Analyzing Neighborhoods of Falsifying Traces in Cyber-Physical Systems

Ram Das Diwakaran
University of Colorado Boulder
ram.diwakaran@colorado.edu

Sriram Sankaranarayanan
University of Colorado Boulder
srirams@colorado.edu

Ashutosh Trivedi
University of Colorado Boulder
ashutosh.trivedi@colorado.edu

ABSTRACT

We study the problem of analyzing falsifying traces of cyber-physical systems. Specifically, given a system model and an input which is a counterexample to a property of interest, we wish to understand which parts of the inputs are "responsible" for the counterexample as a whole. Whereas this problem is well known to be hard to solve precisely, we provide an approach based on learning from repeated simulations of the system under test.

Our approach generalizes the classic concept of "one-at-a-time" sensitivity analysis used in the risk and decision analysis community to understand how inputs to a system influence a property in question. Specifically, we pose the problem as one of finding a neighborhood of inputs that contains the falsifying counterexample in question, such that each point in this neighborhood corresponds to a falsifying input with a high probability. We use ideas from statistical hypothesis testing to infer and validate such neighborhoods from repeated simulations of the system under test. This approach not only helps to understand the sensitivity of these counterexamples to various parts of the inputs, but also generalizes or widens the given counterexample by returning a neighborhood of counterexamples around it.

We demonstrate our approach on a series of increasingly complex examples from automotive and closed loop medical device domains. We also compare our approach against related techniques based on regression and machine learning.

1. INTRODUCTION

We present a simulation-based approach to analyze a given set of counterexamples to temporal logic properties of cyber-physical systems (CPS) towards understanding the root causes that underlie the given counterexamples. The problem of finding counterexamples to system specification is known as *falsification*. There are many approaches to falsification, including symbolic approaches using model checking and simulation-based approaches that treat the system as a black-box and attempt to find inputs that lead to a counterexample. Simulation-based approaches have been well studied in the recent past [27, 5, 3, 15]. This research has led to tools such as S-Taliro [6] and Breach [14], and has been demonstrated through case studies in widely varying domains such as automotive systems [18] and medical devices [9, 33, 34].

Arguably, a successful discovery of a counterexample to a property is often a *starting point* rather than the end goal of the analysis process. For instance, in many applications it may be important to understand the root causes of a falsification in terms of the inputs to the system. This process is often manually implemented using trial and error. However, for large systems with many input parameters and signals, a manual analysis is often time consuming, requiring expertise in the problem domain as well as an understanding of the underlying verification techniques.

Similarly, in other applications a counterexample may only be deemed a possible bug if it is preserved under suitably small perturbations. Otherwise, they are potentially classified as "low likelihood" or even artifacts of the modeling process. Thus, techniques to explore the neighborhood of the counterexample are very important. A key objective of the research presented in this paper is to automate such processes.

Understanding the sensitivity of the output with respect to small variations in the input is typically studied under the general term of *sensitivity analysis* [32] across multiple disciplines including scientific and mathematical modeling, operations research, and risk and decision analysis. There are two variants of sensitivity analysis—local and global sensitivity analysis. Given a complex deterministic system $y = \eta(\mathbf{x})$ and a baseline estimate $\mathbf{x}_0$, local sensitivity analysis is concerned with understanding how changes in the input $\mathbf{x}$ around $\mathbf{x}_0$ affect the output of the system. Local sensitivity analysis is based on derivatives of $\eta(\cdot)$ evaluated at $\mathbf{x} = \mathbf{x}_0$ and measures how the output changes when different variables (dimensions of $\mathbf{x}$) are individually varied across their range while keeping other variables constant to their baseline values. The results of local sensitivity analysis are often presented using sensitivity graphs, tornado diagrams, or spider diagrams [16]. These diagrams offer a visual cue to understand relative relevance of the variables (dimensions) with respect to the output. However, the local sensitivity approach is no longer appropriate [31] when we wish to understand the joint influence of multiple inputs perturbed at the same time.

Global sensitivity analysis [31], on the other hand, involve the use of machine learning to infer classifiers that explains output as function of inputs by simulating points in the neighborhood of the given nominal value. However, these approaches either yield *opaque models* that are hard to interpret for the purposes of root-cause understanding, or provide poor accuracy in some cases, leading to incorrect results.

The approach presented in this paper extends sensitivity analysis to generalize neighborhood of counterexamples by computing a box neighborhood around the original counterexample with a guarantee that if a point were to be chosen at random from this neighborhood according to a fixed sampling distribution, that point would also yield a counterexample with a high probability threshold that is a parameter to our search procedure. We call such a box a *falsifying neighborhood* of the given counterexample. The approach relies on simulations, and effectively treats the system as a black-box with assumptions that guarantee the existence of a neighborhood in the first place. We employ Bayesian statistical hypothesis testing as the basis for checking if a given box is indeed a falsifying neighborhood.

We describe the algorithms for computing falsifying neighborhoods from a given seed counterexample. We demonstrate these algorithms on a few nontrivial case studies drawn from the literature. Wherever possible, we compare our approach with other related approaches such as local sensitivity analysis using sensitivity graphs and tornado diagrams, and decision tree classification.

2. A MOTIVATING EXAMPLE

Let us consider a Simulink$^{®}$/Stateflow$^{\text{TM}}$ model of an automatic transmission controller, originally proposed by Zhao et al [41], to motivate our approach. The model contains nonlinearities and hybrid switching behavior in the form of discontinuous blocks and a stateflow diagram. Figure 1 shows the top level Simulink model. The model has one continuous input signal $u(t)$ that lies inside the range $[0, 100]$ for all time t, representing the user's throttle input through the accelerator pedal and two outputs that represent the vehicle speed $v(t)$ and the engine RPM $r(t)$. The total simulation time is $T = 30$ seconds. We parameterize the input signal $u(t)$ as a *piecewise constant* signal represented by a vector of control points $(u_1, u_2, \ldots, u_7)$, wherein

$$u(t) = \begin{cases} u_1 & 0 \le t < 5 \\ u_2 & 5 \le t < 10 \\ \vdots & \\ u_6 & 25 \le t < 30 \\ u_7 & t \ge 30 \end{cases}$$

We are interested in finding if the system always satisfies the temporal property ϕ_1: *the speed should always remain below 120 kmph or the engine speed should remain below 4500 RPM*. To violate this property, we search for a piecewise constant input signal $u(t)$ that causes both the speed and the engine RPM to exceed their prescribed limits. The S-Taliro tool produces the counterexample described by the control points:

$$(u_1, u_2, \ldots, u_7) = (56.7, 96.6, 91.9, 99.9, 98.6, 95.7, 69.6)\,.$$

Figure 2 shows the input signal corresponding to this counterexample and the corresponding values of speed and RPM showing the property violation.

As such, it is not clear how the values of the various control points affect the overall counterexample. To do so, we perform local sensitivity analysis by considering each input in isolation while fixing the remaining inputs and present the results as a tornado diagram that shows how far each input can be varied while still obtaining counterexamples. Figure 3(a) shows the resulting ranges for each of the 7 control points. At the same time, these ranges assume that only one input can vary at a time. However, in this example, they suggest that input u_5 needs to be varied inside the interval $[20, 100]$ whereas other inputs can be set arbitrarily in their range. This can be misleading since only one input is allowed to vary.

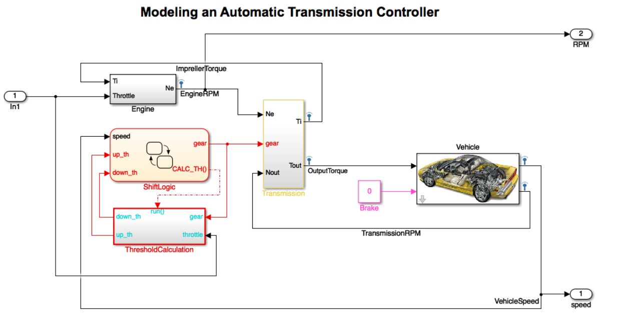


Figure 1: The automatic transmission system Simulink$^{®}$/Stateflow$^{\text{TM}}$ model.

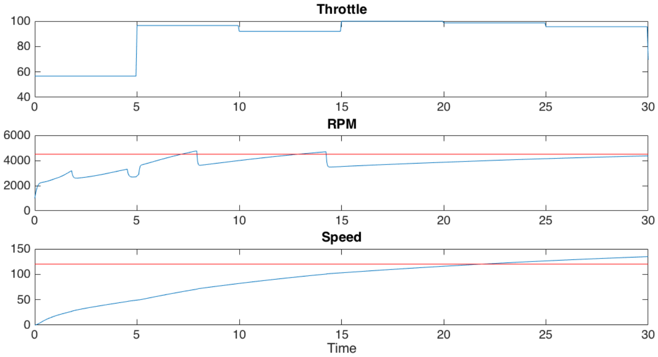


Figure 2: Counter-example for the ϕ_1 found by S-Taliro tool.

Figure 3(b) shows the actual box neighborhood wherein we claim that sampling any point uniformly at random in the neighborhood has at least a 99% chance of yielding a violation through our hypothesis testing procedure. Such a neighborhood captures what happens when all the inputs are potentially varied simultaneously. It shows that inputs u_2, u_3, u_5 and u_6 have to be restricted to a narrow range around the actual counterexample, while the other inputs can be allowed to vary inside larger ranges. In particular, the values of u_4 and u_7 do not matter as long as the other inputs are within their required ranges. Also, since u_7 affects the value of the signal past the simulation horizon, it is not surprising that u_7 has little effect on the falsification. However, it is clear that the values of u_1, u_2, u_3 are responsible initially for exceeding the RPM limit, with a larger range of possible values of u_1 but with u_2, u_3 fixed around a very narrow range. Later in the simulations, the values of u_5 and u_6 ensure that the speed limit is exceeded.

Finally, we sample the neighborhood computed using our method

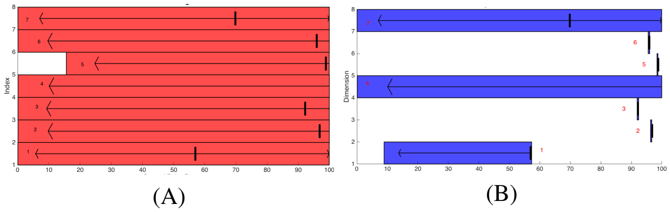
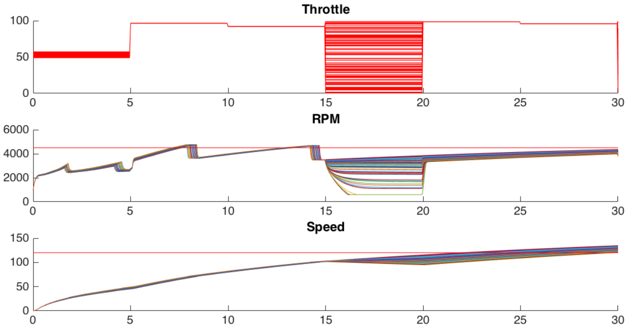


Figure 3: (A) The tornado diagram showing the range for each input control point in isolation, and (B) The falsification neighborhood. The x-axis represents the range of control points $[0, 100]$ and the y-axis represents each index from 1 to 7.



;

Figure 4: Simulations of the sample points drawn from the falsifying neighborhood

to simulate each sample. Figure 4 shows the resulting violations, that are qualitatively similar to the original violation shown in Fig. 2.

3. PRELIMINARIES

In this section, we present preliminary concepts required in the remainder of the paper. We begin by stating our basic assumptions on systems, the properties of interest, and the process of discovering counterexample using *robustness-guided falsification* approach [3].

3.1 System Model and Property Specification

We assume that the systems are treated as black-boxes, i.e, they are equipped with simulation functions that allow us to simulate them for a given set of initial conditions and signals. Wherever needed, we will assume that the system satisfies some conditions that allow us to reason about the properties of its counterexamples.

DEFINITION 1 (SYSTEM MODEL). *A system Π is given by a tuple $\langle X, \textsc{sim}, X_0, D, T\rangle$ that consists of:*

1. *a state-space $X \subseteq \mathbb{R}^n$, where n is the dimensionality of the state-space over which the system evolves;*
2. *a set $X_0 \subseteq X$ representing the initial conditions (including the fixed model parameters) of the system;*
3. *an interval D for the input signals to the system characterizing the set U of input signals defined by considering all functions $\mathbf{u} : [0, T] \to D$;*
4. *a finite time horizon $T > 0$; and*
5. *a* forward *simulator* $\textsc{sim} : X_0 \times U \times [0, T] \mapsto X$ *s.t.* $\mathbf{x}(t) : \textsc{sim}(\mathbf{x}_0, \mathbf{u}, t)$ *represents the state reached at time t given initial conditions $\mathbf{x}_0$, input $\mathbf{u}$ and time $t \geq 0$.*

Given initial condition $\mathbf{x}_0$, and input signals $\mathbf{u}(t)$, the *trace* of a system Π is given by the function $\mathbf{x} : [0, T] \to X$, wherein $\mathbf{x}(t) : \textsc{sim}(\mathbf{x}_0, \mathbf{u}, t)$.

ASSUMPTION 1. *For the sake of simplicity we make the following assumptions.*

1. *The initial conditions are fixed to a single point, i.e. $X_0 = \{\mathbf{x}_0\}$. However, the approach in this paper will extend naturally to sets of initial conditions, as well.*
2. *We do not separately define the states and the outputs of the systems, assuming that all states are observable.*
3. *We also assume a finite time horizon T for our investigations, since we will be restricted to simulations that can only be computed for finite times, in general.*

We assume that the set of properties of the system are specified in formal temporal specification logic such as *metric temporal logic* (MTL) or *signal temporal logic* (STL) [23, 15]. These temporal logics allow one to express rich temporal properties of the system including real-time properties. For instance, the property ϕ from our motivating example in Section 2 can be written as :

$$\Box(\text{rpm}(t) \leq 4500) \ \wedge\ \Box(v(t) \leq 120).$$

Given a formula ϕ, we define the robustness of a trace $\mathbf{x}$ with respect to φ as a real-valued quantity denoted $r : \mathcal{R}(\mathbf{x}, \phi)$. The robustness provides a measure of satisfaction of the trace with respect to the property and is such that:

1. positive values of robustness indicate that the trace satisfies the property (if $r > 0$ then $\mathbf{x} \models \phi$);
2. negative values of robustness indicate that the trace satisfies the negation of the property (if $r < 0$ then $\mathbf{x} \models \neg\phi$); and
3. the robustness provides a notion of distance between the trace and the property: every trace $\mathbf{y}$ such that $||\mathbf{y}(t) - \mathbf{x}(t)||_2 \leq r$ also has the same outcome for the property as $\mathbf{x}$.

We refer the reader to work by Fainekos & Pappas [17] and Donze & Maler [15] for a formal definition of robustness for MTL and STL properties, respectively. It is important to note that there are tools, such as TaLiRo and Breach, to efficiently compute robustness of a trace wrt MTL formula.

Finally, the techniques in this paper depend on taking infinite dimensional continuous time signals $\mathbf{u}(t)$ and parameterizing them to a finite dimensional vector of control points. For this purpose we introduce signal parameterization.

DEFINITION 2 (SIGNAL PARAMETERIZATION). *A parameterization $(V, \pi) \in \mathbb{R}^m \times [V \to U]$ of signals in U represents a signal $\mathbf{u}(t) \in U$ by a vector $\mathbf{v} \in V$ of* control points *such that $\mathbf{u} = \pi(\mathbf{v})$.*

Common parameterizations are obtained by choosing time points $t_0 : 0 \leq t_1 < t_2 < \cdots < t_N \leq t_{N+1} : T$. We describe parameterizations for a single scalar-valued signal $u(t)$ below. Vector-valued extensions are also defined similarly.

Piecewise-Constant. For piecewise-constant parameterizations the signal corresponding to control point $\mathbf{v} : (v_1, \ldots, v_{N+1})$ is given by $u(t) = v_i$ whenever $t_{i-1} \leq t < t_i$ for $1 \leq i \leq N + 1$. This is the parameterization used in the motivating example in section 2.

Piecewise-Linear. For piecewise-linear parameterizations the signal corresponding to control point $\mathbf{v} : (v_1, \ldots, v_{N+1})$ is given by $u(t) = v_{i-1} + (v_i - v_{i-1})((t - t_{i-1})/(t_i - t_{i-1}))$ whenever $t_{i-1} \leq t < t_i$ for $1 \leq i \leq N + 1$.

Bezier Splines. These are piecewise polynomial curves that are constructed using a finite set of control points. For a precise definition and background on Bezier splines we refer the reader to [30].

Finally, we recall the robustness-guided approach to finding counterexamples of properties, originally proposed by Nghiem et al [27, 3]. The approach inputs a model Π and a property ϕ. It then fixes a finite parameterization π of the inputs to the model. The search is cast as the following problem of solving constraints over the decision variables $\mathbf{v}$ for the control points such that the resulting trace $\mathbf{x}$ has a negative value of robustness:

$$\begin{aligned}\text{find } \mathbf{v} \in V \text{ such that} \quad & \mathcal{R}(\mathbf{x}, \phi) < 0, \\ & \mathbf{u}(t) = \pi(\mathbf{v}),\ t \in [0, T] \\ & \mathbf{x}_0 \in X_0, \text{ and} \\ & \mathbf{x}(t) = \textsc{sim}(\mathbf{x}_0, \mathbf{u}, t),\ t \in [0, T].\end{aligned}$$

In turn, the robustness is treated as an objective function and minimized using stochastic optimization techniques. Even though approach lacks the formal guarantees obtained using other symbolic model checking techniques, it is applicable at once to a wide variety of large non-linear models wherein simulations are much cheaper.

4. FALSIFYING NEIGHBORHOODS

In this section, we define the notion of a falsifying neighborhood for a given counterexample. In the subsequent section we describe our approach to infer such neighborhoods.

Let $\Pi : \langle X, X_0, D, \textsc{sim} \rangle$ be a system model under investigation, (V, π) be an N-dimensional parameterization of the input signals U, and let ϕ be an MTL property of interest.

Given $\mathbf{v} \in V$, we define its associated robustness $\rho(\mathbf{v}, \phi)$ wrt property ϕ to be equal to the robustness of the trace $\mathbf{x}(t)$ obtained by simulating the system using the control signal derived from $\mathbf{v}$, i.e. $\rho(\mathbf{v}, \phi) = \mathcal{R}(\mathbf{x}(t), \phi)$, where $\mathbf{x}(t) = \textsc{sim}(\mathbf{x}_0, \pi(\mathbf{v}), t)$.

Let $\mathbf{v}_0 : (v_1, \ldots, v_N) \in V$ be a counterexample to ϕ obtained through a falsification tool. Thus, $\rho(\mathbf{v}_0, \phi) < 0$. We will call $\mathbf{v}_0$ the *seed counterexample*. Ideally, our goal is to construct an interval $I(\mathbf{v}_0)$ that contains $\mathbf{v}_0$ such that any control point in I yields a falsification.

ASSUMPTION 2. *The robustness function $\rho(\mathbf{v}, \phi)$ is locally continuous in some nonempty neighborhood N of $\mathbf{v}_0$.*

LEMMA 1. *If the robustness $\rho(\mathbf{v}, \phi)$ is locally continuous around $\mathbf{v}_0$, then there exists a nonempty falsifying neighborhood F such that for all $\mathbf{v} \in F$, $\rho(\mathbf{v}, \phi) < 0$.*

The assumption of local continuity can be proven in many situations including for nonlinear differential equations and switched systems under time-triggered switching [1, 2].

4.1 Tornado Diagrams

A simple first approach employs sensitivity analysis to investigate how much each entry v_i of $\mathbf{v}_0$ can be varied while still obtaining a falsification. Let $[l_i, u_i]$ represent the absolute limits of the control point v_i in the set V. We characterize this information using tornado diagrams.

DEFINITION 3 (TORNADO DIAGRAM). *A* tornado diagram *around $\mathbf{v}_0$ is the largest interval $T(\mathbf{v}_0) : ([x_1, y_1], \ldots, [x_N, y_N])$ such that for each $i \in [1, N]$, we have that*

(a) $l_i \leq x_i \leq v_i \leq y_i \leq u_i$, *and*

(b) $\rho((v_1, \ldots, v_{i-1}, \hat{v_i}, v_{i+1}, \ldots, v_N), \phi) < 0$ *for any* $\hat{v_i} \in [x_i, y_i]$,

In other words, changing a single entry v_i in $\mathbf{v}_0$ to a new value in the interval $[x_i, y_i]$ also yields a falsification.

Tornado diagrams can be computed approximately through simulations by increasing the value of v_i along each dimension while keeping the others constant until a violation is no longer obtained. However, as mentioned before, the results can be misleading, since only a single dimension is considered at a time.

4.2 Falsification Intervals

We now define the notion of a falsification neighborhood given a seed counterexample $\mathbf{v}_0$.

DEFINITION 4 (FALSIFICATION INTERVAL). *An interval $I : [a_1, b_1] \times \cdots \times [a_N, b_N] \subseteq V$ is said to be a* falsification interval *around a seed counterexample $\mathbf{v}_0$ wrt a property ϕ if $\mathbf{v}_0 \in I$ and furthermore, every control point $\mathbf{v} \in I$ is also a counterexample to the property ϕ, i.e, $\rho(\mathbf{v}, \phi) < 0$.*

Whereas the definition is straightforward, it is noteworthy that it is hard (if not impossible) to formally verify whether a claimed interval I is indeed falsifying unless we resort to symbolic reasoning over the system trajectories. For this reason we rely on statistical evidence through simulations.

In order to provide such statistical evidence, we associate a probability $\mathcal{D}$ with the set of control points V. Such probabilities arise naturally in physical systems wherein the distribution that governs the inputs to the systems can be naturally associated with the control points. Failing this assumption, we may still associate distributions to represent a notion of relative weights of different control points in V. Since V is compact and the distribution $\mathcal{D}$ is chosen to be the uniform distribution in such a situation.

EXAMPLE 1. *For the motivating example of the automatic transmission system, we note that the control points correspond to values of the throttle input which are generated by the driver. In principle, it is possible to collect data from drivers to provide a distribution. At the same time, these inputs are hardly likely to be piecewise constant, and will exhibit temporal correlations. Therefore, we choose each control point to be uniformly distributed in the range $[0, 100]$, noting that the distribution is now a weighting function for the control point values in the interval.*

DEFINITION 5 (LIKELY FALSIFICATION INTERVAL). *Given a distribution $\mathcal{D}$ over V, we say that an interval I is a probabilistic falsification interval with threshold c if and only if*

$$\mathsf{Pr}(\rho(\mathbf{v}, \phi) < 0 \mid \mathbf{v} \in I) \geq c.$$

We assume that $\mathbf{v}$ is sampled according to the distribution $\mathcal{D}$. Also, c can be set to some high probability threshold such as 0.99.

Finally, we use statistical hypothesis testing to test if a given interval I is indeed a falsifying interval or not, using data gathered from simulating K samples chosen at random from I according to the distribution $\mathcal{D}$ (formally, we condition the probability on I and sample accordingly). The statistical criterion we employ in this paper is simple:

1. Draw a fixed number K (to be decided later) samples from I.
2. If all K samples are counterexamples, then we accept I as a likely falsification interval.
3. Otherwise, we reject I as a likely falsification interval.

Our approach will be based on the standard Bayesian hypothesis testing framework, using the Jeffries Bayes factor test [22, 20, 42].

Let I be a given interval and $p(I)$ be an unknown probability that a randomly drawn sample from I is a counterexample. We consider two competing hypotheses:

$$\mathcal{H}_0 : \ p(I) < c \text{ versus } \mathcal{H}_1 : \ p(I) \geq c\,.$$

Algorithm 1: Algorithm to check if a given interval is likely falsifying using the Bayes factor test: returns ACCEPT if the test passes, or REJECT with a control point $\mathbf{v} \in V$.

```
Input: Interval $I$, System simulator SIM, control input space
       $V$ and distribution $D$, probability $c$ and Bayes factor $B$.
Result: ACCEPT or $(\text{REJECT}, \mathbf{v})$
$K := \frac{\log(B)}{\log(c)}$ ;
for $i \in [1, K]$ do
    $\mathbf{v}_i := \mathsf{Sample}(V, \mathcal{D})$ ;
    $\mathbf{x}(t) := \text{SIM}(\pi(\mathbf{v}_i), T)$ ;
    if $\mathbf{x} \not\models \phi$ then
        return $(\text{REJECT}, \mathbf{v}_i)$;
return ACCEPT;
```

Our goal is to use data from repeated samples to decide between $\mathcal{H}_1$ vs. $\mathcal{H}_0$. We assume an uniform prior probability distribution over $p(I)$ reflecting our lack of information about what $p(I)$ should be. As a result, we associate a prior probability of c for $\mathcal{H}_0$ and a prior probability of $1-c$ for $\mathcal{H}_1$. Since $c \sim 0.99$, we note that our prior beliefs overwhelmingly favor $\mathcal{H}_0$ over $\mathcal{H}_1$.

Let $\mathbf{v}_1, \ldots, \mathbf{v}_K$ be some K samples drawn from I such that all of them are counterexamples. The probability that this happens under the hypothesis $\mathcal{H}_0$ is given by

$$\mathsf{Pr}(\mathbf{v}_1, \ldots, \mathbf{v}_K \text{ counterex.} |\, \mathcal{H}_0) = \int_0^c p^K dp \,.$$

Likewise, the probability that this happens under $\mathcal{H}_1$ is

$$\mathsf{Pr}(\mathbf{v}_1, \ldots, \mathbf{v}_K \text{ counterex.} |\, \mathcal{H}_1) = \int_c^1 p^K dp \,.$$

The ratio of these probabilities is called the Bayes factor.

$$\frac{\mathsf{Pr}(\mathbf{v}_0, \ldots, \mathbf{v}_K \text{ counterex.} |\mathcal{H}_1)}{\mathsf{Pr}(\mathbf{v}_0, \ldots, \mathbf{v}_K \text{ counterex.} |\, \mathcal{H}_0)} = \frac{1 - c^{K+1}}{c^{K+1}}$$

Bayes factor measures how the data transforms the prior odds of $\mathcal{H}_1$ against $\mathcal{H}_0$ to yield a posterior odds. It is therefore used as a measure of the strength of the evidence in favor of $\mathcal{H}_1$ and against $\mathcal{H}_0$. Typically, a bayes factor greater than some fixed number B will be used to accept $\mathcal{H}_1$ over $\mathcal{H}_0$.

We can now calculate the value of K, the number of simulations needed for a given c and Bayes factor threshold of B, so that $\frac{1-c^{K+1}}{c^{K+1}} > B$. Simplifying yields, $K > -\frac{\log(B)}{\log(c)}$. For example, if we set $c = 0.99$ and $B = 100$, we require $K > 460$ simulations. A higher confidence level of $B > 1000$ and $c = 0.99$ requires $K > 700$. Typically, $B > 100$ is considered "decisive" in the literature [22].

Formally the likelihood that we falsely accept $\mathcal{H}_1$ when $\mathcal{H}_0$ is really the truth is given by the formula $\frac{1}{\gamma B + 1}$, where $\gamma = \frac{1-c}{c}$. For $B = 100$, and $c = 0.99$, this is nearly $\frac{1}{2}$. Thus, we shoot for a much higher value of $B \sim 10^5$ for $c = 0.99$ in our experiments.

Thus far, we have defined *likely falsification intervals* and provided a statistical procedure to test whether a given interval is indeed a likely falsification interval, with a given probability threshold c. This procedure is described in Algorithm 1.

A *type-1* error occurs when Algorithm returns ACCEPT on an interval I that is in fact not a likely falsifying interval with threshold c (informally, the evidence fools us into accepting a false statement).

THEOREM 2. *The likelihood of a type-1 error in Algorithm 1 is at most* $\frac{c}{c+(1-c)B}$.

The bounds above use the well-known rates of type-1 error for Bayes factor-based hypothesis testing. For our experiments, we set $K = 1300$ (a large number) for $c = 0.99$. This gives us a Bayes factor $B = 4.15 \times 10^5$ and a probability of drawing a wrong conclusion of roughly 2.5×10^{-4}. We will thus use Algorithm 1 with $K = 1300$.

5. FINDING NEIGHBORHOODS

Thus far, we have defined the notion of a likely falsification interval and provided a simple procedure for checking a given likely falsification interval. In this section, we will focus on the inference of such intervals through simulations. Our procedure guarantees that if it finds an interval, the interval is guaranteed to be a likely falsification interval. The overall procedure involves three broad steps: 1) finding candidate intervals for the search, 2) testing the candidate interval and 3) shrinking a candidate interval if required.

5.1 Finding Candidate Intervals

We are given, as input a seed counterexample $\mathbf{v}_0$ and a domain $D : [l_1, u_1] \times \cdots \times [l_N, u_N]$. Our goal is to find likely intervals $I_1, \ldots, I_m$ that could be a candidate.

To do so, we generate samples $S : \{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ over the domain D and partition the samples into two sets. The set S_+ contains all samples that satisfy the property ϕ

$$S_+ : \{\mathbf{v}_i \in S \mid \rho(\mathbf{v}_i, \phi) > 0\} \,.$$

The remaining samples belong to the set S_-.

We enforce three requirements on the unknown candidate interval $I : [x_1, y_1] \times \cdots [x_N, y_N]$: (a) it must contain $\mathbf{v}_0$, the seed counterexample, (b) it must exclude all points in S_+ and (c) it must satisfy a *minimum* width criterion $(\epsilon_1, \ldots, \epsilon_N)$, specified by the user $(y_i - x_i) \geq \epsilon_i$.

However, there can be multiple falsifying intervals. Figure 5 shows how such intervals can arise naturally. In fact the number of possible intervals can be exponential in N, the dimensionality of the space V of control points. The problem of learning "good" intervals is closely related to the maximum empty rectangle problem that is recently shown to be NP-complete [7]. Given this intractability result, we propose an approach to pick finitely many such intervals using SMT (SATisfiability modulo theory) solvers with an objective function that optimizes the width of the box along a particular dimension. The extension of SMT solvers with optimization has received increasing attention recently [35] with widely used solvers such as Z3 supporting the maximization and minimization of objective functions [8].

SMT Encoding. We use variables x_i, y_i to denote the upper and lower bounds of the candidate interval along the dimension $i = 1, \ldots, N$. We first enforce that the bounds must be in the domain $D : [l_1, u_1] \times \cdots \times [l_N, u_N]$ and must contain the seed counterexample $\mathbf{v}_0 : (v_1, \ldots, v_N)$.

$$\psi_1 : \bigwedge_{i=1}^{N} \left[\begin{array}{l} (x_i \leq v_i) \,\wedge\, (v_i \leq y_i) \,\wedge \\ (\ell_i \leq x_i) \,\wedge\, (y_i \leq u_i) \end{array} \right] .$$

Next, we enforce a minimum width ϵ_i along each dimension. This is a parameter input by the user:

$$\psi_2 : \bigwedge_{i=1}^{N} (y_i - x_i) \geq \epsilon_i \,.$$

Each point $\mathbf{w} : (w_1, \ldots, w_N) \in S_+$, must be excluded. We first compare w_i with the seed counterexample v_i. If $w_i \geq v_i$ then the value of y_i needs to be adjusted or else the value of x_i needs to be

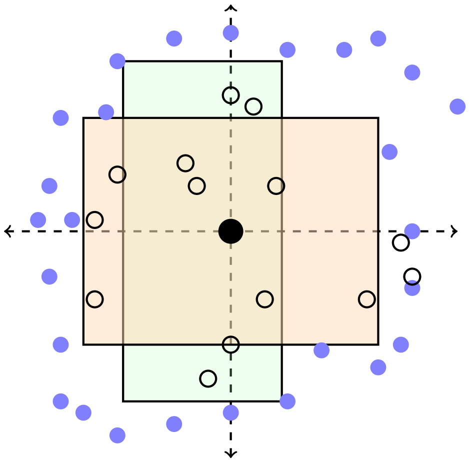


Figure 5: This figure illustrates the construction of candidate intervals from simulation data. Here, the blue filled points represent control points satisfying ϕ whereas the black circles represent counterexamples. The seed counterexample is shown as the black circle at the center.

adjusted. We also use a parameter λ to place a "gap" between the interval boundaries $[x_i, y_i]$ and the excluded counterexample:

$$\psi_3(\mathbf{w}) : \bigwedge_{i=1}^{N} \left(\begin{array}{l} w_i \geq v_i \ \Rightarrow\ y_i \leq \lambda w_i + (1-\lambda)v_i \ \wedge \\ w_i \leq v_i \ \Rightarrow\ x_i \geq \lambda w_i + (1-\lambda)v_i \end{array} \right) .$$

The overall SMT encoding combines the formulas as follows:

$$\Psi : \ \psi_1 \ \wedge \ \psi_2 \ \wedge \ \bigwedge_{\mathbf{w} \in S_+} \ \psi_3(\mathbf{w}) \,.$$

The following theorem follows from the SMT formulation in a straightforward manner.

THEOREM 3. *Any falsifying interval J that satisfies the minimum width constraints also satisfies the formula Ψ.*

However, as mentioned earlier, the formula Ψ can have too many solutions. It is not feasible to explore them all. As a result, we use objective functions to help us select some solutions from this space.
Maximize Width: One simple class of objective considers the width $(y_i - x_i)$ for each $i \in [1, \ldots, N]$. Doing so along each dimension yields us a set of N candidates.
Maximum Number of Samples: Another approach considers maximizing the number of samples in S_- set that the interval contains. This is enforced in SMT solvers by adding "soft constraints" with weights, which are then maximized by the solver while searching for a solution.

5.2 Testing and Refining Candidates

Once we have a candidate interval I, we can apply the statistical testing described in Algorithm 1. If the candidate passes the statistical testing procedure we can output the interval. However, if the candidate fails the statistical test procedure, we have to arrive at a new candidate. We now discuss the refinement process.

The refinement process for an interval I is triggered by a new sample $\mathbf{w} \in I$ such that $\rho(\mathbf{w}, \phi) \geq 0$. We wish to refine our interval I to exclude $\mathbf{w}$. This can be achived in one of the following two ways.

1. **Back to "Drawing Board":** This approach simply augments the input data to the SMT formula from the previous section and recomputes candidates from scratch, now considering one or more new counterexamples that cause our tests to fail. We then repeat the tests on the new candidates that arise from this process. However, this process is expensive since it requires repeated solving of an SMT problem each time our testing procedure fails.

2. **Greedy Refinement:** A simpler, heuristic approach is to refine the candidate interval I to exclude the counterexample $\mathbf{w} : (w_1, \ldots, w_N)$ returned by Algo. 1. This is done by (a) choosing a dimension i and (b) reducing the width of the interval along dimension I to exclude $\mathbf{w}$. The resulting interval $\hat{I}$ is now tested from scratch using Algo. 1. In particular, the re-testing cannot use any previously seen samples to avoid biasing our test procedure. The choice of which dimension to shrink can affect the interval we obtain. One strategy is to use the dimension that shrinks the least to exclude $\mathbf{w}$. Another strategy is to fix a *priority order* amongst the dimensions up front and choose the first dimension whose current width is larger than the minimum permitted width.

6. EVALUATION

In this section, we describe the experimental evaluation of our approach over a series of benchmark systems drawn from our previous work. Each evaluation run uses the following scheme.

1. We run the S-Taliro tool to collect a set of up to 5 *seed* counterexamples.

2. We sample control points around this seed and record the resulting robustness.

3. The samples are then used to collect upto N interval candidates using the optimization modulo theory solver implemented inside the Z3 SMT solver.

4. The interval candidates are tested using Algo. 1 and refined heuristically until we accept the interval.

The testing procedure is run with $K = 1300$ and $c = 0.99$, guaranteeing an appropriately small type-1 error likelihood $\sim 2.5 \times 10^{-4}$.

6.1 Quadrotor System

We first examine the quadrotor model of the robotics toolbox [12], and also available as a demo example for the S-Taliro tool [6]. The model allows the specification of two types of regions: a "good" region that is to be visited by a quadrotor aircraft and a "bad region" to be avoided.

The system consists of a detailed model of a nonlinear quadrotor aircraft which has two inputs representing the commanded x, y position of the quadrotor. Internally, it uses nested PI controllers to stabilize the pitch and yaw to reach the desired commanded position. We use S-Taliro tool to command a sequence of x and y positions so that the trajectories exhibit the following properties.

Property-1: The quadrotor should visit the good region, while avoiding the bad region.

Property-2: The quadrotor should eventually enter and stay forever inside the good region while avoiding the bad region during the time interval $[4, 5]$.

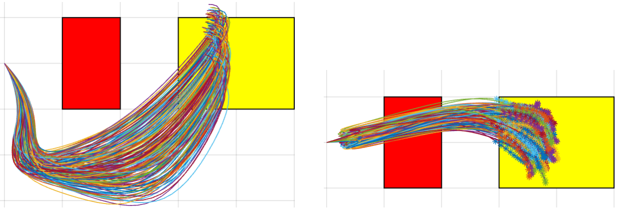


Figure 6: Quadrotor sample trajectories from the likely falsifying neighborhoods for Property-1 (left) and Property-2 (right). In each figure the x and y axis represents the position of the quadrotor in the configuration space.

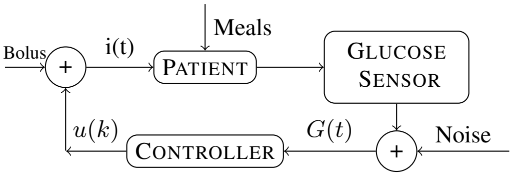


Figure 7: Closed loop diagram for the artificial pancreas control scheme taken from Cameron et al [10].

Note, that in this instance, the tool S-Taliro is not used to falsify a requirement but rather to generate a control input to navigate the robot towards a specified goal. The input signals are piecewise constant over the time horizon with 5 pieces, giving us a total of 10 control points. We generated 3 seed control points for each property. Our approach computes intervals around the seed control points that also constitute valid control inputs to achieve Property-1 and Property-2. Figure 6 shows the sampled trajectories from the intervals discovered by our algorithms.

6.2 Artificial Pancreas System

This evaluation extends our previous work on falsifying MTL properties for an artificial pancreas controller that controls the external insulin infusion in a patient with type-1 diabetes [10]. The controller used in this example is directly inspired by the PID control scheme proposed by Steil et al. [38, 37, 39]. A detailed analysis of this control scheme was presented by Palerm [29]. In conjunction with this controller, we use the Dalla-Man et al model to capture the insulin-glucose regulation in the human patient [13, 26].

We focus on the analysis of counterexamples obtained in our original work. Table 1 taken from Cameron et al. [10] describes the purpose of each dimension of the control points. There are 111 in all, with 8 values for parameters that describe the meals and insulin boluses of the patient and nearly 620 points that represent the sensor noise values. The property numbers used here are the same as in Cameron et al [10].

Property #2: This property expresses that the patient must not suffer a severe hyperglycemia defined as blood glucose levels above 350 `mg/dl`, two hours after the controller is switched on. Our approach generalizes the the seed counterexample discovered by S-Taliro. The SMT solver multiple candidates but only one was examined since all the candidate intervals were found to be quite close to each other. After refinement, our procedure discovered a likely falsifying interval defined by a narrow range around the seed counterexample for 4 out of the 111 parameters: these included $T_1 \in [17.51, 17.55]$, $X_1 \in [230, 231]$, $T_2 \in [180, 275]$, and $IC_1 \in [0.007, 0.008]$. The remaining 117 parameters can potentially take on any values within their permitted ranges. In particular, we infer that the property violation is completely insensitive to the sensor noise. Figure 8 shows the blood glucose levels for 50 sample traces from the final likely interval.

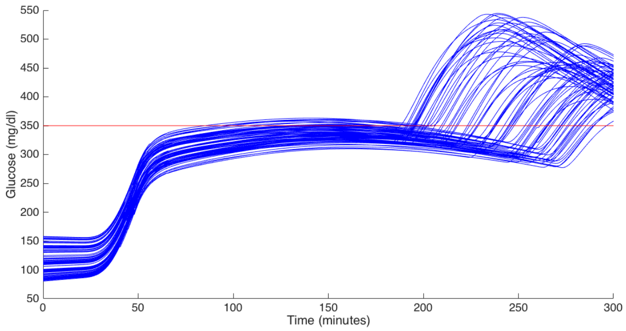


Figure 8: Simulations of samples from the likely falsifying interval for property # 2.

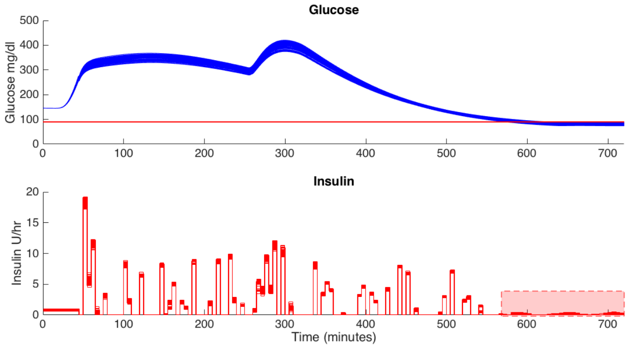


Figure 9: Simulations of samples from the likely falsifying interval for property #3, showing glucose (top) and insulin.

Property #3: This property states that the controller must not infuse insulin when the patient's blood glucose levels are below 90 `mg/dl`. Once again, we started from a seed violation found by S-Taliro. However, in this case, the resulting box represented a very narrow range around the counterexample. Figure 9 shows the resulting simulations including the blood glucose and insulin levels. Here, we note that the violations happen at the very end of the simulation near time $t = 720$. Small changes to the seed counterexample resulted in a behavior that did not violate the property of interest.

Property #6: This property states that the patient must not undergo a prolonged episode of hyperglycemia lasting more than 180 minutes, wherein the blood glucose levels are above 240`mg/dl`. For this property, we find a falsifying interval that simply restricts two out of the 120 dimensions, specifically the timing of meal # 1 is restricted to a narrow interval $[10, 12]$ minutes and the amount of carbohydrates is restricted to $[230, 250]$. All other dimensions can vary arbitrarily across the full range. The falsifying interval once again highlights that the control algorithm is perhaps not "aggressive" enough to treat high blood glucose levels due to the safety limitations that saturate the insulin infusion to a maximum value.

Table 2 summarizes the cumulative running times and the number of simulations for the various phases of our overall evaluation procedure. The running times were collected using a serial implementation of our algorithm implemented on a laptop with 2.8

Table 1: Inputs that are set by S-Taliro to falsify properties for the AP control system.

T_1	$[0, 60]$ mins.	Dinner time.
X_1	$[50, 150]$ gms	Amount of CHO in dinner.
T_2	$[180, 300]$ mins.	Snack time.
X_2	$[0, 40]$ gms	Amount of CHO in snack.
IC_1, IC_2	$[0, 0.01]$ U/gm	Insulin-to-CHO ratio for meal boluses.
δ_1, δ_2	$[-15, 15]$ min	timing of insulin relative to meal j
$d(100), d(105), \ldots, d(720)$	$[-20, 20]$ mg/dl	sensor error at each sample time.

Table 2: Running times and number of simulations for various phases of our evaluation. The machine learning instances all ran within 1 minute, their running times are not reported.

Prop.			Init. Simulations		SMT		Test/Refine		Machine Learning	
ID	# Cex	T	# Sims	Time	Time	# Cand.	Time	# Sims	Lin. Sep.	DTrees
Artificial Pancreas										
2	1	300	10,000	1h26	98s	1	25m	2892	98%	69 %
3	1	720	10,000	8h30m	364s	1	4h23m	2491	99%	82 %
4	1	380	10,000	1h50m	308s	1	14m30s	1316	89%	66%
Quadrotor										
1	3	5	7,500	54m	3.3s	7	2h5m	11272	89%	83%
2	3	5	7,500	54m	3.9s	6	5h30m	12247	97%	86 %

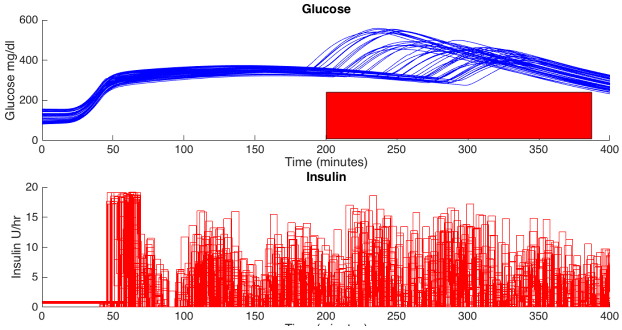


Figure 10: Simulations of samples from the likely falsifying interval for property #6, showing glucose (top) and insulin.

GHz intel core i7 processor with 8 GB RAM. We note that each row of the quadrotor benchmark represents the cumulative running time for multiple seed counterexamples. Overall, the initial simulations dominate the running time, since we required a large number of simulations for our benchmarks. The time to infer boxes using Z3 was much smaller in comparison. Finally, the number of simulations needed by the test and refine procedure is also reported. In each case, the time to simulate the model dominates the overall computation time. In particular, the time to sample boxes and refine them was negligible in comparison. We will update our implementation to run multiple simulations in parallel, potentially yielding linear speedups.

Machine Learning: We also used the data collected to learn classifiers to separate the counterexamples from those satisfying the property. We used two different approaches: a simple hyperplane separator (Lin. Sep. in Tab. 2) and a decision tree classifier (DTree) implemented using Python's scikit.learn package. Each classifier was trained on part of the simulation data (80% of the data for decision trees and 50% of the data for hyperplane classifiers), while using the remaining data to test. The accuracies over the test data averaged over multiple runs of the learning process are also reported. We note that in a few cases, the hyperplane separator seems to be as precise as the falsifying intervals in the first place. However, the accuracies reported are not subject to rigorous statistical tests. Secondly, the accuracies are quite variable across different benchmark instances. Thus, the falsifying interval approach is much more expensive but yields consistent accuracies set by the threshold $c = 0.99$ with high confidence.

Finally, we also note that the decision trees are often hard to interpret even when they have promising accuracies. The decision tree obtained for the artificial pancreas benchmark property 6 is shown in the appendix.

7. RELATED WORK

The theme of sensitivity analysis is perhaps closest to the approach presented in this paper. Both local and global sensitivity analysis aim to understand the input-output relationships for complex systems. For local sensitivity analysis, the input variables are perturbed one-at-a-time around a given baseline value, and findings are plotted as sensitivity diagram (one plot for each variable depicting how output changes with respect to change in that variable), tornado diagram (a vertical bar-chart representation with variables ordered in terms of the sensitivity of the output), and spider diagram (plotting change in output value against percentage change from the nominal value) [16]. In order to understand the effects of interactions between various variables, sensitivity analysis techniques have been proposed where two or more variables are simultaneously perturbed. When the values of variables are chosen according to a given probability distribution, sensitivity analysis is known as probabilistic sensitivity analysis. A related idea is of uncertainty analysis [31] where the goal is to learn a probability distribution on the output given probability distributions over various input variables. To the best of our knowledge, ours is the first attempt to apply sensitivity analysis to analyze counterexamples of temporal logic properties of CPS.

Recently Ferrere et al. investigate the problem of slicing traces to remove portions of a trace irrelevant to the truth of a temporal property ϕ [19]. Their approach seeks to help the same process of understanding violations that motivates our work as well. However, our work also effectively accounts for the system model by searching over neighborhoods in the input space.

Statistical model checking [40, 24] is an efficient verification technique based on selective sampling of system traces in order to statistically verify the temporal logic properties of black-box systems. In this approach the traces of the systems are sampled according to a given distribution and checked against the system property until enough statistical evidence has been generated to verify the system. Statistical model checking has been effectively applied to verify temporal logic properties of cyber-physical systems [42, 11]. While the focus of statistical model checking approach is to statistically verify the system, our focus is on explaining and widening the counterexamples to assist the system designer in further debugging.

Although the emphasis is much different, our approach can be considered as learning specification with the specification being the neighborhood around the counterexample that stays a counterexample. Specification mining is a well-studied topic in program verification [4, 25] and has been recently applied [21] to mine requirements for CPS. Another closely related research direction to specification mining is precondition [28] or invariant [36] inference by computing good and bad traces of a program.

Research on robust satisfaction of MTL formulas [17] and quantitative semantics for PSTL [15] extend the traditional qualitative notion of binary satisfiability of logical formulas to quantitative notion of distance of system behavior from satisfying a specification. Such quantitative semantics provide rich insights with a counterexample related to "trend" of the satisfaction, that allows machine learning and mining approaches to uncover interesting properties of the system under verification [21, 6, 27, 17, 33]. Our work on analyzing neighborhood of counterexample is an example of such approach.

8. CONCLUSION

Thus, we have shown that our approach can be used to infer falsifying neighborhoods that allow us to conclude facts about which parts of the counterexample actually matter for the falsification. This can also potentially scale to large systems, as demonstrated by the artificial pancreas benchmark, which has nearly 111 control parameters as inputs.

However, the approach relies on a large number of simulations to generate candidates and test them. Parallelizing our approach can mitigate the overall running time of this process. At the same time, the comparison with a few classification tools demonstrates the ability of our approach to infer falsifying neighborhoods with very high accuracies and statistical confidence. However, the main drawbacks lie in the use of heuristics in the refinement procedure. As part of our future work, we will investigate the inference of intervals as a *theory* integrated inside an SMT solver so that the failure of the test procedure can directly result in the addition of new clauses to the SMT formula, thus refining the candidates in a systematic manner.

9. REFERENCES

[1] H. Abbas and G. Fainekos. Computing descent direction of mtl robustness for non-linear systems. In *2013 American Control Conference*, pages 4405–4410, June 2013.

[2] H. Abbas, A. Winn, G. Fainekos, and A. A. Julius. Functional gradient descent method for metric temporal logic specifications. In *2014 American Control Conference*, pages 2312–2317, June 2014.

[3] Houssam Abbas, Georgios Fainekos, Sriram Sankaranarayanan, Franjo Ivancic, and Aarti Gupta. Probabilistic temporal logic falsification of cyber-physical systems. *Trans. on Embedded Computing Systems (TECS)*, 12:95–, 2013.

[4] Glenn Ammons, Rastislav Bodík, and James R. Larus. Mining specifications. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '02, pages 4–16, New York, NY, USA, 2002. ACM.

[5] Yashwanth Singh Rahul Annapureddy and Georgios E. Fainekos. Ant colonies for temporal logic falsification of hybrid systems. In *Proceedings of the 36th Annual Conference of IEEE Industrial Electronics*, pages 91 – 96, 2010.

[6] Yashwanth Singh Rahul Annapureddy, Che Liu, Georgios E. Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *Tools and algorithms for the construction and analysis of systems*, volume 6605 of *LNCS*, pages 254–257. Springer, 2011.

[7] Jonathan Backer and J. Mark Keil. The mono- and bichromatic empty rectangle and square problems in all dimensions. In Alejandro López-Ortiz, editor, *LATIN 2010*, pages 14–25. Springer, 2010.

[8] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. *νZ - An Optimizing SMT Solver*, pages 194–199. Springer Berlin Heidelberg, 2015.

[9] F. Cameron, B. Wayne Bequette, D.M. Wilson, Bruce Buckingham, Huyjin Lee, and Günter Niemeyer. Closed-loop artificial pancreas based on risk management. *J. Diabetes Sci Technol.*, 5(2):368–79, 2011.

[10] Faye Cameron, Georgios Fainekos, David M. Maahs, and Sriram Sankaranarayanan. Towards a verified artificial pancreas: Challenges and solutions for runtime verification. In *Proceedings of Runtime Verification (RV'15)*, volume 9333 of *Lecture Notes in Computer Science*, pages 3–17, 2015.

[11] Edmund M Clarke and Paolo Zuliani. Statistical model checking for cyber-physical systems. In *International Symposium on Automated Technology for Verification and Analysis*, pages 1–12. Springer, 2011.

[12] Peter I. Corke. *Robotics, Vision & Control: Fundamental Algorithms in Matlab*. Springer, 2011.

[13] Chiara Dalla Man, Robert A Rizza, and Claudio Cobelli. Meal simulation model of the glucose-insulin system. *IEEE Transactions on Biomedical Engineering*, 1(10):1740–1749, 2006.

[14] Alexandre Donzé. Breach: A toolbox for verification and parameter synthesis of hybrid systems. In *CAV*, volume 6174 of *Lecture Notes in Computer Science*. Springer, 2010.

[15] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *FORMATS*, volume 6246 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 2010.

[16] Ted G Eschenbach. Spiderplots versus tornado diagrams for sensitivity analysis. *Interfaces*, 22(6):40–46, 1992.

[17] Georgios Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410:4262–4291, 2009.

[18] Georgios Fainekos, Sriram Sankaranarayanan, Koichi Ueda, and Hakan Yazarel. Verification of automotive control applications using s-taliro. In *Proceedings of the American*

Control Conference, 2012.

[19] Thomas Ferrère, Oded Maler, and Dejan Ničković. *Trace Diagnostics Using Temporal Implicants*, pages 241–258. Springer, 2015.

[20] Sumit Kumar Jha, Edmund M. Clarke, Christopher James Langmead, Axel Legay, André Platzer, and Paolo Zuliani. A bayesian approach to model checking biological systems. In *CMSB*, volume 5688 of *Lecture Notes in Computer Science*, pages 218–234. Springer, 2009.

[21] Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V. Deshmukh, and Sanjit A. Seshia. Mining requirements from closed-loop control models. In *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, HSCC '13, pages 43–52, New York, NY, USA, 2013. ACM.

[22] Robert E. Kass and Adrian E. Raftery. Bayes factors. *J. Amer. Stat. Assoc.*, 90(430):774–795, 1995.

[23] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[24] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *International Conference on Runtime Verification*, pages 122–135. Springer, 2010.

[25] W. Li, A. Forin, and S. A. Seshia. Scalable specification mining for verification and diagnosis. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 755–760, June 2010.

[26] Chiara Dalla Man, M. Camilleri, and Claudio Cobelli. A system model of oral glucose absorption: Validation on gold standard data. *Biomedical Engineering, IEEE Transactions on*, 53(12):2472 –2478, dec. 2006.

[27] Truong Nghiem, Sriram Sankaranarayanan, Georgios E. Fainekos, Franjo Ivančić, Aarti Gupta, and George J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Hybrid Systems: Computation and Control*, pages 211–220. ACM Press, 2010.

[28] Saswat Padhi, Rahul Sharma, and Todd Millstein. Data-driven precondition inference with learned features. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 42–56. ACM, 2016.

[29] Cesar C. Palerm. Physiologic insulin delivery with insulin feedback: A control systems perspective. *Computer Methods and Programs in Biomedicine*, 102(2):130 – 137, 2011.

[30] Hartmut Prautzsch, Wolfgang Boehm, and Marco Paluszny. *Bézier and B-spline techniques*. Springer Science & Business Media, 2013.

[31] Andrea Saltelli, Paola Annoni, and Beatrice D'Hombres. How to avoid a perfunctory sensitivity analysis. *Procedia - Social and Behavioral Sciences*, 2(6):7592 – 7594, 2010.

[32] Andrea Saltelli, Karen Chan, E Marian Scott, et al. *Sensitivity analysis*, volume 1. Wiley New York, 2000.

[33] Sriram Sankaranarayanan and Georgios Fainekos. Simulating insulin infusion pump risks by *In-Silico* modeling of the insulin-glucose regulatory system. In *Computational Methods in Systems Biology (CMSB)*, volume 7605 of *Lecture Notes in Computer Science*, pages 322–339, 2012.

[34] Sriram Sankaranarayanan, Suhas Akshar Kumar, Faye Cameron, B. Wayne Bequette, Georgios Fainekos, and David M. Maahs. Model-based falsification of an artificial pancreas control system. *ACM SIGBED Review (Special Issue on Medical Cyber Physical Systems)*, 2016. To Appear.

[35] Roberto Sebastiani and Silvia Tomasi. Optimization in smt with la(q) cost functions. In *Proc. Intl. Joint Conf. on Automated Reasoning (IJCAR)*, pages 484–498. Springer-Verlag, 2012.

[36] Rahul Sharma, Saurabh Gupta, Bharath Hariharan, Alex Aiken, and Aditya V Nori. Verification as learning geometric concepts. In *International Static Analysis Symposium*, pages 388–411. Springer, 2013.

[37] Garry M. Steil. Algorithms for a closed-loop artificial pancreas: The case for proportional-integral-derivative control. *J. Diabetes Sci. Technol.*, 7:1621–1631, November 2013.

[38] G.M. Steil, A.E. Panteleon, and K. Rebrin. Closed-loop insulin delivery - the path to physiological glucose control. *Advanced Drug Delivery Reviews*, 56(2):125–144, 2004.

[39] S Weinzimer, G Steil, K Swan, J Dziura, N Kurtz, and W. Tamborlane. Fully automated closed-loop insulin delivery versus semiautomated hybrid control in pediatric patients with type 1 diabetes using an artificial pancreas. *Diabetes Care*, 31:934–939, 2008.

[40] Håkan L. S. Younes and Reid G. Simmons. Statistical probabilitistic model checking with a focus on time-bounded properties. *Information & Computation*, 204(9):1368–1409, 2006.

[41] Q Zhao, B. H. Krogh, and P Hubbard. Generating test inputs for embedded control systems. *IEEE Control Systems Magazine*, pages 49–57, August 2003.

[42] Paolo Zuliani, André Platzer, and Edmund M. Clarke. Bayesian statistical model checking with application to simulink/stateflow verification. In *HSCC*, pages 243–252. ACM, 2010.

APPENDIX

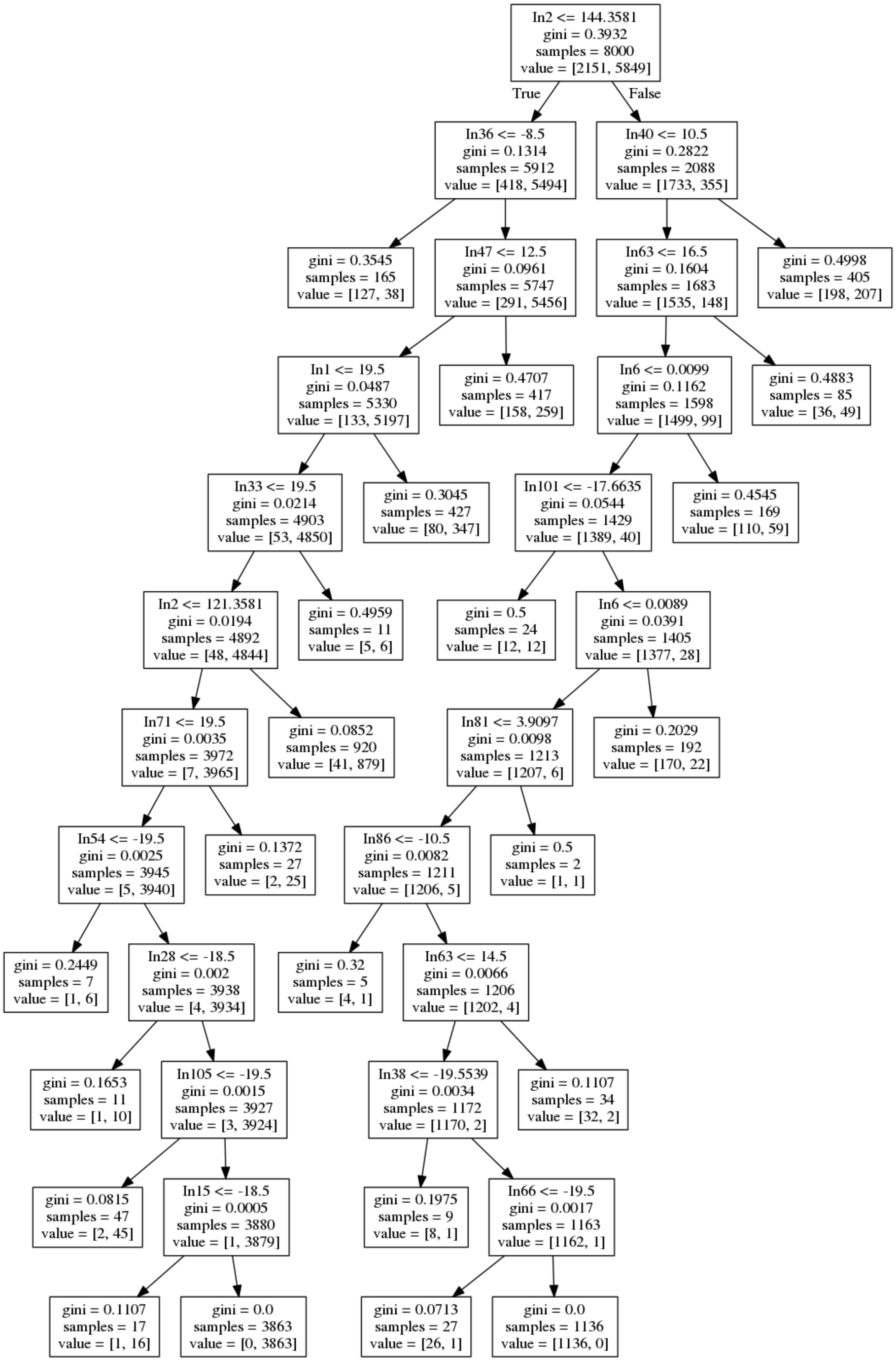


Figure 11: Decision tree inferred for the artificial pancreas example, property 6. The different dimensions are represented as In1, ... , In111. At each node `value = [a,b]` indicates `a` counterexamples and `b` satisfying samples. `gini` refers to the GINI index used by the learner to decide on the decision variable at each node.