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 automatic synthesis of controllers from specifications using model predictive control. We build on recent approaches that reduce the controller synthesis problem to solving one or more mixed integer linear programs (MILPs), where infeasibility of an 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 relative to the synthesis algorithm, i. e., they provide a diagnosis that makes the synthesis problem infeasible, 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 controller synthesis 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 to the adoption of these methods in practice is the difficulty of writing 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. However, in many cases, a system specification can be more conveniently provided as a *contract* separating the responsibilities of the system under control (i. e., the guarantees) from the assumptions on the external, possibly adversarial environment [20, 19]. In such a scenario, besides *"weakening" the guarantees*, realizability of a controller can be achieved by *"strengthening" the assumptions*. Indeed, a specification could be unrealizable because the environment assumptions are too weak, 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 [24]. In the context of synthesis from LTL, Raman et al. [22] 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 was first proposed by Li et al. [13] and further explored by others [2, 14]. Our approach, based on exploiting information from optimization solvers, has similarities to these techniques as well as to the work of Nuzzo et al. [18] on extracting 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)* [16], a specification language that is well-suited for hybrid systems. Our work is conducted in the setting of automated synthesis from STL via optimization in a model predictive control (MPC) framework [23, 21]. 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 [23]. In the reactive setting, this approach proceeds by iteratively solving a combination of optimization problems using a *counterexample-guided inductive syn-*

HSCC'16, April 12 - 14, 2016, Vienna, Austria

ISBN 978-1-4503-3955-1/16/04…$15.00

DOI: http://dx.doi.org/10.1145/2883817.2883847

thesis (CEGIS) scheme [21]. In this context, an unrealizable STL specification leads to an infeasible optimization problem. The problem of infeasibility in constrained predictive control schemes has been widely addressed in the literature, e.g., by adopting robust MPC, soft constraints, and penalty functions [12, 25, 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.

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. 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 relative to the synthesis algorithm, i. e., they provide a diagnosis that makes the synthesis problem infeasible, and always terminate with a non-trivial specification that is feasible using the chosen synthesis method, when such a repair exists in the space of possible repairs. Our use of MILP enables us to 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 autonomous driving and aircraft electric power system applications.

The paper is organized as follows. We begin in Sec. 2 and 3 with preliminaries and a running example. We formalize the diagnosis and repair problems in Sec. 4 and describe our algorithms for both monolithic and contract specifications in Sec. 5 and 6. Case studies are presented in Sec. 7.

2. PRELIMINARIES

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

2.1 Hybrid Dynamical Systems

We consider continuous-time hybrid dynamical systems:

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

where $x_t \in \mathcal{X} \subseteq (\mathbb{R}^{n_c} \times \{0,1\}^{n_l})$ represents the hybrid (continuous and logical) state at time t, $u_t \in \mathcal{U} \subseteq (\mathbb{R}^{m_c} \times \{0,1\}^{m_l})$ is the hybrid control input, $y_t \in \mathcal{Y} \subseteq (\mathbb{R}^{p_c} \times \{0,1\}^{p_l})$ is the output, and $w_t \in \mathcal{W} \subseteq (\mathbb{R}^{e_c} \times \{0,1\}^{e_l})$ is the hybrid external input, 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:

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

where state and output evolve over time steps $k \in \mathbb{N}$, where $x_k = x(\lfloor t/\Delta t \rfloor) \in \mathcal{X}$. Given the initial state of the system $x_0 \in \mathcal{X}$, a *run* of the system is expressed as:

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

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 define $\xi_k = (x_k, y_k, u_k, w_k)$.

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

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

where $x_1, \dots, x_{H-1}$, $y_0, \dots, y_{H-1}$ 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) 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 formula [7, 6].

An STL formula φ is evaluated on a signal ξ at time t: $(\xi, t) \models \varphi$ denotes that φ evaluates to true on 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 a function of signal ξ at 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:

$$\varphi ::= \mu \mid \neg\mu \mid \varphi \wedge \psi \mid \mathbf{G}_{[a,b]}\psi \mid \mathbf{F}_{[a,b]}\psi \mid \varphi\, \mathbf{U}_{[a,b]}\psi, \tag{4}$$

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. For example, $\xi \models \mathbf{G}_{[a,b]}\psi$ specifies that ψ must hold for signal ξ at all times of the given interval, i. e., $\forall t \in [a, b], (\xi, t) \models \psi$. Formally, the satisfaction of a formula φ for a signal ξ at time t is defined as:

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

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 [6].

2.3 Model Predictive Control

Model Predictive Control (MPC), or *Receding Horizon Control* (RHC), is a well studied control method for hybrid dynamical systems [17, 10]. In RHC, at any time step, the state of the system is observed and an optimal control problem is solved over a finite time horizon H, for a given set of constraints and a cost function J. When f, as defined in (2), is nonlinear, we assume this 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 $[k, k + H - 1]$. Then, only the first component of $\mathbf{u}_*^H$ is applied, and a similar optimization is solved at $k+1$ to compute a new optimal control sequence along the interval $[k + 1, k + H]$ for the model linearized around time step $k + 1$. While the global optimality of MPC is not guaranteed, the technique is widely used and performs well in practice.

In this paper, we use STL to express temporal constraints on the environment and system runs during MPC. We then translate an STL specification into a set of mixed integer

linear constraints [23, 21]. Given a formula φ to be satisfied over a finite horizon H, the associated optimization is:

$$\min_{\mathbf{u}^H} J(\xi^H(x_0, \mathbf{u}^H)) \quad \text{s. t.} \quad \xi^H(x_0, \mathbf{u}^H) \models \varphi, \tag{5}$$

which yields 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 (5) [23, 21].

While (5) applies to systems without uncontrolled inputs, a more general formulation can be provided to account for an uncontrolled disturbance input $\mathbf{w}^H$ that acts, in general, adversarially [21]. To provide this formulation, we assume the specification is given in the form of an STL *assume-guarantee (A/G) contract* [20, 19] $\mathcal{C} = (V, \varphi_e, \varphi \equiv \varphi_e \rightarrow \varphi_s)$, 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 is then represented as a *minimax* optimization problem:

$$\begin{array}{lll} \underset{\mathbf{u}^H}{\text{minimize}} & \underset{\mathbf{w}^H \in \mathcal{W}^e}{\text{maximize}} & J(\xi^H(x_0, \mathbf{u}^H, \mathbf{w}^H)) \\ \text{subject to} & \forall \mathbf{w}^H \in \mathcal{W}^e & \xi^H(x_0, \mathbf{u}^H, \mathbf{w}^H) \models \varphi, \end{array} \tag{6}$$

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 (6) 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.

Following [23, 21], we solve the optimization problems in (5) and (6) by translating the STL formula φ into a set of mixed integer constraints, thus reducing the problem to a *Mixed Integer Program* (MIP). In this paper, we consider control problems that are 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. Recall that $(\xi, t) \models \varphi \Leftrightarrow \rho^{\varphi}(\xi, t) > 0$. The robustness value of a formula with temporal or Boolean operators is expressed recursively as the *min* or *max* of the robustness values of the operands over time. These operations can in turn be encoded as mixed integer constraints. For instance, to encode $min(\rho^{\varphi_1}, ..., \rho^{\varphi_n})$, we introduce Boolean variables z^{φ_i} for $i \in \{1, \ldots, n\}$ and a continuous variable p. The resulting MILP constraints are:

$$\begin{aligned} & p \leq \rho^{\varphi_i}, \sum_{i=1 \ldots n} z^{\varphi_i} \geq 1 \\ & \rho^{\varphi_i} - (1 - z^{\varphi_i})M \leq p \leq \rho^{\varphi_i} + (1 - z^{\varphi_i})M, \end{aligned} \tag{7}$$

where M is a constant selected to be much larger than $|\rho^{\varphi_i}|$ 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 (7).

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.

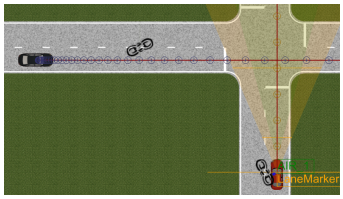


Figure 1: Vehicles crossing an intersection. The red car is the *ego* vehicle, while the black car is part of the environment.

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.,

$$\begin{aligned} & \tilde{x}_t = (x_t^{\text{ego}}, y_t^{\text{ego}}, v_t^{\text{ego}}, x_t^{\text{adv}}, y_t^{\text{adv}}, v_t^{\text{adv}}) \\ & u_t = a_t^{\text{ego}} \quad w_t = a_t^{\text{adv}}. \end{aligned} \tag{8}$$

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

$$\begin{bmatrix} \dot{x}^{\text{ego}} \\ \dot{y}^{\text{ego}} \\ \dot{v}^{\text{ego}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x^{\text{ego}} \\ y^{\text{ego}} \\ v^{\text{ego}} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u. \tag{9}$$

A similar equation holds for the environment vehicle which is, however, constrained to move along the horizontal axis rather than the vertical axis. We assume the *ego* vehicle is initialized at the coordinates $(0, -1)$ and the other vehicle is initialized at $(-1, 0)$. All 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.,* $a_t^{\text{adv}} = 2$*, while the initial velocities are* $v_0^{\text{adv}} = 0$ *and* $v_0^{\text{ego}} = 0$*. We encode our requirements using the formula* $\varphi := \varphi_1 \wedge \varphi_2$*, where* φ_1 *and* φ_2 *are defined as follows:*

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

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). *In the race scenario, assuming the adversary's velocity always exceeds* 0.5*, the* ego *vehicle must maintain a velocity above* 0.5*. We formalize our requirement as a contract* $(\psi_e, \psi_e \rightarrow \psi_s)$*, 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:*

$$\begin{aligned} & \psi_e = \mathbf{G}_{[0,\infty)} (v_t^{\text{adv}} \geq 0.5), \\ & \psi_s = \mathbf{G}_{[0,\infty)} (-1 \leq a_t^{\text{ego}} \leq 1) \wedge (v_t^{\text{ego}} \geq 0.5). \end{aligned} \tag{10}$$

The initial velocities are $v_0^{\text{adv}} = 0.55$ *and* $v_0^{\text{ego}} = 0$*, while the environment vehicle's acceleration is* $a_t^{\text{adv}} = 1$ *at all times. We 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 varies up to a value of* 2*. Initially,* $v_0^{\text{adv}} = 0$ *and* $v_0^{\text{ego}} = 0$ *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* $(\phi_w,\ \phi_w \rightarrow \phi_s)$*, where:*

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

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. The *controller synthesis* problem, denoted $\mathcal{P} = (f_d, g_d, x_0, \varphi, J)$, is to solve (5) (when φ is a monolithic specification of the desired system behaviors) or (6) (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 that is 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 atomic predicates and the time intervals of its temporal operators. We start by providing a definition of the *support* of a formula's atomic predicates, i.e., the set of times at which the value of a predicate affects satisfiability of the formula. We build on this definition to formalize 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* $\gamma : \Phi \rightarrow \Phi$ *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* $\Delta \in \{\mathbf{G}, \mathbf{F}, \mathbf{U}\}$*.*

We can compose repair actions 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 $\texttt{REPAIR}(\varphi)$ the set of all possible formulae obtained through allowed sequences of repairs on φ. Further, given a set of atomic predicates $\mathcal{D}$ and a set of time intervals $\mathcal{T}$, let $\texttt{REPAIR}_{\mathcal{T},\mathcal{D}}(\varphi) \subseteq \texttt{REPAIR}(\varphi)$ 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 formulate the problems addressed in this paper, namely that of diagnosis and repair of a *monolithic* specification φ (*general diagnosis and repair*), and of an A/G contract $(\varphi_e, \varphi_e \rightarrow \varphi_s)$ (*contract diagnosis and repair*).

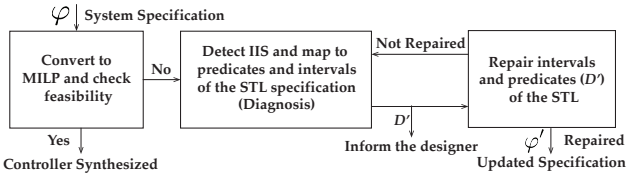


Figure 2: Diagnosis and repair flow diagram.

PROBLEM 1 (GENERAL DIAGNOSIS AND REPAIR). *Given a controller synthesis problem* $\mathcal{P} = (f_d, g_d, x_0, \varphi, J)$ *such that* (5) *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*,*
- $\varphi' \in \texttt{REPAIR}_{\mathcal{T},\mathcal{D}}(\varphi)$*,*

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*

$$\nexists\,(\mathcal{D}', s_{\mathcal{D}'}) \quad \text{s.t.} \quad ||s_{\mathcal{D}'}|| \leq ||s_{\mathcal{D}}|| \wedge \exists\,\varphi'' \in \texttt{REPAIR}_{\mathcal{T},\mathcal{D}'}(\varphi)$$
$$\text{s.t. } \mathcal{P}'' = (f_d, g_d, x_0, \varphi'', J) \text{ is feasible.} \tag{11}$$

- (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*:*

$$\nexists\,\mathcal{T}' = \{\tau_1', \dots, \tau_l'\}, \text{ s.t.}$$
$$\exists i \in \{1, \dots, l\}, ||\tau_i^*|| \leq ||\tau_i'|| \wedge \exists\,\varphi'' \in \texttt{REPAIR}_{\mathcal{T}',\mathcal{D}}(\varphi)$$
$$\text{s.t. } \mathcal{P}'' = (f_d, g_d, x_0, \varphi'', J) \text{ is feasible.} \tag{12}$$

PROBLEM 2 (CONTRACT DIAGNOSIS AND REPAIR). *Given a controller synthesis problem* $\mathcal{P} = (f_d, g_d, x_0, \varphi \equiv \varphi_e \rightarrow \varphi_s, J)$ *such that* (6) *is infeasible, find:*

- *Sets of atomic predicates* $\mathcal{D}_e = \{\mu_1^e, \dots, \mu_d^e\}$*,* $\mathcal{D}_s = \{\mu_1^s, \dots, \mu_d^s\}$ *or sets of time intervals* $\mathcal{T}_e = \{\tau_1^e, \dots, \tau_l^e\}$*,*$\mathcal{T}_s = \{\tau_1^s, \dots, \tau_l^s\}$*, respectively, of the original formulas* φ_e *and* φ_s*,*
- $\varphi_e' \in \texttt{REPAIR}_{\mathcal{T}_e,\mathcal{D}_e}(\varphi_e)$*,* $\varphi_s' \in \texttt{REPAIR}_{\mathcal{T}_s,\mathcal{D}_s}(\varphi_s)$*,*

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)*.*

5. MONOLITHIC SPECIFICATIONS

Fig. 2 represents the workflow adopted to diagnose inconsistencies in the specification and provide constructive feedback to the designer. In this section, we describe our solution to Prob. 1, as summarized in Alg. 1. Given a problem $\mathcal{P}$, defined as in Sec. 4, the method `GenMILP` reformulates (5) in terms of the following MILP:

$$\begin{aligned} \underset{\mathbf{u}^H}{\text{minimize}} \quad & J(\xi^H) \\ \text{subject to} \quad & f_i^{\text{dyn}} \leq 0 \qquad i \in \{1, \dots, m_d\} \\ & f_k^{\text{stl}} \leq 0 \qquad k \in \{1, \dots, m_s\}, \end{aligned} \tag{13}$$

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

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

Algorithm 1 `DiagnoseRepair`

```
procedure DiagnoseRepair
    Input: 𝒫
    Output: u^H, 𝒟, repaired, φ'
    (J, C) ← GenMILP(𝒫), repaired ← 0
    u^H ← Solve(J, C)
    if u^H = ∅ then
        𝒟 ← ∅, 𝒮 ← ∅, I ← ∅, ℳ ← (0, C)
        while repaired = 0 do
            (𝒟', 𝒮', I') ← Diagnosis(ℳ, 𝒫)
            𝒟 ← 𝒟 ∪ 𝒟', 𝒮 ← 𝒮 ∪ 𝒮', I ← I ∪ I'
            options ← UserInput(𝒟')
            λ ← ModifyConstraints(I', options)
            (repaired, ℳ, φ') ← Repair(ℳ, I', λ, 𝒮, φ)
        u^H ← Solve(J, ℳ.C)
```

Algorithm 2 `Diagnosis`

```
procedure Diagnosis(ℳ, 𝒫)
    Input: ℳ, 𝒫
    Output: 𝒟, 𝒮, I'
    I_C ← IIS(ℳ)
    (𝒟, 𝒮) ← ExtractPredicates(I_C, 𝒫)
    I' ← ExtractConstraints(ℳ, 𝒟)
```

dynamics and the STL specification φ. We let (J, C) represent this MILP, where J is the objective, and C is the set of constraints. If Prob. (13) 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`) and get converted into a set of weights λ (`ModifyConstraints`). The designer can 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` subroutines.

5.1 Diagnosis

Our diagnosis procedure is summarized in Alg. 2. The method `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 in Eq. (13), or a relaxation thereof. This feasibility problem has the same (possibly relaxed) constraints as Eq. (13), but zero cost. Formally, we provide the following definition of a *relaxed* constraint and 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 say that $f \leq 0$ is relaxed into $f' \leq 0$. 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 to the set of STL predicates from which it originates, which is then used to construct the set $\mathcal{D} = \{\mu_1, \ldots, \mu_d\}$ 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. The set $\mathcal{D}$ will be used to produce a relaxed version of $\mathcal{M}$ as further detailed in Sec. 5.2. The procedure also returns the subset I of all the constraints in $\mathcal{M}$ that are associated with predicates in $\mathcal{D}$.

Algorithm 3 `Repair`

```
procedure Repair
    Input: ℳ, I, λ, 𝒮, φ
    Output: repaired, ℳ, φ
    ℳ.J ← ℳ.J + λ^⊤ s_I
    for c in I do
        if λ(c) > 0 then
            ℳ.C(c) ← ℳ.C(c) + s_c
    (repaired, s*) ← Solve(ℳ.J, ℳ.C)
    if repaired = 1 then
        φ ← ExtractFeedback(s*, 𝒮, φ)
```

5.2 Repair

The diagnosis procedure isolates a set of STL atomic predicates that jointly produce a source 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 Alg. 3. We formulate relaxed versions of the feasibility problem $\mathcal{M}$ associated with Eq. (13) by using *slack variables*.

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

$$\begin{aligned} \underset{\mathbf{s} \in \mathbb{R}^{|I|}}{\text{minimize}} \quad & ||\mathbf{s}|| \\ \text{subject to} \quad & f_i - s_i \leq 0 && i \in \{1, \ldots, |I|\} \\ & f_i \leq 0 && i \in \{|I|+1, \ldots, m\} \\ & s_i \geq 0 && i \in \{1, \ldots, |I|\}, \end{aligned} \tag{14}$$

where $\mathbf{s} = s_1 \ldots s_{|I|}$ is a vector of slack variables added to the set I obtained after the latest call of `Diagnosis`. Note that not all the constraints in the original optimization Eq. (13) 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 Eq. (14) 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 applications, 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 be used, including weighted norms based on the set of weights λ. If Problem (14) is feasible, `ExtractFeedback` uses the solution $\mathbf{s}^*$ to repair the original infeasible specification φ. Otherwise, an infeasible problem is returned for another round of diagnosis to retrieve further constraints to relax. Next, we provide details on the implementation of `ExtractFeedback`.

Given a minimum norm solution $\mathbf{s}^*$ to Eq. (14), the slack variables $\mathbf{s}^*$ are mapped to a set of *predicate repairs* $s_{\mathcal{D}}$, as defined in Problem 1, as follows. The slack vector $\mathbf{s}^*$ in Alg. 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 $\forall\ i \in \{1, \ldots, d\}$,

$$s_i = \mu_i^* - \mu_i = \max_{t \in \{1, \cdots, H\}} s^*_{\mu_i,t}, \tag{15}$$

H being the time horizon for (13), and $s_{\mathcal{D}} = \{s_1, \ldots, s_d\}$. To find a set of *time-interval repairs* instead, we proceed as follows:

1. The slack vector $\mathbf{s}^*$ in Alg. 3 includes the set of slack variables $\{s^*_{\mu_i,t}\}$, where $s^*_{\mu_i,t}$ is added to the optimization constraint associated with atomic predicate $\mu_i \in \mathcal{D}$ at time t. For each μ_i, with support interval σ_i, we search for the largest time interval $\sigma'_i \subseteq \sigma_i$ such that $\forall t \in \sigma'_i,\ s^*_{\mu_i,t} = 0$. If $\mu_i \notin \mathcal{D}$, 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 formula obtained from φ after these transformations[2].

3. We construct the syntactic parse tree of $\hat{\varphi}$ based on (4): each node is an operator, and the leaves are atomic predicates. The nodes of the parse tree of $\hat{\varphi}$ can 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:

$$\sigma^*_i = \begin{cases} \sigma'_i & \text{if } i \in \nu \\ \bigcap\limits_{j \in C(i)} \sigma^*_j & \text{if } i \in \kappa \cup \delta_{\mathbf{U}} \\ \sigma^*_{j \in C(i)} & \text{if } i \in \delta_{\mathbf{G}} \end{cases} \tag{16}$$

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 a $\mathbf{G}$ node has a single child. Therefore, with some abuse of notation, we use $C(i)$ in (16) 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 the set of intervals $\{\hat{\tau}_j\}$ to 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 demonstrated in Sec. 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 limits on the acceleration of the* ego *vehicle, both the cars end up entering the forbidden box at the same time. Alg. 1 chooses to repair φ_1 by adding slacks to all of its predicates, such that $\varphi'_1 = (-0.5 - s_{l1} \leq y^{\text{ego}}_t \leq 0.5 + s_{u1}) \wedge (-0.5 - s_{l2} \leq x^{\text{adv}}_t \leq 0.5 + s_{u2})$. Table 1 shows the optimal slack values at each t, while s_{u1} and s_{l2} are set to zero at all t. We conclude that the specification replacing φ_1 with φ'_1*

$$\varphi'_1 = \mathbf{G}_{[0,\infty)}\neg\big((-0.24 \leq y^{\text{ego}}_t \leq 0.5) \wedge (-0.5 \leq x^{\text{adv}}_t \leq 0.43)\big)$$

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 by $(1.5 - s_l \leq a^{\text{ego}}_t \leq 2.5 + s_u)$. In this case, we keep 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 gives $\varphi'_2 = \mathbf{G}_{[0,\infty)}\big(0.68 \leq a^{\text{ego}}_t \leq 2.5\big)$. The ego *vehicle should then slow down to prevent entering the forbidden box at the same time as the other car.*

[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 $H = 10$.

Our algorithm offers the following guarantees, for which a proof sketch is given below. The complete proofs can be found in the extended version of this paper [11].

Theorem 1 (Soundness). *Given a controller synthesis problem $\mathcal{P} = (f_d, g_d, x_0, \varphi, J)$, such that (5) is infeasible at time t, let $\varphi' \in$ `REPAIR`$_{\mathcal{D},\mathcal{T}}(\varphi)$ be the repaired formula returned from Alg. 1 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 (φ', $\mathcal{D}$, $\mathcal{T}$) satisfy the minimality conditions in Prob.1.*

Theorem 2 (Completeness). *Assume the controller synthesis problem $\mathcal{P} = (f_d, g_d, x_0, \varphi, J)$ results in (5) being infeasible at time t. If there exist a set of predicates $\mathcal{D}$ or time-intervals $\mathcal{T}$ and $\Phi \subseteq$ `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 (ϕ, $\mathcal{D}$, $\mathcal{T}$) are minimal in the sense of Problem 1, then Alg. 1 returns a repaired formula φ' in Φ.*

Proof Sketch. We start by discussing the case of soundness for *predicate repair*. Let $\mathcal{M}$ be the MILP encoding of $\mathcal{P}$ as defined in (13), $\mathcal{M}'$ be the encoding of $\mathcal{P}'$, and $\mathcal{M}''$ the feasible MILP obtained from Alg. 1, together with the optimal slack set $\{s^*_{\mu,t} | \mu \in \mathcal{D}, t \in \{1, \ldots, H\}\}$. We note that $\mathcal{M}'$ and $\mathcal{M}''$ are both relaxed versions of $\mathcal{M}$. Moreover, each constraint with a nonzero slack variable in $\mathcal{M}''$ is relaxed in $\mathcal{M}'$, and offset by the largest slack value over the horizon H. Since $\mathcal{M}''$ is feasible, $\mathcal{M}'$, and subsequently $\mathcal{P}'$, are feasible. To prove that (φ',$\mathcal{D}$) satisfy the predicate minimality condition, by Definition 4, at least one predicate in $\mathcal{D}$ generates a conflicting constraint and must be repaired. Moreover, because Alg. 1 finds all the IISs in the original optimization problem and allows relaxing any constraints in the union of the IISs, repairing any predicate outside of $\mathcal{D}$ is redundant. Therefore, if a formula $\tilde{\varphi}$ is obtained from φ after repairing a set of predicates $\tilde{\mathcal{D}}$, then the associated repair set $s_{\tilde{\mathcal{D}}}$ is seen as a repair set on the same predicate set as $s_{\mathcal{D}}$. Finally, by the norm minimization in (14), we conclude $||s_{\mathcal{D}}|| \leq ||s_{\tilde{\mathcal{D}}}||$.

We now consider the MILP formulation $\mathcal{M}'$ associated with φ' in the case of *time-interval repairs*. For each atomic predicate $\mu_i \in \mathcal{D}$, $\mathcal{M}'$ 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. Moreover, because of the structure of the MILP encoding and the manner in which slacks are added, if the constraints corresponding to the atomic predicates in $\mathcal{D}$ have slack zero, so will any constraints enforcing Boolean or temporal combinations of these predicates. Thus, $\mathcal{M}'$ is feasible. To show the satisfaction of the minimality condition, we observe that Alg. 1 selects, for each $\mu_i \in \mathcal{D}$, the largest interval σ'_i such that the associated constraints are feasible, i.e., their slack variables are zero after norm minimization. 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 $\hat{\varphi}$, and finally of the temporal operators in φ, will be maximal.

To prove completeness, we first observe that Alg. 1 always terminates with a feasible solution since the set of MILP constraints to diagnose and repair is finite. Let $\mathcal{D}$ be the

set of predicates modified to obtain $\phi \in \Phi$ and $\mathcal{D}'$ the set of diagnosed predicates returned by Alg. 1. Then, because $\mathcal{D}'$ includes all the predicates responsible for inconsistencies, as argued above, we conclude $\mathcal{D} \subseteq \mathcal{D}'$. By Eq. (14), $||s_{\mathcal{D}'}|| \leq ||s_{\mathcal{D}}||$, hence $\varphi' \in \Phi$. Further, if $\phi \in \Phi$ 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 the MILP associated with ϕ and make the overall problem feasible. Then, the relaxed MILP associated with φ after slack norm minimization will include a set of constraints admitting zero slacks over the same set of time intervals, thus terminating with a set of non-empty intervals $\mathcal{T}' = \{\tau_1', \ldots, \tau_l'\}$. Finally, because Alg. 1 finds the longest such intervals, we are guaranteed that $||\tau_i'|| \geq ||\tau_i||$ for all $i \in \{1, \ldots, l\}$, hence $\varphi' \in \Phi$ holds. $\square$

In the worst case, Alg. 1 solves a number of MILP problem instances equal to the number of atomic predicates 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 quadratic in the size (number of predicates and operators) of the STL specification.

6. CONTRACTS

In this section, we consider specifications provided in the form of a contract $(\varphi_e, \varphi_e \rightarrow \varphi_s)$, where φ_e expresses the assumptions and φ_s captures the guarantees. To repair contracts, we capture tradeoffs between assumptions and guarantees in terms of minimization of a weighted norm of slacks. We now describe our results for both non-adversarial and adversarial environments.

6.1 Non-Adversarial Environment

For a contract, we distinguish 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 $\varphi \equiv \varphi_e \rightarrow \varphi_s$, equation (6) reduces to:

$$\begin{aligned} \underset{\mathbf{u}^H}{\text{minimize}} \quad & J(\xi^H(x_0, \mathbf{u}^H, \mathbf{w}^H)) \\ \text{subject to} \quad & \xi^H(x_0, \mathbf{u}^H, \mathbf{w}^H) \models \varphi, \end{aligned} \tag{17}$$

Because of the similarity of Eq. (17) and Eq. (5), we diagnose and repair a contract using the same methodology illustrated in Sec. 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 Alg. 1, allocating different weights to predicates in the assumption and guarantee formulae. We provide the same guarantees as in Thms. 1 and 2, where $\varphi \equiv \varphi_e \rightarrow \varphi_s$ 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 $H = 10$. 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 $v_0^{ego} = 0$, 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 Sec. 5.2, we modify the conflicting predicates in the specification by using slack variables as follows: $v_t^{\text{adv}} + s_e(t) \geq 0.5$ and $v_t^{\text{ego}} + s_s(t) \geq 0.5$. Moreover, we assign 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 by falsifying it, which would provide a trivial solution. We instead prefer setting $\lambda_s < \lambda_e$, obtaining the slack values in Table 2, which leads to the following predicate repair: $\psi_{s2}' = \mathbf{G}_{[0,\infty)}(v_t^{\text{ego}} \geq -0.01)$.*

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.

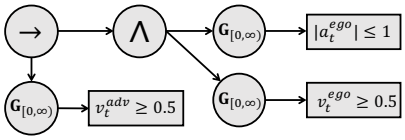


Figure 3: Parse tree of $\psi \equiv \psi_e \rightarrow \psi_s$ used in Example 2 and 5.

We can also modify the time interval of the temporal operator associated with ψ_{s2} to repair the overall specification. Based on the slack values in Table 2, we conclude $\sigma_1' = \sigma_2' = [0, 9]$ (the optimal slack values for these predicates are always zero), while $\sigma_3' = [3, 9]$. For the syntax tree in Fig. 3, we 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. Since none of the above intervals are 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, and we can keep the upper bound of the $\mathbf{G}$ operator at infinity.

6.2 Adversarial Environment

When the environment behaves adversarially, the control synthesis problem assumes the structure in (6). 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 $\varphi_w \wedge \varphi_e \rightarrow \varphi_s$, representing the contract, as $\varphi \equiv \varphi_w \rightarrow \psi$, where $\psi \equiv (\varphi_e \rightarrow \varphi_s)$. Our diagnosis and repair method is summarized in Alg. 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 Prob. (6) is feasible (`CheckSAT` routine):

$$\begin{aligned} \underset{\mathbf{u}^H, \mathbf{w}^H}{\text{minimize}} \quad & J(\xi^H(x_0, \mathbf{u}^H, \mathbf{w}^H)) \\ \text{subject to} \quad & \xi^H(x_0, \mathbf{u}^H, \mathbf{w}^H) \models \varphi \\ & \mathbf{w}^H \models \varphi_w \wedge \varphi_e. \end{aligned} \tag{18}$$

If (18) is unsatisfiable, we use the techniques introduced in Sec. 5.2 and 6.1 to diagnose and repair the infeasibility. Therefore, we assume that (18) is satisfiable, hence there exist $\mathbf{u}_0^H$ and $\mathbf{w}_0^H$ that solve (18). 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 following problem:

$$\begin{aligned} \underset{\mathbf{w}^H}{\text{minimize}} \quad & \rho^{\varphi}(\xi^H(x_0, \mathbf{u}^H, \mathbf{w}^H), 0) \\ \text{subject to} \quad & \mathbf{w}^H \models \varphi_e \wedge \varphi_w \end{aligned} \tag{19}$$

with $\mathbf{u}^H = \mathbf{u}_0^H$. The optimal $\mathbf{w}_1^H$ from (19) will falsify the specification if the resulting robustness value is below zero[3]. If this is the case, we look for a $\mathbf{u}_1^H$ which solves (17) with the additional restriction of $\mathbf{w}^H \in \mathcal{W}_{cand} = \{\mathbf{w}_1^H\}$. If this

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

Algorithm 4 `DiagnoseRepairAdversarial`

```
procedure DiagnoseRepairAdversarial
    Input: P
    Output: u^H, P'
    (J, C) ← GenMILP(P)
    (u_0^H, w_0^H, sat) ← CheckSAT(J, C)
    if sat then
        W*_cand ← SolveCEGIS(u_0^H, P)
        W_cand ← W*_cand
        while W_cand ≠ ∅ do
            P_w ← RepairAdversarial(W_cand, P)
            W_cand ← SolveCEGIS(u_0^H, P_w)
        W_cand ← W*_cand, P_ψ ← P
        while W_cand ≠ ∅ do
            P_ψ ← DiagnoseRepair(P_ψ)
            W_cand ← SolveCEGIS(u_0^H, P_ψ)
        P' ← FindMin(P_w, P_ψ)
```

step is feasible, we once again attempt to find a worst-case disturbance sequence $\mathbf{w}_2^H$ that solves (19) 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 (19) 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 (17) 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 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 Sec. 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, a basic linear search procedure is implemented as follows. Let:

$$w_u = \max_{\substack{\mathbf{w}_i^H \in \mathcal{W}_{cand} \\ t \in \{1,\dots,H-1\}}} w_{i,t} \qquad w_l = \min_{\substack{\mathbf{w}_i^H \in \mathcal{W}_{cand} \\ t \in \{1,\dots,H-1\}}} w_{i,t}, \tag{20}$$

and define $s_u = w_{\max} - w_u$ and $s_l = w_l - w_{\min}$. The differences s_u and s_l are used 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 $s_u \leq s_l$, $[w'_{\min}, w'_{\max}]$ is set to $[w_{\min}, w_u - \epsilon]$, $\epsilon \in \mathbb{R}^+$ being a suitable (small) constant; otherwise $[w'_{\min}, w'_{\max}]$ is set $[w_l + \epsilon, w_{\max}]$. We implement an improved version of the above procedure, which allows optimizations over subsets of the time sets in (20) based on the time instants at which an infeasibility occurs. Moreover, we use binary search over the range of w_t for faster convergence. Finally, we use the updated formula φ'_w to run `SolveCEGIS` again until a realizable control sequence $\mathbf{u}^H$ is found. In Alg. 4, for a predicate repair procedure, `FindMin` provides the solution with minimum slack norm over all solutions 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 $a_t^{\text{adv}} = 2$ for all $t \in \{0, \dots, H-1\}$ as the single element in $\mathcal{W}_{cand}$ for which no controller sequence is found. We then choose to tighten the environment assumptions to make the controller realizable and shrink the bounds on a_t^{adv} by using Alg. 4 (with $\epsilon = 0.01$). After a few iterations, we finally obtain $w'_{\min} = 0$ and $w'_{\max} = 1.24$, and therefore $\phi'_w = \mathbf{G}_{[0,\infty)}\big(0 \leq a_t^{\text{adv}} \leq 1.24\big)$.*

To account for the error introduced by ϵ, given $\varphi' \in \texttt{REPAIR}_{\mathcal{D},\mathcal{T}}(\varphi)$, we say that ($\varphi'$, $\mathcal{D}$, $\mathcal{T}$) are ϵ-minimal if the magnitudes of the predicate repairs (predicate slacks) or time-interval repairs differ by at most ϵ from a minimal repair in the sense of Problem 2. Assuming that `SolveCEGIS` terminates before reaching the maximum number of iterations[4], the following theorems state the properties of Alg. 4.

Theorem 3 (Soundness). *Given a controller synthesis problem $\mathcal{P} = (f_d, g_d, x_0, \varphi, J)$, such that (6) is infeasible at time t, let $\varphi' \in \texttt{REPAIR}_{\mathcal{D},\mathcal{T}}(\varphi)$ be the repaired formula returned from Alg. 4 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 (φ', $\mathcal{D}$, $\mathcal{T}$) is ϵ-minimal.*

Theorem 4 (Completeness). *Assume the controller synthesis problem $\mathcal{P} = (f_d, g_d, x_0, \varphi, J)$ results in (6) being infeasible at time t. If there exist a set of predicates $\mathcal{D}$ and time-intervals $\mathcal{T}$ such that there exists $\Phi \subseteq \texttt{REPAIR}_{\mathcal{D},\mathcal{T}}(\varphi)$ for which $\forall\ \phi \in \Phi$, $\mathcal{P}' = (f_d, g_d, x_0, \phi, J)$ is feasible at time t and (ϕ, $\mathcal{D}$, $\mathcal{T}$) is ϵ-minimal, then Alg. 4 returns a repaired formula φ' in Φ.*

Proof Sketch. When $\psi = \varphi_e \rightarrow \varphi_s$ is modified using Alg. 1, soundness and completeness are guaranteed by Thm. 1 and the termination of the CEGIS loop. Assuming Alg. 4 modifies the atomic predicates in ϕ_w, the `RepairAdversarial` routine and (20), together with the termination of the CEGIS loop, assure that φ_w is repaired in such a way that the controller is realizable and ϵ-optimal. This gives us soundness. For completeness, we 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, repeated application of (20) and `RepairAdversarial` will 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. Hence, $\varphi' \in \Phi$ holds. □

7. CASE STUDIES

We developed the toolbox DiaRY (Diagnosis and Repair for sYnthesis)[5] implementing our algorithms. DiaRY uses Yalmip [15] to formulate the optimization problems and Gurobi [1] to solve them. It interfaces to different synthesis tools, e.g., BluSTL[6] and CrSPrSTL[7]. Here, we summarize some of the results of DiaRY for diagnosis and repair.

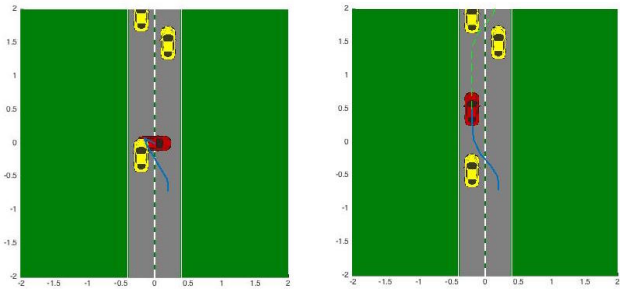
7.1 Autonomous Driving

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:

$$\begin{aligned} \dot{x} &= v\cos\theta \qquad \dot{y} = v\sin\theta \\ \dot{\theta} &= v \cdot u_1/m \qquad \dot{v} = u_2, \end{aligned} \tag{21}$$

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

[4] Under failing assumptions, Alg. 4 terminates with `UNKNOWN`.
[5] https://github.com/shromonag/DiaRY
[6] https://github.com/BluSTL/BluSTL
[7] https://github.com/dsadigh/CrSPrSTL



(a) (b)

Figure 4: Lane Change: (a) Infeasible at $t = 1.2$ s, (b) Repaired.

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 are initialized as follows: $u_0^{\text{Car 1}} = [0 \;\; 1]^\top$, $u_0^{\text{Car 2}} = [0 \;\; -0.25]^\top$, $u_0^{\text{Car 3}} = [0 \;\; 0]^\top$ and $u_0^{\text{ego}} = [0 \;\; 0]^\top$. The objective of *ego* is to safely change lane, while satisfying the following requirements:

$$\begin{aligned} \varphi_{\text{str}} &= \mathbf{G}_{[0,\infty)}(|u_1| \leq 2) && \text{Steering Bounds} \\ \varphi_{\text{acc}} &= \mathbf{G}_{[0,\infty)}(|u_2| \leq 1) && \text{Acceleration Bounds} \\ \varphi_{\text{vel}} &= \mathbf{G}_{[0,\infty)}(|v| \leq 1) && \text{Velocity Bounds} \end{aligned} \tag{22}$$

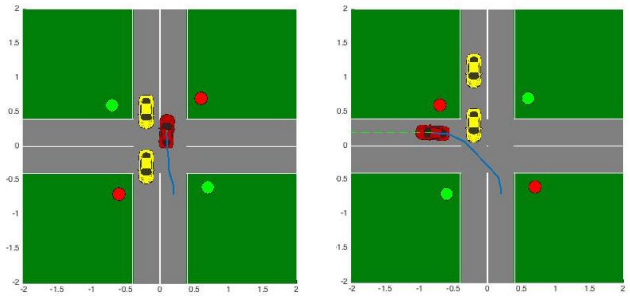
The solid blue line in Fig. 4 is the trajectory of *ego* as obtained from our MPC scheme, while the dotted green line is the future trajectory pre-computed for a given horizon at a given time. MPC becomes infeasible at time $t = 1.2$ s when the no-collision requirement is violated, and a possible collision 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. The run time is negligible with respect to the time needed to encode the original optimization problem, which is typically higher by an order of magnitude. To make the system feasible, the proposed repair increases both the acceleration bounds and the velocity bounds on the *ego* vehicle as follows:

$$\varphi_{\text{acc}}^{\text{new}} = \mathbf{G}_{[0,\infty)}(|u_2| \leq 3.5), \quad \varphi_{\text{vel}}^{\text{new}} = \mathbf{G}_{[0,\infty)}(|v| \leq 1.54).$$

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



(a) (b)

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

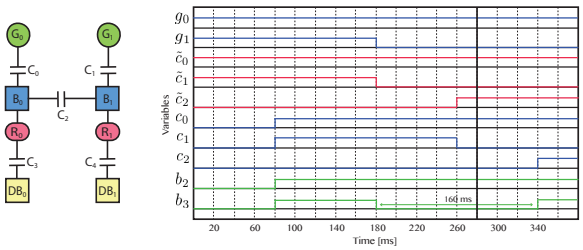


Figure 6: Simplified model of an aircraft electric power system (left) and counterexample trajectory (right). The blue, green and red lines represent environment, state, and controller variables, respectively, for a 380-ms run.

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 (22).

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 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:

$$\varphi_{\text{acc}}^{\text{new}} = \mathbf{G}_{[0,\infty)}\big(|u_2| \leq 11.903\big), \quad \varphi_{\text{vel}}^{\text{new}} = \mathbf{G}_{[0,\infty)}\big(|v| \leq 2.42\big).$$

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 [20]. 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 [20]. 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 rerout-

ing 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 (0) stands for "closed" ("open")), c_i be a set of Boolean variables denoting the state of the contactors. The system assumptions are 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 is 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 (0) stands for "available" ("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 (0) stands for "powered" ("unpowered"). Additional guarantees expressed as STL formulas, include: (i) If both AC generators are available, the left (right) AC generator shall power the left (right) AC bus. C_3 and C_4 shall be closed. (ii) If only one generator is available, all buses shall be connected to it. (iii) Two generators must never be directly connected.

We apply the diagnosis and repair procedure in Sec. 6.2 to investigate if there exists a control strategy that satisfies the specification above over all possible values of contactor delays. Fig. 6 shows 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, Alg. 4 suggests to modify our assumptions to $\mathbf{G}_{[0,\infty)}(0 \leq k_i \leq 2)$ for $i \in \{0, \ldots, 4\}$. Alternatively, by interpreting the provided counterexamples, it is possible to relax the guarantee on DB_1 to $\mathbf{G}_{[0,\infty)}(\neg b_3 \rightarrow \mathbf{F}_{[0,8]} b_3)$. The execution time was 326 s, which includes formulating and executing 3 CEGIS loops, requiring 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 mixed integer programming approach. Given an unrealizable specification, our algorithms 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, handle a broader range of environment assumptions, and apply to the control of human-in-the-loop systems as explored in [14].

9. ACKNOWLEDGMENTS

This work was partially supported by IBM and United Technologies Corporation via the iCyPhy consortium, TerraSwarm, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO, DARPA, NSF grants CCF-1139138 and CCF-1116993, and NDSEG Fellowship.

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] S. Ghosh, D. Sadigh, P. Nuzzo, V. Raman, A. Donze, A. Sangiovanni-Vincentelli, S. Sastry, and A. Seshia. Diagnosis and repair for synthesis from signal temporal logic specifications. http://arxiv.org/abs/1602.01883, Feb 2016.
[12] E. C. Kerrigan and J. M. Maciejowski. Soft constraints and exact penalty functions in model predictive control. In *Control 2000 Conference, Cambridge*, 2000.
[13] W. Li, L. Dworkin, and S. A. Seshia. Mining assumptions for synthesis. In *ACM/IEEE Int. Conf. Formal Methods and Models for Codesign*, 2011.
[14] W. Li, D. Sadigh, S. S. Sastry, and S. A. Seshia. Synthesis for human-in-the-loop control systems. In *TACAS*. 2014.
[15] J. Löfberg. Yalmip: A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.
[16] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. 2004.
[17] M. Morari, C. Garcia, J. Lee, and D. Prett. *Model predictive control*. Prentice Hall Englewood Cliffs, NJ, 1993.
[18] 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.
[19] 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.
[20] 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.
[21] 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.
[22] V. Raman and H. Kress-Gazit. Explaining impossible high-level robot behaviors. *IEEE Trans. Robotics*, 29, 2013.
[23] 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.
[24] V. Schuppan. Towards a notion of unsatisfiable cores for LTL. In *Fundamentals of Software Engineering*, 2009.
[25] P. O. Scokaert and J. B. Rawlings. Feasibility issues in linear model predictive control. *AIChE Journal*, 45(8):1649–1659, 1999.