# Robustness of Specifications and its applications to Falsification, Parameter Mining, and Runtime Monitoring with S-TaLiRo

Georgios Fainekos[1], Bardh Hoxha[2] and Sriram Sankaranarayanan[3].

[1] Arizona State University, Tempe, AZ, USA. fainekos@asu.edu
[2] Southern Illinois University, Carbondale, IL, USA. bhoxha@cs.siu.edu
[3] University of Colorado, Boulder, CO, USA.srirams@colorado.edu

**Abstract.** Logical specifications have enabled formal methods by carefully describing what is correct, desired or expected of a given system. They have been widely used in runtime monitoring and applied to domains ranging from medical devices to information security. In this tutorial, we will present the theory and application of robustness of logical specifications. Rather than evaluate logical formulas to Boolean valuations, robustness interpretations attempt to provide numerical valuations that provide degrees of satisfaction, in addition to true/false valuations to models. Such a valuation can help us distinguish between behaviors that "barely" satisfy a specification to those that satisfy it in a robust manner. We will present and compare various notions of robustness in this tutorial, centered primarily around applications to safety-critical Cyber-Physical Systems (CPS). We will also present key ways in which the robustness notions can be applied to problems such as runtime monitoring, falsification search for finding counterexamples, and mining design parameters for synthesis.

## 1 Introduction

Embedding computers in physical engineered systems has created new opportunities and at the same time major challenges. A prime example is the recent Boeing 737 MAX 8. Improving the efficiency of an existing and proven airframe led to a new design which could become unstable at higher angles-of-attack. In turn, the Maneuvering Characteristics Augmentation System (MCAS) was developed to restrict pilot inputs and protect the system from entering potentially unsafe operating regions. However, the system was not properly developed and/or tested, which led to two tragic airplane crashes with devastating human losses. In light of these accidents, further scrutiny and investigation revealed a number of software related issues [31]. Unfortunately, software related issues are not only troubling the aerospace industry, but also the autonomous vehicle industry [39], and they have been a long term issue in the medical device [45] and automotive [35] industries.

One of the main challenges on which the prior software related issues can be attributed to is that Cyber-Physical Systems (CPS) are inherently more complex than traditional computer systems. In general, the primary reason for the higher complexity is that the software or hardware interactions with the physical world cannot be abstracted away in order to use classical Boolean-based design frameworks and tools. In turn, this means that traditional software testing methods cannot be directly utilized for testing CPS software.

A decade ago, we recognized the limitations of traditional software testing methods in the context of CPS and we proposed a search based testing method explicitly targeted on testing CPS [43]. The framework proposed in [43] – sometimes also termed requirements guided falsification – falls under the broader class of search-based testing methods [36]. In brief, requirements guided falsification for CPS uses formal requirements (specifications) in temporal logic in order to guide the search for system trajectories (traces) which demonstrate that the requirement does not hold on that system (in other words the specification is falsified). In [43], the search is guided by the robustness with which a trajectory satisfies the formal requirement [27]. The robustness concept [27] captures how well a trajectory satisfies a formal requirement. Namely, the robustness measure provides a bound on how large disturbances a system trajectory can tolerate before it does not satisfy the requirement any more. Our robustness guided falsification is leveraging this property to identify regions in the search space which are more promising for the existence of falsifying system behaviors. In other words, our method is based on the principle that falsifying behaviors are more likely to exist in the neighborhood of low robustness behaviors.

Despite the apparent simplicity of robustness guided testing for CPS, the framework has been successfully utilized for a range of applications from medical devices [12] to Unmanned Aerial Vehicles (UAV) [47] and Automated Driving Systems (ADS) [46]. In addition, there is a growing community working on CPS falsification problems, e.g., see the ARCH falsification competition [24,19].

This paper provides a tutorial on the software tool S-TaLiRo [7], which started as an open source project [44] for the methods developed in [27] and [43]. Even though the temporal logic robustness computation engine [27,25] is implemented in C, S-TaLiRo is primarily a Matlab toolbox. The tutorial provides a quick overview of the S-TaLiRo support for temporal logic robustness computation [26], falsification [1,42], parameter mining [33], and runtime monitoring [17]. For other S-TaLiRo features such as conformance testing [3], elicitation and debugging of formal specifications [34,18], and robustness-guided analysis of stochastic systems [2] we refer the reader to the respective publications. The paper concludes with some current research trends.

**Beyond S-TaLiRo:** This paper accompanies a tutorial on S-TaLiRo and its applications. Whereas, S-TaLiRo remains one of the many tools in this space, there are many other tools that use temporal logic robustness for monitoring, falsification, requirements mining and parameter mining [10,36]. A detailed report on current tools and their performance on benchmark problems is available from the latest ARCH competition report on falsification tools [24]. Besides commercial tools such as the Simulink(tm) design verifier toolbox in Matlab [40] and Reactis tester(tm) [8], falsification techniques have been explored by academic tools such as Breach [21], Falstar [50] and Falsify [5], in addition to S-TaLiRo. A recent in-depth survey on runtime monitoring provides an in-depth introduction to the specification formalisms such as Signal Temporal Logic (STL) and the use of robustness for runtime monitoring, falsification and parameter mining [10]. Other recent surveys focus broadly on modeling, specification and verification techniques for Cyber-Physical Systems [14].

## 2 Systems and Signals

In S-TaLiRo, we treat a CPS as an input-output map. Namely, for a system $\Sigma$, the set of *initial operating conditions* $X_0$ and *input signals* $\mathbf{U} \subseteq U^N$, are mapped to *output signals* $Y^N$ and *timing* (or *sampling*) functions $\mathfrak{T} \subseteq \mathbb{R}_+^N$. The set $U$ is closed and bounded and contains the possible input values at each point in time (input space). Here, $Y$ is the set of output values (output space), $\mathbb{R}$ is the set of real numbers and, $\mathbb{R}_+$ the set of positive reals. The set $N \subseteq \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers, is used as a set of indexes for the finite representation (simulations) of system behavior.

A system $\Sigma$ can be viewed as a function $\Delta_\Sigma : X_0 \times \mathbf{U} \rightarrow Y^N \times \mathfrak{T}$ which takes as an input an initial condition $x_0 \in X_0$ and an input signal $u \in \mathbf{U}$ and it produces as output a signal $\mathbf{y} : N \rightarrow Y$ (also referred to as *trajectory*) and a timing function $\tau : N \rightarrow \mathbb{R}_+$. The only restriction on the timing function $\tau$ is that it must be a monotonic function, i.e., $\tau(i) < \tau(j)$ for $i < j$. The pair $\mu = (\mathbf{y}, \tau)$ is usually referred to as a *timed state sequence*, which is a widely accepted model for reasoning about real-time systems [6].

The set of all timed state sequences of a system $\Sigma$ will be denoted by $\mathcal{L}(\Sigma)$. That is, $\mathcal{L}(\Sigma) = \{(\mathbf{y}, \tau) \mid \exists x_0 \in X_0 \,.\, \exists u \in \mathbf{U} \,.\, (\mathbf{y}, \tau) = \Delta_\Sigma(x_0, u)\}$.

### 2.1 Input Signals

We assume that the input signals, if any, must be parameterizable using a finite number of parameters. This assumption enables us to define a search problem of finite dimensionality. In S-TaLiRo, the input signals are parameterized using $m$ number of control points. The control points vector $\vec{\lambda}$ and the timing vector $\vec{t}$, in conjunction with an interpolation function $\mathfrak{U}$, define the input signal $u$. Namely, at time $t$, $u(t) = \mathfrak{U}(\vec{\lambda}, \vec{t})(t)$.

The practitioner may choose different interpolation functions depending on the system and application. Example functions, as shown in Fig. 1, include linear, piecewise constant, splines, piecewise cubic interpolation, etc. If timing control points are not included, the state control points will be distributed equidistantly with respect to time with a chosen interpolation function. Otherwise, the timing of the state control points is defined by the timing vector $\vec{t}$. The timing option is illustrated in Fig. 2. Choosing the appropriate number of control points and interpolation functions is application dependent. Timing should be included in the search space whenever the system should be tested under conditions where the input variation could be high in a very short period of time. By including

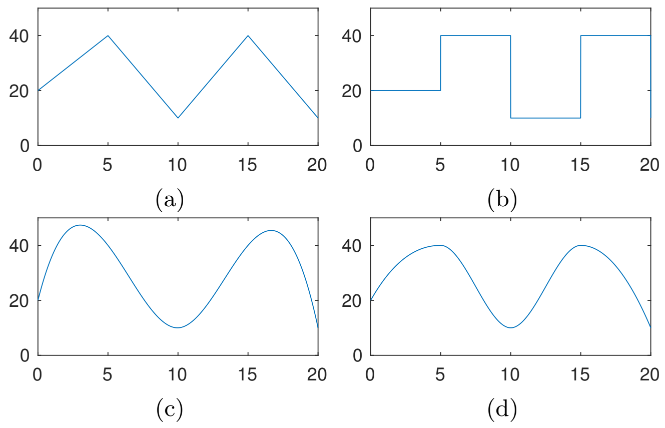


Fig. 1: Signal generation with state control points $\vec{\lambda} = [20, 40, 10, 40, 10]$ and equidistant timing control points $\vec{t} = [0, 5, 10, 15, 20]$ with various interpolation functions. (a) Linear, (b) Piecewise constant, (c) Spline, (d) Piecewise cubic interpolation.

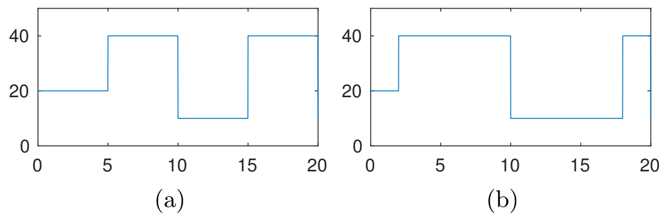


Fig. 2: Signal generation with state control points $\vec{\lambda} = [20, 40, 10, 40, 10]$ and piecewise constant interpolation. (a) With no timing control points, (b) With timing control points $\vec{t} = [0, 2, 10, 18, 20]$.

timing between control points in the search space, one may be able to produce behaviors such as jerking behavior for the gas and brake throttle of an automotive vehicle. Note that in this framework, for systems with multiple inputs, each input can have a different number of control points and interpolation function. This enables the practitioner to define a wide array of input signals.

## 2.2 Automotive Transmission (AT)

As a running example, we consider an Automatic Transmission model that is widely used as a benchmark for CPS testing and verification [1,33,20,19,24]. We modify the original Simulink model provided by Mathworks[4] slightly to enable input and output interaction with Matlab scripts and S-TaLiRo. The

[4] Simulink model discussed at: `http://www.mathworks.com/help/simulink/examples/modeling-an-automatic-transmission-controller.html`

input space of the model is the throttle schedule $u \in [0, 100]$. The physical component of the model has two continuous state variables $x$ which are also its outputs $\mathbf{y}$: the speed of the engine $\omega$ (RPM) and the speed of the vehicle $v$. The output space is $Y = \mathbb{R}^2$ with $\mathbf{y}(i) = [\omega(i)\ v(i)]^T$ for all $i$ in the simulation time. The vehicle is at rest at time 0. The model contains a Stateflow chart with two concurrently executing Finite State Machines (FSM) with 4 and 3 states, respectively. The FSM models the logic that controls the switching between the gears in the transmission system. We remark that the system is deterministic, i.e., under the same input signal $u$, we will observe the same output signal $\mathbf{y}$. For a more detailed presentation of this model see [32].

## 3 Metric Temporal Logic

Metric Temporal Logic (MTL) is an extension of Linear Temporal Logic that enables the definition of timing intervals for temporal operators. It was introduced in [37] to reason over quantitative timing properties of Boolean signals.

In addition to propositional logic operators such as conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$), MTL supports temporal operators such as next ($\bigcirc$), weak next ($\circleddash$), until ($\mathcal{U}_I$), release ($\mathcal{R}_I$), always ($\Box_I$) and eventually ($\Diamond_I$).

**Definition 1 (MTL Syntax)** *MTL syntax is defined by the grammar:*

$$\phi ::= \top \mid p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \bigcirc \phi \mid \phi_1 \mathcal{U}_I \phi_2 \tag{1}$$

*where $p \in AP$ with $AP$ being the set of atomic propositions, and $\top$ is True ($\bot = \neg\top$ is False). Also, $I$ is a nonsingular interval of the positive reals.*

MTL enables the formalization of complex requirements with respect to both state and time as presented in Table 1.

In S-TaLiRo, the user defines a specification as a string where the temporal operators $\bigcirc$, $\Box$ and $\Diamond$ are represented as $X$, [ ] and <>, respectively.

The atomic propositions in our case label subsets of the output space $Y$. In other words, each atomic proposition is a shorthand for an arithmetic expression of the form $p \equiv g(y) \leq c$, where $g : Y \to \mathbb{R}$ and $c \in \mathbb{R}$. We define an observation map $\mathcal{O} : AP \to 2^Y$ such that for each $p \in AP$ the corresponding set is $\mathcal{O}(p) = \{y \mid g(y) \leq c\} \subseteq Y$. Examples of MTL specifications for our running example AT can be found on Table 2.

### 3.1 Parametric Metric Temporal Logic

MTL specifications may also be parameterized and presented as templates, where one or more state or timing parameters are left as variables. The syntax of parametric MTL is defined as follows (see [33,9] for more details):

**Definition 2 (Syntax of Parametric MTL)** *Let $\vec{\theta} = [\theta_1 \ldots \theta_n]$ be a vector of parameters. The set of all well-formed Parametric MTL (PMTL) formulas is*

Table 1: Specifications in MTL and natural language.

| **Specification** | **Natural Language** |
|---|---|
| Safety ($\Box_{[0,\theta]}\phi$) | $\phi$ should always hold from time 0 to $\theta$. |
| Liveness ($\Diamond_{[0,\theta]}\phi$) | $\phi$ should hold at some point from 0 to $\theta$ (or now). |
| Coverage ($\Diamond\phi_1 \wedge \Diamond\phi_2 \ldots \wedge \Diamond\phi_n$) | $\phi_1$ through $\phi_n$ should hold at some point in the future (or now), not necessarily in order or at the same time. |
| Stabilization ($\Diamond\Box\phi$) | At some point in the future (or now), $\phi$ should always hold. |
| Recurrence ($\Box\Diamond\phi$) | At every point in time, $\phi$ should hold at some point in the future (or now). |
| Reactive Response ($\Box(\phi \rightarrow \psi)$) | At every point in time, if $\phi$ holds then $\psi$ should hold. |

*the set of all well-formed MTL formulas where for all $i$, $\theta_i$ either appears in an arithmetic expression, i.e., $p[\theta_i] \equiv g(y) \leq \theta_i$, or in the timing constraint of a temporal operator, i.e., $\mathcal{I}[\theta_i]$.*

We will denote a PMTL formula $\phi$ with parameters $\vec{\theta}$ by $\phi[\vec{\theta}]$. Given a vector of parameters $\vec{\theta} \in \Theta$, then the formula $\phi[\vec{\theta}]$ is an MTL formula. There is an implicit mapping from the vector of parameters $\vec{\theta}$ to the corresponding arithmetic expressions and temporal operators in the MTL formula. Once a parameter valuation is defined, a PMTL formula is transformed into an MTL formula.

## 4 Robustness of Metric Temporal Logic Formulas

Once system specifications are defined in MTL, we can utilize the theory of the robustness of MTL to determine whether a particular behavior satisfies or falsifies (does not satisfy) the specification. Furthermore, we can quantify how "close" that particular behavior is to falsification. A positive robustness value indicates that the specification is satisfied and a negative robustness value indicates that the specification is falsified.

Using a metric $d$ [28], we can define the distance of a point $x \in X$ from a set $S \subseteq X$ as follows:

**Definition 3 (Signed Distance)** *Let $x \in X$ be a point, $S \subseteq X$ be a set and $d$ be a metric on $X$. Then, we define the Signed Distance from $x$ to $S$ to be*

$$\mathbf{Dist}_d(x, S) := \begin{cases} -\min\{d(x,y) \mid y \in S\} & \text{if } x \notin S \\ \min\{d(x,y) \mid y \in X \backslash S\} & \text{if } x \in S \end{cases}$$

MTL formulas are interpreted over timed state sequences $\mu$. We let the valuation function be the depth (or the distance) of the current point of the signal

Table 2: Various specifications for the AT model [32].

| | Natural Language | MTL |
|---|---|---|
| $\psi_1$ | It is not the case that eventually, the vehicle will be in fourth gear and the speed of the vehicle is less than 50. | $\psi_1 = \neg\Diamond(g_4 \rightarrow (v < 50))$ |
| $\psi_2$ | There should be no transition from gear two to gear one and back to gear two in less than 2.5 sec. | $\Box((g_2 \wedge \bigcirc g_1) \rightarrow \Box_{(0,2.5]}\neg g_2)$ |
| $\psi_3$ | After shifting into gear one, there should be no shift from gear one to any other gear within 2.5 sec. | $\Box((\neg g_1 \wedge \bigcirc g_1) \rightarrow \Box_{(0,2.5]} g_1)$ |
| $\psi_4$ | When shifting into any gear, there should be no shift from that gear to any other gear within 2.5sec. | $\wedge_{i=1}^{4}\Box((\neg g_i \wedge \bigcirc g_i) \rightarrow \Box_{(0,2.5]} g_i)$ |
| $\psi_5$ | If engine speed is always less than $\bar{\omega}$, then vehicle speed can not exceed $\bar{v}$ in less than $T$ sec. | $\neg(\Diamond_{[0,T]}(v > \bar{v}) \wedge \Box(\omega < \bar{\omega}))$ or $\Box(\omega < \bar{\omega}) \rightarrow \Diamond_{[0,T]}(v > \bar{v})$ |
| $\psi_6$ | Within T sec the vehicle speed is above $\bar{v}$ and from that point on the engine speed is always less than $\bar{\omega}$. | $\Diamond_{[0,T]}((v \geq \bar{v}) \wedge \Box(\omega < \bar{\omega}))$ |
| $\psi_7$ | A gear increase from first to fourth in under 10secs, ending in an RPM above $\bar{\omega}$ within 2 seconds of that, should result in a vehicle speed above $\bar{v}$. | $((g_1\, \mathcal{U}\, g_2\, \mathcal{U}\, g_3\, \mathcal{U}\, g_4) \wedge \Diamond_{[0,10]}(g_4 \wedge \Diamond_{[0,2]}(\omega \geq \bar{\omega}))) \rightarrow \Diamond_{[0,10]}(g_4 \wedge \bigcirc(g_4\, \mathcal{U}_{[0,1]}\, (v \geq \bar{v})))$ |

$\omega$: Engine rotation speed, $v$: vehicle velocity, $g_i$ : gear $i$. Recommended values: $\bar{\omega}$ : 4500, 5000, 5200, 5500 RPM; $\bar{v}$ : 120, 160, 170, 200 mph; $T$ : 4, 8, 10, 20 sec
$\Box$: Always, $\Diamond$: Eventually, $\mathcal{U}$: Until, $\bigcirc$: Next

$\mathbf{y}(i)$ in the set $\mathcal{O}(p)$ labeled by the atomic proposition $p$. This robustness estimate over a single point can be extended to all points on a trajectory by applying a series of min and max operations over time. This is referred to as the robustness estimate and is formally presented in Definition 4. The robustness estimate defines how much of a perturbation a signal can tolerate without changing the Boolean truth value of the specification.

For the purposes of the following discussion, we use the notation $[\![\phi]\!]$ to denote the robustness estimate with which the timed state sequence $\mu$ satisfies the specification $\phi$. Formally, the valuation function for a given formula $\phi$ is $[\![\phi]\!] : Y^N \times \mathfrak{T} \times N \rightarrow \overline{\mathbb{R}}$. In the definition below, we also use the following notation: for $Q \subseteq R$, the *preimage* of $Q$ under $\tau$ is defined as: $\tau^{-1}(Q) := \{i \in N \mid \tau(i) \in Q\}$. Also, given an $\alpha \in \mathbb{R}$ and $\mathcal{I} = \langle l, u\rangle$, we define the timing interval shift operation as $\alpha + \mathcal{I} = \langle \alpha + l, \alpha + u\rangle$. Here, $\langle$ and $\rangle$ are used to denote brackets or parentheses for closed and open intervals.

**Definition 4 (Robustness Estimate [28])** *Let $\mu = (\mathbf{y}, \tau) \in Y^{[0,T]}$, and $i, j, k \in N$, then the robustness estimate of any formula MTL formula is defined as:*

$$
\begin{aligned}
[\![\top]\!](\mu, i) &:= +\infty \\
[\![p]\!](\mu, i) &:= \mathbf{Dist}_d(\mathbf{y}(i), \mathcal{O}(p)) \\
[\![\neg\phi]\!](\mu, i) &:= -[\![\phi]\!](\mu, i) \\
[\![\phi_1 \vee \phi_2]\!](\mu, i) &:= \max([\![\phi_1]\!](\mu, i), [\![\phi_2]\!](\mu, i)) \\
[\![\bigcirc\phi]\!](\mu, i) &:= \begin{cases} [\![\phi]\!](\mu, i+1) & \textit{if } i+1 \in N \\ -\infty & \textit{otherwise} \end{cases}
\end{aligned}
$$

$$[\![\phi_1 \mathcal{U}_{\mathcal{I}} \phi_2]\!](\mu, i) := \max_{j \in \tau^{-1}(\tau(i)+\mathcal{I})} \Big(\min([\![\phi_2]\!](\mu, j), \min_{i \le k < j} [\![\phi_1]\!](\mu, k))\Big)$$

When $i = 0$, then we write $[\![\phi]\!](\mu)$. With $[\![\phi]\!](\Sigma)$, We denote the system robustness as the minimum robustness over all system behaviors.

$$[\![\phi]\!](\Sigma) = \min_{\mu \in \mathcal{L}(\Sigma)} [\![\phi]\!](\mu) \tag{2}$$

In S-TaLiRo, the robustness of an MTL formula with respect to a timed state sequence is computed using two algorithms. The first algorithm, *dp_taliro* [25], uses a dynamic programming algorithm to compute the robustness in segments, iteratively . The second algorithm, *fw_taliro* [28], uses formula rewriting techniques. This approach maintains a state of the formula with respect to time, however, at a significant computation cost. If we consider the robustness estimate over systems, the resulting robustness landscape can be both nonlinear and non-convex. An example of the robustness landscape for an MTL specification is illustrated in Fig. 3.

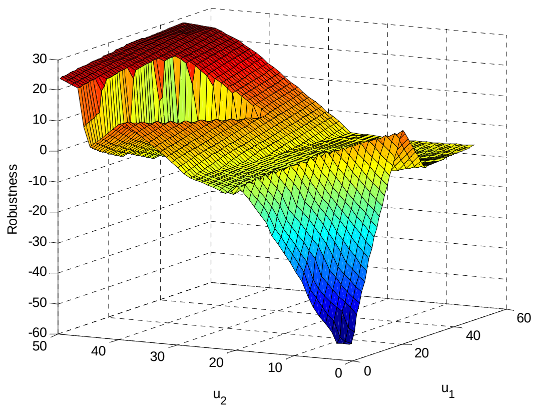


Fig. 3: Robustness estimate landscape for the AT model and specification $\phi_{AT} = \neg(\Diamond_{[0,30]}(v > 100) \wedge \Box(\omega \le 4500)) \wedge \neg\Diamond_{[10,40]}\Box_{[0,5]}(60 < v \le 80) \wedge \neg\Diamond_{[50,60]}\Box_{[0,3]}(v \le 60)$. The input signal to the system is generated by linearly interpolating control points $u_1$, $u_2$ at time 0 and 60, respectively, for the throttle input $u$. That is, $u(t) = \frac{60-t}{60}u_1 + \frac{t}{60}u_2$.

We note that a similar notion of robustness is presented in [22] for Signal Temporal Logic (STL) formulas [41]. While between the two approaches the robust interpretation (semantics) for predicates of the form $x < a$ is identical, the two approaches differ over arbitrary predicates of form $f(x) < 0$. Using the notion of robustness in Def. 4, predicates of the form $f(x) < 0$ are interpreted as the signed distance of the current point $x$ from the set $\{x \mid f(x) < 0\}$. On the other hand, predicates of the form $f(x) < 0$ are not directly supported by the theory as introduced in [22]. If the robustness of $f(x) < 0$ is simply defined as $f(x)$, then the robustness estimate is not guaranteed to define a robustness tube within which all other trajectories satisfy the same property. Nevertheless, for both semantics, positive robustness implies Boolean satisfaction, while negative robustness implies falsification.

## 5 Falsification with S-TaLiRo

The problem of determining whether a CPS $\Sigma$ satisfies a specification $\phi$ is an undecidable problem, i.e. there is no general algorithm that terminates and returns whether $\Sigma \models \phi$. Therefore, it is not possible to determine exactly the minimum robustness over all system behaviors. However, by repeatedly testing the system, we can check whether a behavior that does not satisfy the specification exists. In other words, we try to find a counter-example or falsifying example that proves that the system does not satisfy the specification within a set number of tests. The MTL falsification problem is presented as follows:

**Problem 1 (MTL Falsification)** *Given an MTL formula $\phi$ and a system $\Sigma$, find initial conditions and input signals such that, when given to $\Sigma$, generate a trajectory that does not satisfy $\phi$. Formally, find $x_0 \in X_0$, $u \in \mathbf{U}$, where $\mu = \Delta_\Sigma(x_0, u)$ and $[\![\phi]\!](\mu) < 0$ (or with Boolean semantics $\mu \not\models \phi$).*

In S-TaLiRo [7,44], this is defined as an optimization problem that uses the notion of MTL robustness to guide the search. To solve this problem, an automated test case generation framework is utilized (see Fig. 4). S-TaLiRo takes as input a model, an MTL specification and a well-defined search space over the system inputs and initial conditions. Then, a stochastic optimization algorithm generates a point $x_0$ for the initial conditions and input signal $u$. These are given to the system model which generate an execution trace (output trajectory and timing function). By analyzing the execution trace, a robustness value is computed. This is then used by the stochastic optimization algorithm to select the next sample until a falsifying trajectory is generated or the maximum number of test is reached. The algorithm will return the least robust system behavior with the corresponding input signal and initial conditions.

### 5.1 Falsification with the Hybrid Distance

For falsification of specifications such as $\psi_1 = \neg\Diamond(g_4 \rightarrow (v < 50))$ over the AT model, where there is an implication relation and the antecedent is over a discrete mode of the system rather than a continuous state, the robustness estimate from Def. 4 does not provide sufficient information to the stochastic optimizer to find a falsifying behavior. In fact, the formula may be trivially satisfied since the search space that pushes the system to gear four is never explored. Therefore,

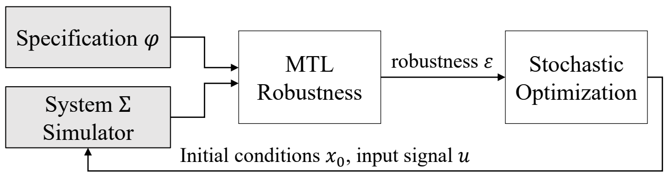


Fig. 4: The falsification framework in S-TaLiRo. Once a system $\Sigma$, initial conditions $x_0$, and input signals $u$ are given, an output trajectory $\mathbf{y}$ is generated. The output trajectory $\mathbf{y}$ is then analyzed with respect to a specification and a robustness estimate $\varepsilon$ is produced. This robustness estimate is then used by the stochastic optimizer to generate a new initial condition and input signal with the goal of minimizing the system robustness.

the antecedent is false and the formula evaluates to true. For such specifications, in S-TaLiRo, a hybrid distance metric may be utilized to attempt falsification. In this case, we assume that the user has information on the logical modes of the model including a connectivity graph $G$ and transition guards. Now, the output space becomes a hybrid space $Y = \{g_1, g_2, g_3, g_4\} \times \mathbb{R}^2$.

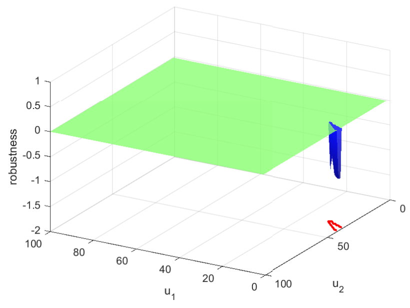
The hybrid distance metric is defined as a piecewise function. In the case when the current mode of the system is not in the mode where the specification can be falsified, then the hybrid distance is composed of two components. The first contains the number of hops/transitions from the current mode to the mode where falsification may occur. The second component is the continuous distance to the guard transition in the shortest path to the falsifying mode. In the case where falsification may occur in the current mode, the hybrid distance metric is the robustness estimate from Def. 4. For a detailed, formal presentation of the hybrid distance see [1].

In Fig. 5, we illustrate the robustness landscape from Def. 4 and the hybrid distance. While one is flat, offering little information to the stochastic optimizer, the other one has a gradient that leads to falsification. Note that the hybrid distance there is mapped to a real value using a weighting function that emphasizes the number of hops to the mode where falsification may occur.

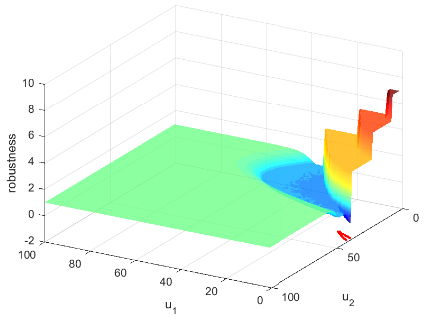
## 6 Parameter Mining

Parameter mining refers to the process of determining parameter valuations for parametric MTL formulas for which the specification is falsified. The parameter mining problem can be viewed as an extension of the falsification problem, where not only are we interested in finding falsifying behaviors for a specific parameter valuation of a parametric MTL formula, but we are interested in finding falsifying behaviors for a range of parameter valuations. In other words, we answer the question of what parameter ranges cause falsification.

Our high-level goal is to explore and infer properties that a system does not satisfy. We assume that the system designer has partial understanding about the



(a)



(b)

Fig. 5: Robustness landscape for the specification $\psi_1 = \neg\Diamond(g_4 \rightarrow (v < 50))$ over the AT model using the a) euclidean robustness estimate and b) hybrid robustness estimate.

properties that the system satisfies (or does not satisfy) and would like to be able to determine these properties precisely. The practical benefits of this method are twofold. One, it allows for the analysis and development of specifications. In many cases, system requirements are not well formalized by the initial system design stages. Two, it allows for the analysis and exploration of system behavior. If a specification can be falsified, then it is natural to inquire for the range of parameter values that cause falsification. That is, in many cases, the system design may not be modified, but the guarantees provided should be updated.

The parameter mining problem is formally defined as follows.

**Problem 2 (MTL Parameter Mining)** *Given a parametric MTL formula $\phi[\vec{\theta}]$ with a vector of $m$ unknown parameters $\vec{\theta} \in \Theta = [\underline{\vec{\theta}}, \overline{\vec{\theta}}]$ and a system $\Sigma$, find the set $\Psi = \{\vec{\theta}^* \in \Theta \mid \Sigma$ does not satisfy $\phi[\vec{\theta}^*]\}$.*

That is, the solution to Problem 2 is the set $\Psi$ such that for any parameter $\vec{\theta}^*$ in $\Psi$ the specification $\phi[\vec{\theta}^*]$ does not hold on system $\Sigma$. In other words, it is the set of parameter valuations for which the system is falsified. In the following, we refer to $\Psi$ as the parameter falsification domain.

## 6.1 Monotonicity of parametric MTL

In S-TaLiRo, we solve this problem for a class of monotonic parametric MTL specifications. For these specifications, as you increase the parameter valuation, the robustness of the system is either non-increasing or non-decreasing. The first step in the parameter mining algorithm in S-TaLiRo is to automatically determine the monotonicity of the parametric MTL specification. A formal result on the monotonicity problem is presented next.

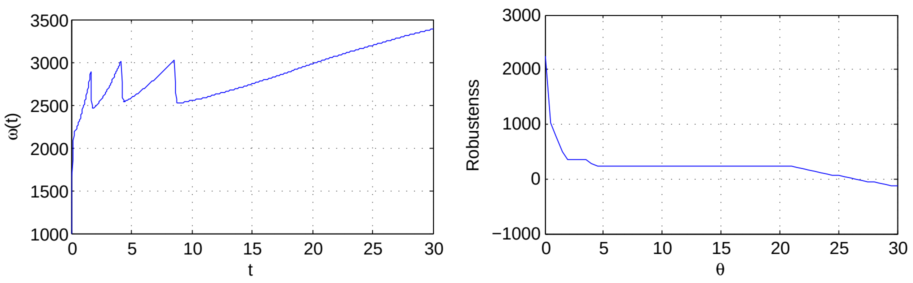


Fig. 6: Left: An output trajectory of the AT model for engine speed $\omega(t)$ for constant input throttle $u(t) = 50$; Right: corresponding robustness estimate of the specification $\Box_{[0,\theta]}(\omega \leq 3250)$ with respect to $\theta$.

**Theorem 1 (Monotonicity of parameteric MTL)** *Consider a PMTL formula $\psi[\vec{\theta}]$, where $\vec{\theta}$ is a vector of parameters, such that $\psi[\vec{\theta}]$ contains temporal subformulas $\phi[\vec{\theta}] = \phi_1[\vec{\theta}]\mathcal{U}_{\mathcal{I}[\theta_s]}\phi_2[\vec{\theta}]$, or propositional subformulas $\phi[\vec{\theta}] = p[\vec{\theta}]$. Then, given a timed state sequence $\mu = (\mathbf{y}, \tau)$, for $\vec{\theta}$, $\vec{\theta}' \in \overline{\mathbb{R}}^n_{\geq 0}$, such that $\vec{\theta} \leq \vec{\theta}'$, where $1 \leq j \leq n$, and for $i \in N$, we have:*

- *if for all such subformulas (i) $\max \mathcal{I}(\theta_s) = \theta_s$ or (ii) $p[\vec{\theta}] \equiv g(x) \leq \vec{\theta}$, then $[\![\phi[\vec{\theta}]]\!](\mu, i) \leq [\![\phi[\vec{\theta}']]\!](\mu, i)$, i.e., function $[\![\phi[\vec{\theta}]]\!](\mu, i)$ is non-decreasing with respect to $\vec{\theta}$,*
- *if for all such subformulas (i) $\min \mathcal{I}(\theta_s) = \theta_s$ or (ii) $p[\vec{\theta}] \equiv g(x) \geq \vec{\theta}$, then $[\![\phi[\vec{\theta}]]\!](\mu, i) \geq [\![\phi[\vec{\theta}']]\!](\mu, i)$, i.e., function $[\![\phi[\vec{\theta}]]\!](\mu, i)$ is non-increasing with respect to $\vec{\theta}$.*

Consider the parametric MTL formula $\phi[\theta] = \Box_{[0,\theta]}p$ where $p \equiv (\omega \leq 3250)$. The function $[\![\phi[\theta]]\!](\mu)$ is non-increasing with respect to $\theta$ [33,9]. Intuitively, this relationship holds since by extending the value of $\theta$ in $\phi[\theta]$, it becomes just as or more difficult to satisfy the specification. In Fig. 6, this is illustrated using a single output trajectory over the AT model. However, using Theorem. 1, we know that the monotonicity of the specification holds over all system behaviors.

## 6.2 Robustness-Guided Parameter Mining

To solve the parameter mining problem, we utilize the theory of robustness of MTL specifications to pose it as an optimization problem.

In Fig. 7, the estimated robustness landscape over parameter valuations is presented for the specification $\phi[\theta] = \Box_{[0,\theta]}(v < 120)$. The graph was generated by conducting falsification at parameter valuations 0 to 30. For each parameter valuation, 100 tests are conducted (hence estimated). Although we cannot calculate the exact robustness landscape, from Theorem 1, we know that the monotonicity of the specification is non-increasing. By increasing the value of

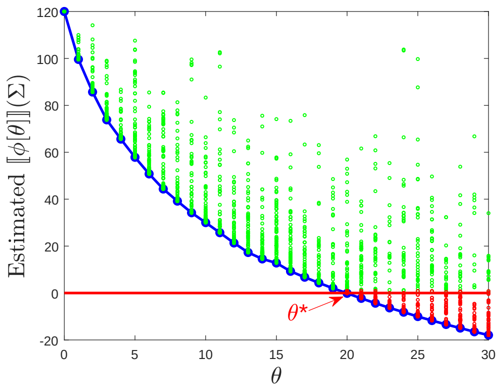


Fig. 7: Estimate of the robustness landscape for specification $\phi[\theta] = \Box_{[0,\theta]}(v < 120)$ over the AT model. The figure is generated by running a falsification algorithm for parameter valuations 0 to 30. The red line drawn at 0 marks the boundary between satisfaction and falsification. The green (red) dots represent trajectories over different input signals that satisfied (falsified) the specification.

$\theta$ you extend the time bounds for which $(v < 120)$ has to hold, and therefore the robustness cannot increase. Of particular interest is the parameter valuation where the robustness line intersects with 0, that is, the point where the specification switches from satisfied to falsified. Formally, in order to solve Problem 2, we solve the following optimization problem:

$$\begin{aligned} &\text{optimize} && f(\vec{\theta}) \\ &\text{subject to} && \vec{\theta} \in \Theta \text{ and } [\![\phi[\vec{\theta}]]\!](\Sigma) = \min_{\mu \in \mathcal{L}_\tau(\Sigma)} [\![\phi[\vec{\theta}]]\!](\mu) \leq 0 \end{aligned} \tag{3}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a non-increasing ($\geq$) or a non-decreasing ($\leq$) function.

The function $[\![\phi[\vec{\theta}]]\!](\Sigma)$, which is the robustness of the system for a parameter valuation over all system behaviors, cannot be computed using reachability analysis algorithms nor is known in closed form for the systems we are considering. Therefore, we have to compute an under-approximation of $\Theta^*$. We reformulate an optimization problem that can be solved using stochastic optimization methods. In particular, we reformulate the optimization problem (3) into a new one where the constraints due to the specification are incorporated into the cost function:

$$\text{optimize}_{\vec{\theta} \in \Theta} \left( f(\vec{\theta}) + \begin{cases} \gamma \pm [\![\phi[\vec{\theta}]]\!](\Sigma) \text{ if } [\![\phi[\vec{\theta}]]\!](\Sigma) \geq 0 \\ 0 \quad \text{otherwise} \end{cases} \right) \tag{4}$$

where the sign ($\pm$) and the parameter $\gamma$ depend on whether the problem is a maximization or a minimization problem. The parameter $\gamma$ must be properly chosen so that the solution of the problem in Eq. (4) is in $\Theta$ if and only if $[\![\phi[\vec{\theta}]]\!](\Sigma) \leq 0$. Therefore, if the problem in Eq. (3) is feasible, then the optimal points of Eq. (3) and Eq. (4) are the same. For more details on Eq. (4), see [33].

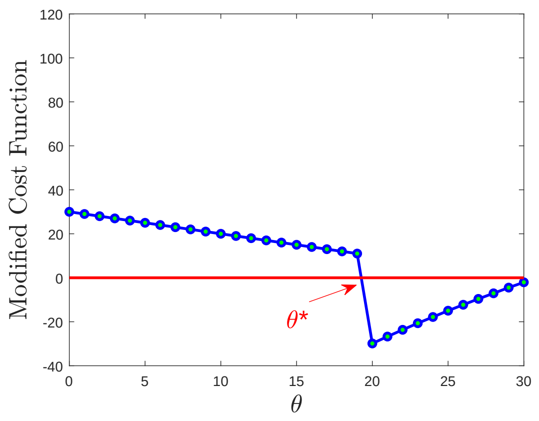


Fig. 8: The modified cost function for parameter mining for the specification $\phi[\theta] = \Box_{[0,\theta]}(\nu < 120)$ over the AT model. The solution to the optimization problem in Eq. (4) returns $\theta^*$

For specifications with more than one parameter, the robustness landscape over the parameters forms a Pareto front. One inefficient and potentially misleading approach for generating the parameter falsification domain is by running a falsification algorithm for a set number of parameter valuations. For example, consider the parameter falsification domain in Fig. 9 (Left) for specification $\phi[\vec{\theta}] = \neg(\Diamond_{[0,\theta_1]} \wedge \Box(\omega < \theta_2))$. The figure was generated by running the falsification algorithm for 200 iterations for every green/red dot. This approach is computationally very expensive and this specific example took 52 hours to compute on a computer with an I7 4770k CPU and 16GB of RAM. Furthermore, this approach may be misleading. It is possible that for a particular parameter valuation, falsification fails when in fact there exists falsifying behavior. For example, in Fig. 9 (Left), the green dot for the parameter valuation [36,4360], i.e. specification $\phi[36, 4360] = \neg(\Diamond_{[0,36]} \wedge \Box(\omega < 4360))$, has falsifying behavior. We know this since there exists falsifying behavior for the red dot for parameter valuation [34,4190]. From Theorem 1, we know that the specification has a non-increasing robustness with respect to parameters. We say that the parameter valuation [34,4190] dominates [36,4360] in terms of falsification because if there exists a trajectory $\mu$ such that $\mu \not\models \phi[34, 4190]$ then $\mu \not\models \phi[36, 4360]$.

In S-TaLiRo, we provide two efficient approaches to explore the parameter falsification domain iteratively. The Robustness-Guided Parameter Falscation Domain Algorithm (RGDA) and the Structured Parameter Falsication Domain Algorithm (SDA). In RGDA, parameters weights are utilized to guide the search towards unexplored areas of the parameter falsification domain. In SDA, the search is finely structured and does not depend on randomized weights. For details and analysis on the two algorithms, see [33]. The parameter falsification domain in Fig. 9 (Right) was computed with the RGDA algorithm with 100 iterations in 52 minutes.

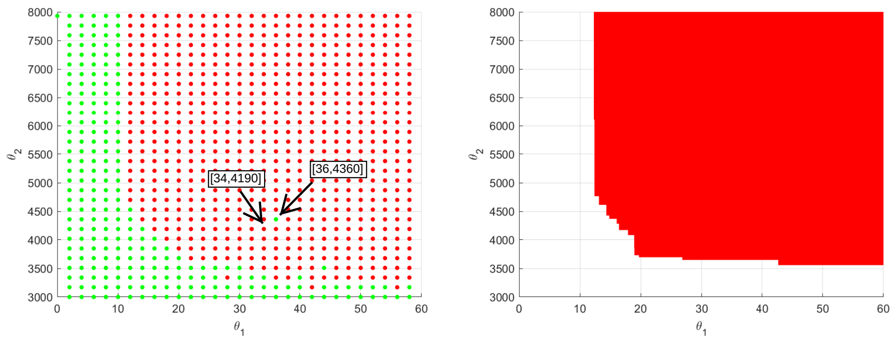


Fig. 9: The parameter falsification domain $\Psi$ for the specification $\phi[\vec{\theta}] = \neg(\Diamond_{[0,\theta_1]} \wedge \Box(\omega < \theta_2))$ over the AT model for parameter valuations 0 to 60 for $\theta_1$ and 3000 to 8000 for $\theta_2$. Left: The figure is generated by running the falsification algorithm with 100 iterations for each dot in the figure. The green (red) dots represent the minimum robustness over 100 iterations that satisfied (falsified) the specification. Right: The figure is generated using the RGDA algorithm. The red area represents the parameter falsification domain, i.e. parameter valuations for which the specification is falsified.

## 7 Runtime Monitoring

The discussion so far was primarily concerned with applications of offline monitoring of temporal logic robustness. However, many applications require *online* or *runtime* monitoring of robustness. In offline monitoring, the whole system trace is available offline for analysis, which means that it is possible to move forward and backward in time in order to compute its robustness with respect to a given specification. On the other hand, in runtime verification, the data become available during system execution.

Therefore, it is not clear that future time formal languages can always capture requirements in a meaningful way. For example, let us consider the future tense requirement that "*the RPM can remain higher than 2000 for more than 5 sec only when the vehicle is in gear 4.*" Formally, the requirement is $\varphi = \Box(\neg g_4 \rightarrow \Diamond_{[0,5]}(\omega < 2000))$ which is equivalent to $\Box(g_4 \vee \Diamond_{[0,5]}(\omega < 2000))$. If at time $t$ the gearbox is not in the fourth gear, then we can only know at time $t + 5$ if the requirement was violated. From a requirements perspective, it is desirable to check at time $t$ if the requirement is satisfied or violated. In other words, we may want to monitor an invariant like "*at least once in the past 5 seconds, either the system was in gear four or the RPM was below 2000.*" The past tense requirement would guarantee that at time $t$ we have not observed an interval that the RPM was above 2000 and the system was not in gear four.

In [17], we developed a Simulink block in S-TaLiRo which can perform runtime robustness monitoring for past-future time MTL specifications. Namely, the Simulink block enables past time formula monitoring on data generated by

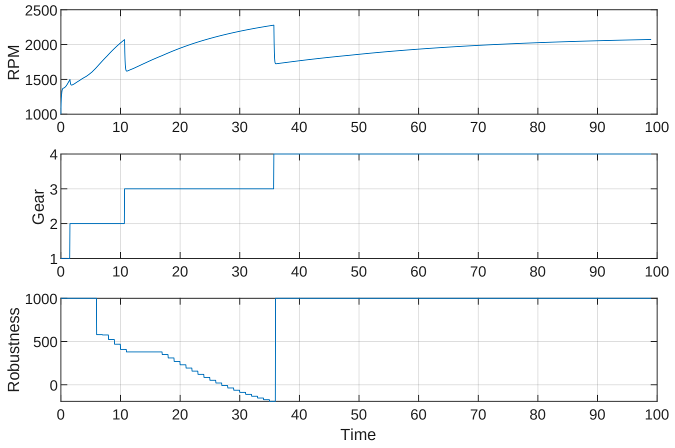


Fig. 10: S-TaLiRo runtime monitoring on the automatic transition demo.

a model or by a physical system interfaced with Simulink. In addition, if the Simulink model contains a prediction model with a prediction finite-time horizon $h$, then the user can express requirements with unbounded past and bounded future time horizon. In more detail, we have extended the syntax of MTL with past time operators such as "the previous sample satisfies $\phi$ ($\odot\phi$), "$\phi_1$ since $\phi_2$" ($\phi_1 \mathcal{S} \phi_2$), "sometime in the past $\phi$" ($\diamondminus\phi \equiv \top\mathcal{S}\phi$), and "always in the past $\phi$" ($\boxminus\phi \equiv \neg\diamondminus\neg\phi$). All these past-time operators can be additionally restricted though timing constraints as in the case of the future time operators. Details on the semantics of these operators can be found in [17].

Figure 10 presents the output of the S-TaLiRo Simulink monitoring block when applied to the Simulink automatic transmission model. For this demonstration, the throttle input has been set to a constant value of 20 while the brake input to a constant value of 0. The invariant checked is for the specification $\varphi = \diamondminus_{[0,5]}(g_4 \vee (\omega < 2000))$ and its robustness is plotted in Fig. 10. Notice that about time 21.6 the RPM exceed the 2000 threshold while the gear is still in three. The instantaneous robustness value of $\varphi$ drops below zero 5 sec later as expected. When the system switches into gear four, the robustness of $\varphi$ immediately becomes positive again. We remark that the robustness value of predicates with Boolean interpretation, e.g., $gear = 4$, must be mapped to some arbitrary large value within Simulink. Formalization of the robust interpretation of such predicates through hybrid distances (see Sec. 5.1) is not straightforward in Simulink, but potentially using input-output types [29] the process can become more systematic.

## 8 Future Directions

We will now turn our attention to problems surrounding learning-enabled and autonomous systems, which will form an important application area for many of the techniques detailed thus far. Autonomous systems such as self-driving cars are increasingly common on our streets. They rely heavily on machine learning components such as neural networks for detecting other vehicles, obstacles, pedestrians, traffic signals and signs from a combination of image and LiDAR data [38]. Besides perception, neural networks are also increasingly used as controllers that directly output steering and throttle commands to the vehicle [11]. Numerous accidents involving autonomous vehicles motivate the need for ensuring the safety of these systems [39]. At the same time, the key challenge lies in the lack of detailed component-wise specifications for the neural networks. In fact, most specifications are "end-to-end" high level specifications such as "the vehicle should brake if it detects a pedestrian in front".

Falsification approaches are increasingly being used to tackle the issue of missing specifications by using generative models tied in with rendering tools that can create realistic inputs to these systems with known "ground truth". Falsification tools including S-TaLiRo have been employed directly to find corner cases that can cause these systems to potentially fail [4,46,23,30]. However, the key challenges posed by these applications are numerous:
**(a)** Falsification techniques have been designed primarily for control systems. The use of neural networks poses many challenges requiring a better formulation of the robustness concept that encompasses the robustness of classifiers as well as better stochastic search techniques [48]. Regarding the former, some first steps have been taken in [16] by defining a new formal logic for perception systems.
**(b)** Simply providing a falsifying scenario does not solve the issue of designing safe systems. Unlike human designed components, it is nearly impossible to localize failures to a bad value of a parameter or incorrect logic in software. Often neural networks have to be retrained. This requires us to consider a family of falsifying scenarios that can provide new training data for retraining the network to hopefully correct the underlying issue. Preliminary approaches have been proposed that involve repeated application of falsification search and retraining [13,49]. However, a lot of open challenges remain before this problem can be considered satisfactorily resolved.
**(c)** The problem of helping developers understand *root causes* for falsification is yet another important future challenge in this area. Current approaches for understanding root causes are at their infancy [15]. Ideas from other fields such as explainable machine learning and natural language processing are needed to tackle the challenge of producing human understandable explanations of failures.

## 9 Conclusions

Robustness of temporal logic specification provides a systematic way of defining real-valued semantics to denote the degree of satisfaction of a specification by a trace of the system. Starting from robustness, we present important applications including falsification, parameter mining, runtime monitoring and safe autonomy. This area continues to be active with new challenges arising from the rapid emergence of learning-enabled autonomous systems. Future work in this area will continue to draw upon ideas from diverse areas including machine learning, robotics, natural language processing and human factors.

**Acknowledgments:** GF acknowledges support from NSF award 1350420. SS acknowledges support from NSF award numbers 1646556, 1815983 and the Air Force Research Laboratory (AFRL). All opinions expressed are those of the authors and not necessarily of the US NSF or AFRL.

## References

1. Abbas, H., Fainekos, G.E., Sankaranarayanan, S., Ivancic, F., Gupta, A.: Probabilistic temporal logic falsification of cyber-physical systems. ACM Transactions on Embedded Computing Systems **12**(s2) (May 2013)
2. Abbas, H., Hoxha, B., Fainekos, G., Ueda, K.: Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems. In: IEEE 4th Annual International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER) (2014)
3. Abbas, H., Mittelmann, H., Fainekos, G.: Formal property verification in a conformance testing framework. In: 12th ACM-IEEE International Conference on Formal Methods and Models for System Design (2014)
4. Abbas, H., O'Kelly, M., Rodionova, A., Mangharam, R.: Safe at any speed: A simulation-based test harness for autonomous vehicles. In: CyPhy'17 (2017)
5. Akazaki, T., Liu, S., Yamagata, Y., Duan, Y., Hao, J.: Falsification of cyber-physical systems using deep reinforcement learning. In: Formal Methods. Lecture Notes in Computer Science, vol. 10951, pp. 456–465. Springer (2018)
6. Alur, R., Courcoubetis, C., Dill, D.: Model-checking for real-time systems. In: Mitchell, J. (ed.) 5th Annual IEEE Symp. on Logic in Computer Science (LICS). pp. 414–425. IEEE Computer Society Press (June 1990)
7. Annapureddy, Y.S.R., Liu, C., Fainekos, G.E., Sankaranarayanan, S.: S-taliro: A tool for temporal logic falsification for hybrid systems. In: Tools and algorithms for the construction and analysis of systems. LNCS, vol. 6605, pp. 254–257 (2011)
8. Anonymous: Model-based testing and validation of control software with reactis (2003), `http://www.reactive-systems.com/papers/bcsf.pdf`
9. Asarin, E., Donzé, A., Maler, O., Nickovic, D.: Parametric identification of temporal properties. In: Runtime Verification. LNCS, vol. 7186, pp. 147–160 (2012)
10. Bartocci, E., Deshmukh, J., Donze, A., Fainekos, G., Maler, O., Nickovic, D., Sankaranarayanan, S.: Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications. In: Lectures on Runtime Verification. LNCS, vol. 10457, pp. 135–175 (2018)
11. Bojarski, M., Testa, D.D., Dworakowski, D., et al.: End to end learning for self-driving cars. CoRR **abs/1604.07316** (2016)
12. Cameron, F., Fainekos, G., Maahs, D.M., Sankaranarayanan, S.: Towards a verified artificial pancreas: Challenges and solutions for runtime verification. In: Runtime Verification. LNCS, vol. 9333, pp. 3–17. Springer (2015)
13. Claviere, A., Dutta, S., Sankaranarayanan, S.: Trajectory tracking control for robotic vehicles using counterexample guided training of neural networks. In: ICAPS. pp. 680–688. AAAI Press (2019)
14. Deshmukh, J., Sankaranarayanan, S.: Formal techniques for verification and testing of cyber-physical systems. In: Design Automation of Cyber-Physical Systems. pp. 69–105. Springer-Verlag (2019)

15. Diwakaran, R.D., Sankaranarayanan, S., Trivedi, A.: Analyzing neighborhoods of falsifying traces in cyber-physical systems. In: Intl. Conference on Cyber-Physical Systems (ICCPS). pp. 109–119. ACM Press (2017)
16. Dokhanchi, A., Amor, H.B., Deshmukh, J.V., Fainekos, G.: Evaluating perception systems for autonomous vehicles using quality temporal logic. In: Runtime Verification (RV). LNCS, vol. 11237. Springer (2018)
17. Dokhanchi, A., Hoxha, B., Fainekos, G.: On-line monitoring for temporal logic robustness. In: Runtime Verification. LNCS, vol. 8734, pp. 231–246. Springer (2014)
18. Dokhanchi, A., Hoxha, B., Fainekos, G.: Formal requirement debugging for testing and verification of cyber-physical systems. ACM Transactions on Embedded Computing Systems (TECS) **17**(2), 34 (2018)
19. Dokhanchi, A., Yaghoubi, S., Hoxha, B., Fainekos, G., Ernst, G., Zhang, Z., Arcaini, P., Hasuo, I., Sedwards, S.: Arch-comp18 category report: Results on the falsification benchmarks. In: ARCH@ ADHS. pp. 104–109 (2018)
20. Dokhanchi, A., Zutshi, A., Sriniva, R.T., Sankaranarayanan, S., Fainekos, G.: Requirements driven falsification with coverage metrics. In: Proceedings of the 12th International Conference on Embedded Software. pp. 31–40. IEEE Press (2015)
21. Donze, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Computer Aided Verification, LNCS, vol. 6174, pp. 167–170 (2010)
22. Donze, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Formal Modeling and Analysis of Timed Systems, LNCS, vol. 6246, pp. 92–106. Springer (2010)
23. Dreossi, T., Ghosh, S., Sangiovanni-Vincentelli, A., Seshia, S.A.: Systematic testing of convolutional neural networks for autonomous driving (2017), reliable Machine Learning in the Wild (RMLW) workshop
24. Ernst, G., Arcaini, P., Donze, A., Fainekos, G., Mathesen, L., Pedrielli, G., Yaghoubi, S., Yamagata, Y., Zhang, Z.: Arch-comp 2019 category report: Falsification. EPiC Series in Computing **61**, 129–140 (2019)
25. Fainekos, G., Sankaranarayanan, S., Ueda, K., Yazarel, H.: Verification of automotive control applications using s-taliro. In: Proceedings of the American Control Conference (2012)
26. Fainekos, G.E., Girard, A., Kress-Gazit, H., Pappas, G.J.: Temporal logic motion planning for dynamic robots. Automatica **45**(2), 343–352 (2009)
27. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications. In: Formal Approaches to Testing and Runtime Verification. LNCS, vol. 4262, pp. 178–192. Springer (2006)
28. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theoretical Computer Science **410**(42), 4262–4291 (2009)
29. Ferrère, T., Nickovic, D., Donzé, A., Ito, H., Kapinski, J.: Interface-aware signal temporal logic. In: 22nd ACM International Conference on Hybrid Systems: Computation and Control. pp. 57–66 (2019)
30. Fremont, D.J., Dreossi, T., Ghosh, S., Yue, X., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Scenic: a language for scenario specification and scene generation. In: PLDI. pp. 63–78 (2019)
31. Gregg, A., MacMillan, D.: Airlines cancel thousands of flights as boeing works to fix 737 max software problems. The Washington Post **July 14** (2019)
32. Hoxha, B., Abbas, H., Fainekos, G.: Benchmarks for temporal logic requirements for automotive systems. In: Workshop on Applied Verification for Continuous and Hybrid Systems (2014)

33. Hoxha, B., Dokhanchi, A., Fainekos, G.: Mining parametric temporal logic properties in model based design for cyber-physical systems. International Journal on Software Tools for Technology Transfer **20**, 79–93 (2018)
34. Hoxha, B., Mavridis, N., Fainekos, G.: Vispec : A graphical tool for elicitation of mtl requirements. In: IEEE/RSJ IROS (2015)
35. Johnson, T.T., Gannamaraju, R., Fischmeister, S.: A survey of electrical and electronic (e/e) notifications for motor vehicles. In: ESV'15 (2015)
36. Kapinski, J., Deshmukh, J.V., Jin, X., Ito, H., Butts, K.: Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques. IEEE Control Systems **36**(6), 45–64 (2016)
37. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems **2**(4), 255–299 (1990)
38. LeCun, Y., Kavukcuoglu, K., Farabet, C.: Convolutional networks and applications in vision. In: Proceedings of 2010 IEEE International Symposium on Circuits and Systems. pp. 253–256 (May 2010)
39. Lee, T.B.: Report: Software bug led to death in ubers self-driving crash. Ars Technica **May 07** (2018)
40. Leitner, F., Leue, S.: Simulink Design Verifier vs. SPIN - a comparative case study (short paper). In: Formal Methods for Industrial Critical Systems (2008)
41. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Proceedings of FORMATS-FTRTFT. LNCS, vol. 3253, pp. 152–166 (2004)
42. Mathesen, L., Yaghoubi, S., Pedrielli, G., Fainekos, G.: Falsification of cyber-physical systems with robustness uncertainty quantification through stochastic optimization with adaptive restart. In: IEEE CASE (2019)
43. Nghiem, T., Sankaranarayanan, S., Fainekos, G.E., Ivancic, F., Gupta, A., Pappas, G.J.: Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In: Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control. pp. 211–220. ACM Press (2010)
44. S-TaLiRo Tools: `https://sites.google.com/a/asu.edu/s-taliro/`
45. Sandler, K., et al.: Killed by code: Software transparency in implantable medical devices. Tech. rep., Software Freedom Law Center (2010)
46. Tuncali, C.E., Fainekos, G., Ito, H., Kapinski, J.: Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In: IEEE Intelligent Vehicles Symposium (IV) (2018)
47. Tuncali, C.E., Hoxha, B., Ding, G., Fainekos, G., Sankaranarayanan, S.: Experience report: Application of falsification methods on the UxAS system. In: 10th NASA Formal Methods (NFM). LNCS, vol. 10811. Springer (2018)
48. Yaghoubi, S., Fainekos, G.: Gray-box adversarial testing for control systems with machine learning components. In: ACM International Conference on Hybrid Systems: Computation and Control (HSCC) (2019)
49. Yaghoubi, S., Fainekos, G.: Worst-case satisfaction of stl specifications using feedforward neural network controllers: A lagrange multipliers approach. In: International Conference on Embedded Software (EMSOFT) (2019)
50. Zhang, Z., Ernst, G., Sedwards, S., Arcaini, P., Hasuo, I.: Two-layered falsification of hybrid systems guided by monte carlo tree search. IEEE Trans. on CAD of Integrated Circuits and Systems **37**(11), 2894–2905 (2018)