then, $\tilde{\mathcal{D}} \cap \mathcal{D} \neq \emptyset$ holds. Furthermore, since Algorithm 1 iterates by diagnosing and relaxing constraints until feasibility is achieved, $\mathcal{D}$ contains all the predicates that can be responsible for the infeasibility of φ. In other words, Algorithm 1 finds all the IISs in the original optimization problem and allows relaxing any constraint in the union of the IISs. Therefore, repairing any predicate outside of $\mathcal{D}$ is redundant: a predicate repair set that only relaxes the constraints associated with predicates in $\bar{\mathcal{D}} = \tilde{\mathcal{D}} \cap \mathcal{D}$, by the same amount as in $\tilde{\varphi}$, and sets to zero the slack variables associated with predicates in $\mathcal{D} \setminus \bar{\mathcal{D}}$ is also effective and exhibits a smaller slack norm. Let $s_{\bar{\mathcal{D}}}$ be such a repair set and $\bar{\varphi}$ the corresponding repaired formula. $s_{\bar{\mathcal{D}}}$ and $s_{\mathcal{D}}$ can then be seen as two repair sets on the same predicate set. However, by the solution of Problem (19), we are guaranteed that $s_{\mathcal{D}}$ has minimum norm; then, $||s_{\mathcal{D}}|| \leq ||s_{\bar{\mathcal{D}}}||$ will hold for any such formulas $\bar{\varphi}$, and hence $\tilde{\varphi}$.

We now consider the MILP formulation $\mathcal{M}'$ associated with $\mathcal{P}'$ and φ' in the case of time-interval repairs. For each atomic predicate $\mu_i \in \mathcal{D}$, for $i \in \{1, \ldots, |\mathcal{D}|\}$, $\mathcal{M}'$ includes only the associated constraints evaluated over time intervals σ'_i for which the slack variables $\{s_{\mu_i,t}\}$ are zero. Such a subset of constraints is trivially feasible. All the other constraints, enforcing the satisfaction of Boolean and temporal combination of the atomic predicates in φ' are also feasible if the atomic predicate constraints are feasible. Then, $\mathcal{M}'$ is feasible.

To show that $(\varphi', \mathcal{T})$ satisfy the minimality condition in Problem 1, we observe that, by the transformations in step 2 of the time-interval repair procedure, φ is logically equivalent to a formula $\hat{\varphi}$ which only contains *untimed* $\mathbf{U}$ and *timed* $\mathbf{G}$ operators. Moreover, $\hat{\varphi}$ and φ have the same interval pa-

rameters. Therefore, if the proposed repair set is minimal for $\hat{\varphi}$, this will also be the case for φ. We now observe that Algorithm 1 selects, for each atomic predicate $\mu_i \in \mathcal{D}$ the largest interval σ'_i such that the associated constraints are feasible, i.e., their slack variables are zero after norm minimization[3]. Because feasible intervals for Boolean combinations of atomic predicates are obtained by intersecting these maximal intervals, and then propagated to the temporal operators, the length of the intervals of each $\mathbf{G}$ operator in $\hat{\varphi}$, hence of the temporal operators in φ, will also be maximal, which is what we wanted to prove. $\square$

THEOREM 2 (COMPLETENESS). *Assume the controller synthesis problem* $\mathcal{P} = (f_d, g_d, x_0, \varphi, J)$ *results in* (8) *being infeasible at time* t. *If there exist a set of predicates* $\mathcal{D}$ *or time-intervals* $\mathcal{T}$ *such that there exists* $\Phi \subseteq$ REPAIR$_{\mathcal{D},\mathcal{T}}(\varphi)$ *for which* $\forall \phi \in \Phi$, $\mathcal{P}' = (f_d, g_d, x_0, \phi, J)$ *is feasible at time* t *and* $(\phi, \mathcal{D}, \mathcal{T})$ *are minimal in the sense of Problem 1, then Algorithm 1 returns a repaired formula* φ' *in* Φ.

PROOF (THEOREM 2). We first observe that Algorithm 1 always terminates with a feasible solution φ' since the set of MILP constraints to diagnose and repair is finite. We first consider the case of predicate repairs. Let $\mathcal{D}$ be the set of predicates modified to obtain $\phi \in \Phi$ and $\mathcal{D}'$ the set of diagnosed predicates returned by Algorithm 1. Then, by Definition 4 and the iterative approach of Algorithm 1, we are guaranteed that $\mathcal{D}'$ includes all the predicates responsible for inconsistencies, as also argued in the proof of Theorem 1. Therefore, we conclude $\mathcal{D} \subseteq \mathcal{D}'$. $s_{\mathcal{D}}$ and $s_{\mathcal{D}'}$ can then be seen as two repair sets on the same predicate set. However, by the solution of Problem (19), we are guaranteed that $s_{\mathcal{D}'}$ has minimum norm; then, $||s_{\mathcal{D}'}|| \leq ||s_{\bar{\mathcal{D}}}||$ will hold, hence $\varphi' \in \Phi$.

We now consider the case of time-interval repair. If a formula $\phi \in \Phi$ repairs a set of intervals $\mathcal{T} = \{\tau_1, \ldots, \tau_l\}$, then there exists a set of constraints associated with atomic predicates in φ which are consistent in $\mathcal{M}$, the MILP encoding associated with ϕ, and make the overall problem feasible. Then, the relaxed MILP encoding $\mathcal{M}'$ associated with φ after slack norm minimization will also include a set of predicate constraints admitting zero slacks over the same set of time intervals as in $\mathcal{M}$, as determined by $\mathcal{T}$. Since these constraints are enough to make the entire problem $\mathcal{M}$ feasible, this will also be the case for $\mathcal{M}'$. Therefore, our procedure for time-interval repair terminates and produces a set of non-empty intervals $\mathcal{T}' = \{\tau'_1, \ldots, \tau'_l\}$. Finally, because Algorithm 1 finds the longest intervals for which the slack variables associated with each atomic predicate are zero, we are also guaranteed that $||\tau'_i|| \geq ||\tau_i||$ for all $i \in \{1, \ldots, l\}$, as also argued in the proof of Theorem 1. We can then conclude that $\varphi' \in \Phi$ holds. $\square$

In the worst case, Algorithm 1 solves a number of MILP problem instances equal to the number of atomic predicates

---

[3]Because we are not directly maximizing the sparsity of the slack vector, time-interval minimality is to be interpreted with respect to slack norm minimization. Directly maximizing the number of zero slacks is also possible but computationally more intensive.

in the STL formula. While the complexity of solving a MILP is NP-hard, the actual runtime depends on the size of the MILP, which is linear in the number of predicates and operators in the STL specification.

6. CONTRACTS

In this section, we consider specifications provided in the form of a contract $(\varphi_e, \varphi_e \to \varphi_s)$, where φ_e is an STL formula expressing the assumptions, i.e., the set of behaviors assumed from the environment, while φ_s captures the guarantees, i.e., the behaviors promised by the system in the context of the environment. To repair contracts, we can capture tradeoffs between assumptions and guarantees in terms of minimization of a weighted norm of slacks. We describe below our results in both the cases of non-adversarial and adversarial environments.

6.1 Non-Adversarial Environment

For a contract, we make a distinction between controlled inputs u_t and uncontrolled (environment) inputs w_t of the dynamical system. In this section we assume that the environment signal $\mathbf{w}^H$ can be predicted over a finite horizon and set to a known value for which the controller must be synthesized. With $\varphi \equiv \varphi_e \to \varphi_s$, equation (9) reduces to:

$$\begin{aligned} \underset{\mathbf{u}^H}{\text{minimize}} \quad & J(\xi^H(x_0, \mathbf{u}^H, \mathbf{w}^H)) \\ \text{subject to} \quad & \xi^H(x_0, \mathbf{u}^H, \mathbf{w}^H) \models \varphi, \end{aligned} \quad (23)$$

Because of the similarity of Problem (23) and Problem (8), we can then diagnose and repair a contract using the methodology illustrated in Section 5. However, to reflect the different structure of the specification, i.e., its partition into assumption and guarantees, we adopt a weighted sum of the slack variables in Algorithm 1, allocating different weights to predicates in the assumption and guarantee formulas. We can then provide the same guarantees as in Theorems 1 and 2, where $\varphi \equiv \varphi_e \to \varphi_s$ and the minimality conditions are stated with respect to the weighted norm.

EXAMPLE 5 (NON-ADVERSARIAL RACE). *We consider Example 2 with the same discretization step $\Delta t = 0.2$ and horizon $H = 10$ as in Example 1. The MPC scheme results infeasible at time 0. In fact, we observe that ψ_e is always true as $v_0^{\text{adv}} \geq 0.5$ and $a_t^{\text{adv}} = 1 \geq 0$ holds at all times. Since $v_t^{ego} = 0$, the predicate $\psi_{s2} = \mathbf{G}_{[0,\infty)}(v_t^{\text{ego}} \geq 0.5)$ in ψ_s is found to be failing. As in Section 5.2, we can modify the conflicting predicates in the specification by using slack variables as follows: $v_t^{\text{adv}} + s_e(t) \geq 0.5$ (assumptions) and $v_t^{\text{ego}} + s_s(t) \geq 0.5$ (guarantees). However, we also assign a set of weights to the assumption (λ_e) and guarantee (λ_s) predicates, our objective being $\lambda_e|s_e| + \lambda_s|s_s|$. By setting $\lambda_s > \lambda_e$, we encourage modifications in the assumption predicate, thus obtaining $s_e = 0.06$ at time 0 and zero otherwise, and $s_s = 0$ at all times. We can then set $\psi_e' = (v_0^{\text{adv}} \geq 0.56)$, which falsifies ψ_e at time 0, so that $\psi_e \to \psi_s$ is satisfied over the entire range. Alternatively, by setting $\lambda_s < \lambda_e$, we obtain the slack values in Table 2, which lead to the following predicate repair: $\psi_{s2}' = \mathbf{G}_{[0,\infty)}(v_t^{\text{ego}} \geq -0.01)$.*

We can also modify the time interval of the temporal operator associated with ψ_{s2} to repair the overall specification. To

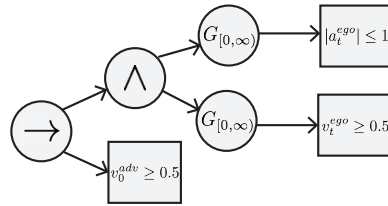| time	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8
s_s	0.51	0.31	0.11	0	0	0	0	0	0	0

Table 2: Slack variables used in Example 2 and 5.

Figure 3: Parse tree of $\psi \equiv \psi_e \to \psi_s$ used in Example 2 and 5.

do so, Algorithm 1 uses the parse tree of $\psi_e \to \psi_s$ in Figure 3. For any of the leaf node predicates μ_i, $i \in \{1, 2, 3\}$, we get a support $\sigma_i = [0, 9]$, which is only limited by the finite horizon H. Then, based on the slack values in Table 2, we can conclude $\sigma_1' = \sigma_2' = [0, 9]$ (the optimal slack values for these predicates are always zero), while $\sigma_3' = [3, 9]$. For the given syntax tree, we also have $\sigma_1^* = \sigma_1'$, $\sigma_2^* = \sigma_2'$, and $\sigma_3^* = \sigma_3'$ for the temporal operator nodes that are parent nodes of μ_1, μ_2, and μ_3, respectively. Since none of the above intervals is empty, a time interval repair is indeed possible by modifying the time interval of the parent node of μ_3, thus achieving $\tau_3^* = \sigma_3^*$. This leads to the following proposed sub-formula $\psi_{s2}' = \mathbf{G}_{[0.6,\infty)}(v_t^{\text{ego}} \geq 0.5)$. In this example, repairing the specification over the first horizon is enough to guarantee controller realizability in the future. We can then keep the upper bound of the $\mathbf{G}$ operator to infinity.

6.2 Adversarial Environment

When the environment can behave adversarially, the control synthesis problem assumes the structure in (9). Specifically, in this paper, we allow w_t to lie in an interval $[w_{\min}, w_{\max}]$ at all times; this corresponds to the STL formula $\varphi_w = \mathbf{G}_{[0,\infty)}(w_{\min} \leq w_t \leq w_{\max})$. We decompose a specification φ of the form $\varphi_w \wedge \varphi_e \to \varphi_s$, representing the contract, as $\varphi \equiv \varphi_w \to \psi$, where $\psi \equiv (\varphi_e \to \varphi_s)$. Our diagnosis and repair method is summarized in Algorithm 4.

We first check the satisfiability of the control synthesis problem by examining whether there exists a pair of $\mathbf{u}^H$ and $\mathbf{w}^H$ for which problem (9) is feasible (CheckSAT routine):

$$\begin{aligned} \underset{\mathbf{u}^H, \mathbf{w}^H}{\text{minimize}} \quad & J(\xi^H(x_0, \mathbf{u}^H, \mathbf{w}^H)) \\ \text{subject to} \quad & \xi^H(x_0, \mathbf{u}^H, \mathbf{w}^H) \models \varphi \\ & \mathbf{w}^H \models \varphi_w \wedge \varphi_e. \end{aligned} \quad (24)$$

If problem (24) is unsatisfiable, we can use the techniques introduced in Section 5.2 and 6.1 to diagnose and repair the infeasibility. Therefore, in the following, we assume that (24) is satisfiable, hence there exist $\mathbf{u}_0^H$ and $\mathbf{w}_0^H$ that solve (24). To check realizability, we use the following CEGIS loop (SolveCEGIS routine). By first fixing the control trajectory to $\mathbf{u}_0^H$, we find the worst case disturbance trajectory $\mathbf{w}_1^H$ that minimizes the robustness value of φ by solving the fol-

Algorithm 4 `DiagnoseRepairAdversarial`

1: **procedure** `DiagnoseRepairAdversarial`
2:    **Input:** $\mathcal{P}$
3:    **Output:** $\mathbf{u}^H$, $\mathcal{P}'$
4:    $(J, C) \leftarrow$ `GenMILP`$(\mathcal{P})$
5:    $(\mathbf{u}_0^H, \mathbf{w}_0^H, sat) \leftarrow$ `CheckSAT`(J, C)
6:    **if** sat **then**
7:       $\mathcal{W}_{cand}^* \leftarrow$ `SolveCEGIS`$(\mathbf{u}_0^H, \mathcal{P})$
8:       $\mathcal{W}_{cand} \leftarrow \mathcal{W}_{cand}^*$
9:       **while** $\mathcal{W}_{cand} \neq \emptyset$ **do**
10:          $\mathcal{P}_w \leftarrow$ `RepairAdversarial`$(\mathcal{W}_{cand}, \mathcal{P})$
11:          $\mathcal{W}_{cand} \leftarrow$ `SolveCEGIS`$(\mathbf{u}_0^H, \mathcal{P}_w)$
12:       $\mathcal{W}_{cand} \leftarrow \mathcal{W}_{cand}^*$, $\mathcal{P}_\psi \leftarrow \mathcal{P}$
13:       **while** $\mathcal{W}_{cand} \neq \emptyset$ **do**
14:          $\mathcal{P}_\psi \leftarrow$ `DiagnoseRepair`$(\mathcal{P}_\psi)$
15:          $\mathcal{W}_{cand} \leftarrow$ `SolveCEGIS`$(\mathbf{u}_0^H, \mathcal{P}_\psi)$
16:       $\mathcal{P}' \leftarrow$ `FindMin`$(\mathcal{P}_w, \mathcal{P}_\psi)$

---

lowing problem:

$$\begin{aligned}
\underset{\mathbf{w}^H}{\text{minimize}} \qquad & \rho^\varphi(\xi^H(x_0, \mathbf{u}^H, \mathbf{w}^H), 0) \\
\text{subject to} \qquad & \mathbf{w}^H \models \varphi_e \wedge \varphi_w
\end{aligned} \tag{25}$$

with $\mathbf{u}^H = \mathbf{u}_0^H$. The optimal $\mathbf{w}_1^H$ from (25) will falsify the specification if the resulting robustness value is below zero[4]. If this is the case, we look for a $\mathbf{u}_1^H$ which solves (23) with the additional restriction of $\mathbf{w}^H \in \mathcal{W}_{cand} = \{\mathbf{w}_1^H\}$. If this step is feasible, we once again attempt to find a worst-case disturbance sequence $\mathbf{w}_2^H$ that solves (25) with $\mathbf{u}^H = \mathbf{u}_1^{H}$: this is the counterexample-guided inductive step. At each iteration i of this CEGIS loop, the set of candidate disturbance sequences $\mathcal{W}_{cand}$ expands to include $\mathbf{w}_i^H$. If the loop terminates at iteration i with a successful $\mathbf{u}_i^H$ (one for which the worst case disturbance $\mathbf{w}_i^H$ in (25) has positive robustness), we conclude that the formula φ is realizable.

The CEGIS loop may not terminate if the set $\mathcal{W}_{cand}$ is infinite. We, therefore, run it for a maximum number of iterations. If `SolveCEGIS` fails to find a controller sequence prior to the timeout, then (23) is infeasible for the current $\mathcal{W}_{cand}$, i.e., there is no control input that can satisfy φ for all disturbances in $\mathcal{W}_{cand}$. We conclude that the specification is not realizable (or, equivalently, the contract is inconsistent). While this infeasibility can be repaired by modifying ψ based on the techniques in Section 5.2 and 6.1, an alternative solution is to repair φ_w by minimally pruning the bounds on w_t (`RepairAdversarial` routine).

To do so, given a small tolerance $\epsilon \in \mathbb{R}^+$, we find

$$w_u = \max_{\substack{\mathbf{w}_i^H \in \mathcal{W}_{cand} \\ t \in \{1, \ldots, H-1\}}} w_{i,t} \qquad w_l = \min_{\substack{\mathbf{w}_i^H \in \mathcal{W}_{cand} \\ t \in \{1, \ldots, H-1\}}} w_{i,t} \tag{26}$$

and define $s_u = w_{\max} - w_u$ and $s_l = w_l - w_{\min}$. We then use s_u and s_l to update the range for w_t in φ_w to a maximal interval $[w'_{\min}, w'_{\max}] \subseteq [w_{\min}, w_{\max}]$ and such that at least one $\mathbf{w}_i^H \in \mathcal{W}_{cand}$ is excluded. Specifically, if $s_u \leq s_l$, we set $[w'_{\min}, w'_{\max}] = [w_{\min}, w_u - \epsilon]$; otherwise we set $[w'_{\min}, w'_{\max}] = [w_l + \epsilon, w_{\max}]$. The smaller the value

---

[4]A tolerance ρ_{min} can be selected to accommodate approximation errors, i.e., $\rho^\varphi(\xi^H(x_0, \mathbf{u}_0^H, \mathbf{w}_1^H), 0) < \rho_{min}$.

---

of ϵ, the larger the resulting interval. Finally, we use the updated formula φ'_w to run `SolveCEGIS` again until a realizable control sequence $\mathbf{u}^H$ is found. In Algorithm 4, assuming a predicate repair procedure, `FindMin` provides the solution with minimum slack norm between the ones repairing ψ and φ_w.

EXAMPLE 6 (ADVERSARIAL RACE). *We consider the specification in Example 3. For the same horizon as in the previous examples, after solving the satisfiability problem, for the fixed $\mathbf{u}_0^H$, the CEGIS loop returns $a_t^{\mathrm{adv}} = 2$ for all $t \in \{0, \ldots, H-1\}$ as the single element in $\mathcal{W}_{cand}$ for which no controller sequence can be found. We then choose to tighten the environment assumptions to make the controller realizable, by shrinking the bounds on a_t^{adv} by using Algorithm 4 with $\epsilon = 0.01$. After a few iterations, we finally obtain $w'_{\min} = 0$ and $w'_{\max} = 1.24$, and therefore $\phi'_w = \mathbf{G}_{[0,\infty)}\big(0 \leq a_t^{\mathrm{adv}} \leq 1.22\big)$.*

Under the assumption that `SolveCEGIS` terminates before reaching the maximum number of iterations[5], and within the selected tolerance ϵ, the following theorems state the properties of Algorithm 4.

THEOREM 3 (SOUNDNESS). *Given a controller synthesis problem $\mathcal{P} = (f_d, g_d, x_0, \varphi, J)$, such that (9) is infeasible at time t, let $\varphi' \in$ REPAIR$_{\mathcal{D}, \mathcal{T}}(\varphi)$ be the repaired formula returned from Algorithm 4 without human intervention, for a given set of predicates $\mathcal{D}$ or time interval $\mathcal{T}$. Then, $\mathcal{P}' = (f_d, g_d, x_0, \varphi', J)$ is feasible at time t and $(\varphi', \mathcal{D}, \mathcal{T})$ satisfy the minimality conditions in Problem 2.*

PROOF (THEOREM 3). We recall that $\varphi \equiv \varphi_w \to \psi$. Moreover, Algorithm 4 provides the solution with minimum slack norm between the ones repairing ψ and ϕ_w in the case of predicate repair. Then, when $\psi = \varphi_e \to \varphi_s$ is modified using Algorithm 1, soundness is guaranteed by Theorem 1 and the termination of the CEGIS loop. On the other hand, assume Algorithm 4 modifies the atomic predicates in ϕ_w. Then, the `RepairArdversarial` routine and (26), together with the termination of the CEGIS loop, assure that φ_w is also repaired in such a way that the controller is realizable, and the length of the bounding box around w_t is maximal within an error bounded by ϵ (i.e., it differs from the maximal interval length by at most ϵ), which concludes our proof. $\square$

THEOREM 4 (COMPLETENESS). *Assume the controller synthesis problem $\mathcal{P} = (f_d, g_d, x_0, \varphi, J)$ results in (9) being infeasible at time t. If there exist a set of predicates $\mathcal{D}$ and time-intervals $\mathcal{T}$ such that there exists $\Phi \subseteq$ REPAIR$_{\mathcal{D}, \mathcal{T}}(\varphi)$ for which $\forall \phi \in \Phi$, $\mathcal{P}' = (f_d, g_d, x_0, \phi, J)$ is feasible at time t and $(\phi, \mathcal{D}, \mathcal{T})$ are minimal in the sense of Problem 2, then Algorithm 4 returns a repaired formula φ' in Φ.*

PROOF (THEOREM 4). As discussed in the proof of Theorem 3, if Algorithm 4 modifies $\psi = \varphi_e \to \varphi_s$ using Algorithm 1, completeness is guaranteed by Theorem 2 and

---

[5]If this is not the case, then Algorithm 4 terminates with `UNKNOWN`.

the termination of the CEGIS loop. On the other hand, let us assume there exists a minimum norm repair for the atomic predicates of φ_w, which returns a maximal interval $[w'_{\min}, w'_{\max}] \subseteq [w_{\min}, w_{\max}]$. Then, given the termination of the CEGIS loop, by repeatedly applying (26) and `RepairAdversarial`, it is also possible to produce a predicate repair such that the corresponding interval $[w''_{\min}, w''_{\max}]$ makes the control synthesis realizable and is maximal within an error bounded by ϵ (i.e., its length differs by at most ϵ from the one of the maximal interval $[w'_{\min}, w'_{\max}]$). Hence, $\varphi' \in \Phi$ holds. $\square$

7. CASE STUDIES

7.1 Autonomous Driving

We consider the problem of synthesizing a controller for an autonomous vehicle in a city driving scenario. We analyze the following two tasks: (i) changing lanes on a busy road; (ii) performing an unprotected left turn at a signalized intersection. We use a simple point-mass model for the vehicles on the road. For each vehicle, we define the state as $\mathbf{x} = [x\ y\ \theta\ v]^\top$, where x and y denote the coordinates, and θ and v represent the direction and speed, respectively. Let $\mathbf{u} = [u_1\ u_2]^\top$ be the control input for each vehicle, where u_1 is the steering input and u_2 is the acceleration. Then, the vehicle's state evolves according to the following dynamics:

$$
\begin{aligned}
\dot{x} &= v \cos\theta \\
\dot{y} &= v \sin\theta \\
\dot{\theta} &= v \cdot u_1 / m \\
\dot{v} &= u_2,
\end{aligned}
\tag{27}
$$

where m is the vehicle mass. To determine the control strategy, we linearize the overall system dynamics around the initial state at each run of the MPC, which is completed in less than 2 s on a 2.3-GHz Intel Core i7 processor with 16-GB memory. We further impose the following constraints on the *ego* vehicle (i.e., the vehicle under control): (i) a minimum distance must be established between the *ego* vehicle and other cars on the road to avoid collisions; (ii) the *ego* vehicle must obey the traffic lights; (iii) the *ego* vehicle must stay within its road boundaries.

7.1.1 Lane Change

We consider a lane change scenario on a busy road as shown in Fig. 4a. The *ego* vehicle is in red. *Car 1* is at the back of the left lane, *Car 2* is in the front of the left lane, while *Car 3* is on the right lane. The states of the vehicles are initialized as follows: $x_0^{\text{Car 1}} = [-0.2\ -1.5\ \frac{\pi}{2}\ 0.5]^\top$, $x_0^{\text{Car 2}} = [-0.2\ 1.5\ \frac{\pi}{2}\ 0.5]^\top$, $x_0^{\text{Car 3}} = [0.2\ 1.5\ \frac{\pi}{2}\ 0]^\top$, and $x_0^{\text{ego}} = [0.2\ -0.7\ \frac{\pi}{2}\ 0]^\top$. The control inputs for *ego* and *Car 3* are initialized at $[0\ 0]^\top$; the ones for *Car 1* and *Car 2* are set to $u_0^{\text{Car 1}} = [0\ 1]^\top$ and $u_0^{\text{Car 2}} = [0\ -0.25]^\top$. The objective of *ego* is to safely change lane, while satisfying the following requirements:

$$
\begin{aligned}
\varphi_{\text{str}} &= \mathbf{G}_{[0,\infty)}(|u_1| \leq 2) & \text{Steering Bounds} \\
\varphi_{\text{acc}} &= \mathbf{G}_{[0,\infty)}(|u_2| \leq 1) & \text{Acceleration Bounds} \\
\varphi_{\text{vel}} &= \mathbf{G}_{[0,\infty)}(|v| \leq 1) & \text{Velocity Bounds}
\end{aligned}
\tag{28}
$$

The solid blue line in Fig. 4 is the trajectory of *ego* as obtained from our MPC scheme, while the dotted green line is the future trajectory pre-computed for a given horizon at a given time. MPC becomes infeasible at time $t = 1.2$ s when
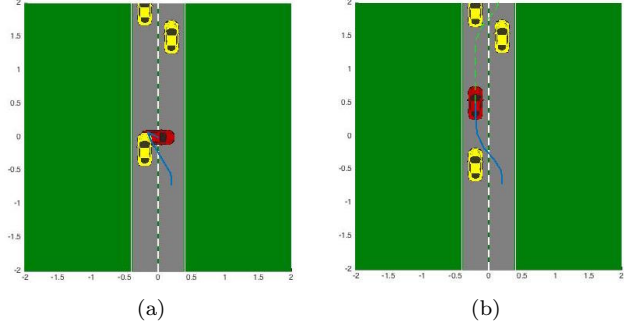

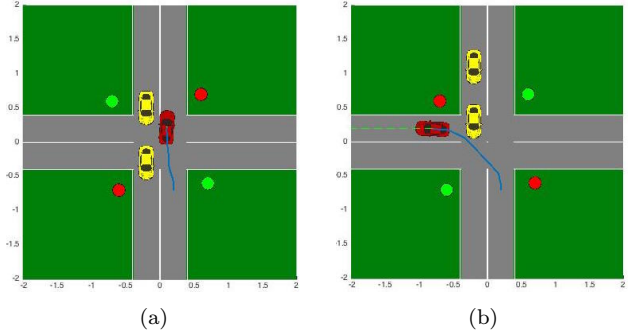Figure 4: Changing lane is infeasible at $t = 1.2$ s in (a) and gets repaired in (b).

Figure 5: Left turn becomes infeasible at time $t = 2.1$ s in (a) and is repaired in (b).

the no-collision requirement is violated, and a possible collision is detected between the *ego* vehicle and *Car 1* before the lane change is completed (Fig. 4a). Our solver takes 2 s, out of which 1.4 s are needed to generate all the IISs, consisting of 39 constraints. To make the system feasible, the proposed repair increases both the acceleration bounds and the velocity bounds on the *ego* vehicle as follows:

$$
\begin{aligned}
\varphi_{\text{acc}}^{\text{new}} &= \mathbf{G}_{[0,\infty)}(|u_2| \leq 3.5) \\
\varphi_{\text{vel}}^{\text{new}} &= \mathbf{G}_{[0,\infty)}(|v| \leq 1.54)
\end{aligned}
\tag{29}
$$

When replacing the initial requirements φ_{acc} and φ_{vel} with the modified ones, the revised MPC scheme allows the vehicle to travel faster and safely complete a lane change maneuver, without risks of collision, as shown in Fig. 4b.

7.1.2 Unprotected Left Turn

In the second scenario, we would like the *ego* vehicle to perform an unprotected left turn at a signalized intersection, where the *ego* vehicle has a green light and is supposed to yield to oncoming traffic, represented by the yellow cars crossing the intersection in Fig. 5. The environment vehicles are initialized at the states $x_0^{\text{Car 1}} = [-0.2\ 0.7\ -\frac{\pi}{2}\ 0.5]^\top$ and $x_0^{\text{Car 2}} = [-0.2\ 1.5\ -\frac{\pi}{2}\ 0.5]^\top$, while the *ego* vehicle is initialized at $x_0^{\text{ego}} = [0.2\ -0.7\ \frac{\pi}{2}\ 0]^\top$. The control input for each vehicle is initialized at $[0\ 0]^\top$. Moreover, we use the same bounds as in (28).

The MPC scheme becomes infeasible at $t = 2.1$ s. The solver takes 5 s, out of which 2.2 s are used to generate the IISs, including 56 constraints. As shown in Fig. 5a, the *ego* vehicle yields in the middle of intersection for the oncoming traffic to pass. However, the traffic signal turns red in the
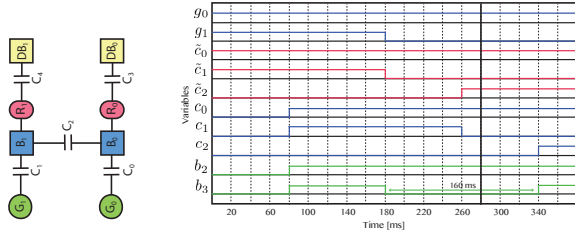
Figure 6: Simplified model of an aircraft electric power system (left) and counterexample trajectory (right). The blue, green and red lines represent environment, state, and controller variables, respectively, for a 380-ms run.

meanwhile, and there is no feasible control input for the *ego* vehicle without breaking the traffic light rules. Since we do not allow modifications to the traffic light rules, the original specification is repaired again by increasing the bounds on acceleration and velocity, thus obtaining:

$$\begin{aligned}
\varphi_{\text{acc}}^{\text{new}} &= \quad \mathbf{G}_{[0,\infty)}\big(|u_2| \leq 11.903\big) \\
\varphi_{\text{vel}}^{\text{new}} &= \qquad \mathbf{G}_{[0,\infty)}\big(|v| \leq 2.42\big)
\end{aligned} \tag{30}$$

As shown by the trajectory in Fig. 5b, under the assumptions and initial conditions of our scenario, higher allowed velocity and acceleration make the *ego* vehicle turn before the oncoming cars get close or cross the intersection.

7.2 Aircraft Electric Power System

Fig. 6 shows a simplified architecture for the primary power distribution system in a passenger aircraft [18]. Two power sources, the left and right generators G_0 and G_1, deliver power to a set of high-voltage AC and DC buses (B_0, B_1, DB_0, and DB_1) and their loads. AC power from the generators is converted to DC power by rectifier units (R_1 and R_2). A bus power control unit (controller) monitors the availability of power sources and configures a set of electromechanical switches, denoted as contactors ($C_0, \ldots, C_4$), such that essential buses remain powered even in the presence of failures, while satisfying a set of safety, reliability, and real-time performance requirements [18]. Specifically, we assume that only the right DC bus DB_1 is essential, and use our algorithms to check the feasibility of a controller that accommodates a failure in the right generator G_1, by rerouting power from the left generator to the right DC bus in a time interval which is less than or equal to $t_{\max} = 100$ ms. In addition, the controller must satisfy the following set of requirements, all captured by an STL contract.

Assumptions. *When a contactor receives an open (close) signal, it shall become open (closed) in 80 ms or less.* Let the time discretization step $\Delta t = 20$ ms, $\tilde{c}_i$, $i \in \{0, \ldots, 4\}$ be a set of Boolean variables describing the controller signal (where 1 stands for "closed" and 0 for "open"), c_i, $i \in \{0, \ldots, 4\}$ be a set of Boolean variables denoting the state (open/closed) of the contactors. We can capture the system assumptions via a conjunction of formulas of the form: $\mathbf{G}_{[0,\infty)}(\tilde{c}_i \rightarrow \mathbf{F}_{[0,4]}c_i)$, providing a model for the discrete-time binary-valued contactor states. The actual delay of each contactor can then be modeled using an integer (environment) variable k_i for which we require: $\mathbf{G}_{[0,\infty)}(0 \leq k_i \leq 4)$.

Guarantees. *If a generator becomes unavailable (fails), the*

controller shall disconnect it from the power network in 20 ms or less. Let g_0 and g_1 be Boolean environment variables representing the state of the generators, where 1 stands for "available" and 0 for "failure." We encode the above guarantees as $\mathbf{G}_{[0,\infty)}(g_i \rightarrow \mathbf{F}_{[0,1]}\tilde{c}_i)$. *A DC bus shall never be disconnected from an AC generator for 100 ms or more,* i.e., $\mathbf{G}_{[0,\infty)}(\neg b_i \rightarrow \mathbf{F}_{[0,5]}b_i)$, where b_i, $i \in \{0, \ldots, 3\}$ is a set of Boolean variables denoting the status of a bus, where 1 stands for "powered" and 0 for "unpowered." Additional guarantees, which can also be expressed as STL formulas, include: (i) If both AC generators are available, the left AC generator shall power the left AC bus, and the right AC generator shall power the right AC bus. C_3 and C_4 shall be closed. (ii) If one generator becomes unavailable, all buses shall be connected to the other generator. (iii) Two generators must never be directly connected.

We apply the diagnosis and repair procedure in Section 6.2 to investigate whether there exists a control strategy that can satisfy the specification above over all possible values of contactor delays. As shown in Fig. 6, the controller is unrealizable; a trace of contactor delays equal to 4 at all times provides a counterexample, which leaves DB_1 unpowered for 160 ms, exceeding the maximum allowed delay of 100-ms. In fact, the controller cannot close C_2 until C_1 is tested as being open, to ensure that G_1 is safely isolated from G_2. To guarantee realizability, Algorithm 4 suggests to either modify our assumptions to $\mathbf{G}_{[0,\infty)}(0 \leq k_i \leq 2)$ for $i \in \{0, \ldots, 4\}$ or relax the guarantee on DB_1 to $G_{[0,\infty)}(\neg b_3 \rightarrow \mathbf{F}_{[0,8]}b_3)$. The overall execution time was 326 s, which includes formulating and executing three CEGIS loops, requiring a total of 6 optimization problems.

8. CONCLUSION

We presented a set of algorithms for diagnosis and repair of STL specifications in the setting of controller synthesis for hybrid systems using a model predictive control scheme. Given an unrealizable specification, our algorithms can detect possible reasons for infeasibility and suggest repairs to make it realizable. We showed the effectiveness of our approach on the synthesis of controllers for several applications. As future work, we plan to investigate techniques that better leverage the structure of the STL formulas and extend to a broader range of environment assumptions in the adversarial setting.

9. ACKNOWLEDGMENTS

10. REFERENCES

[1] Gurobi Optimizer. [Online]: http://www.gurobi.com/.

[2] R. Alur, S. Moarref, and U. Topcu. Counter-strategy guided refinement of GR(1) temporal logic specifications. In *Formal Methods in Computer-Aided Design*, 2013.

[3] A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints.

Automatica, 35, 1999.

[4] A. Bemporad and M. Morari. Robust model predictive control: A survey. In *Robustness in identification and control*, pages 207–226. Springer, 1999.

[5] J. W. Chinneck and E. W. Dravnieks. Locating minimal infeasible constraint sets in linear programs. *ORSA Journal on Computing*, 3(2):157–168, 1991.

[6] A. Donzé, T. Ferrère, and O. Maler. Efficient robust monitoring for STL. In *Computer Aided Verification*, 2013.

[7] A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *FORMATS*, 2010.

[8] A. Donzé, O. Maler, E. Bartocci, D. Nickovic, R. Grosu, and S. Smolka. On temporal logic and signal processing. In *Automated Technology for Verification and Analysis*. 2012.

[9] T. Ferrère, O. Maler, and D. Nickovic. Trace diagnostics using temporal implicants. In *Proc. Int. Symp. Automated Technology for Verification and Analysis*, 2015.

[10] C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: theory and practice–a survey. *Automatica*, 25, 1989.

[11] E. C. Kerrigan and J. M. Maciejowski. Soft constraints and exact penalty functions in model predictive control. In *Control 2000 Conference, Cambridge*, 2000.

[12] W. Li, L. Dworkin, and S. A. Seshia. Mining assumptions for synthesis. In *ACM/IEEE Int. Conf. Formal Methods and Models for Codesign*, 2011.

[13] W. Li, D. Sadigh, S. S. Sastry, and S. A. Seshia. Synthesis for human-in-the-loop control systems. In *TACAS*. 2014.

[14] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. 2004.

[15] M. Morari, C. Garcia, J. Lee, and D. Prett. *Model predictive control*. Prentice Hall Englewood Cliffs, NJ, 1993.

[16] P. Nuzzo, A. Puggelli, S. A. Seshia, and A. L. Sangiovanni-Vincentelli. CalCS: SMT solving for non-linear convex constraints. In *IEEE Int. Conf. Formal Methods in Computer-Aided Design*, 2010.

[17] P. Nuzzo, A. Sangiovanni-Vincentelli, D. Bresolin, L. Geretti, and T. Villa. A platform-based design methodology with contracts and related tools for the design of cyber-physical systems. *Proc. IEEE*, 103(11), Nov. 2015.

[18] P. Nuzzo, H. Xu, N. Ozay, J. Finn, A. Sangiovanni-Vincentelli, R. Murray, A. Donzé, and S. Seshia. A contract-based methodology for aircraft electric power system design. *IEEE Access*, 2:1–25, 2014.

[19] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia. Reactive synthesis from signal temporal logic specifications. In *Proc. Int. Conf. Hybrid Systems: Computation and Control*, 2015.

[20] V. Raman and H. Kress-Gazit. Explaining impossible high-level robot behaviors. *IEEE Trans. Robotics*, 29, 2013.

[21] V. Raman, M. Maasoumy, A. Donzé, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia. Model predictive control with signal temporal logic specifications. In *IEEE Conf. on Decision and Control*, 2014.

[22] V. Schuppan. Towards a notion of unsatisfiable cores for LTL. In *Fundamentals of Software Engineering*, 2009.

[23] P. O. Scokaert and J. B. Rawlings. Feasibility issues in linear model predictive control. *AIChE Journal*, 45(8):1649–1659, 1999.