Hyperproperties of Real-Valued Signals

Luan Viet Nguyen
University of Texas at Arlington
luanvnguyen@mavs.uta.edu

James Kapinski
Toyota Motor North America R&D
jim.kapinski@toyota.com

Xiaoqing Jin
Toyota Motor North America R&D
xiaoqing.jin@toyota.com

Jyotirmoy V. Deshmukh
Toyota Motor North America R&D
jyotirmoy.deshmukh@toyota.com

Taylor T. Johnson
Vanderbilt University
taylor.johnson@vanderbilt.edu

ABSTRACT

A *hyperproperty* is a property that requires two or more execution traces to check. This is in contrast to properties expressed using temporal logics such as LTL, MTL and STL, which can be checked over individual traces. Hyperproperties are important as they are used to specify critical system performance objectives, such as those related to security, stochastic (or average) performance, and relationships between behaviors. We present the first study of hyperproperties of cyber-physical systems (CPSs). We introduce a new formalism for specifying a class of hyperproperties defined over real-valued signals, called HyperSTL. The proposed logic extends signal temporal logic (STL) by adding existential and universal trace quantifiers into STL's syntax to relate multiple execution traces. Several instances of hyperproperties of CPSs including stability, security, and safety are studied and expressed in terms of HyperSTL formulae. Furthermore, we propose a testing technique that allows us to check or falsify hyperproperties of CPS models. We present a discussion on the feasibility of falsifying or verifying various classes of hyperproperties for CPSs. We extend the quantitative semantics of STL to HyperSTL and show its utility in formulating algorithms for falsification of HyperSTL specifications. We demonstrate how we can specify and falsify HyperSTL properties for two case studies involving automotive control systems.

ACM Reference format:
Luan Viet Nguyen, James Kapinski, Xiaoqing Jin, Jyotirmoy V. Deshmukh, and Taylor T. Johnson. 2017. Hyperproperties of Real-Valued Signals. In *Proceedings of MEMOCODE '17, Vienna, Austria, September 29-October 2, 2017,* 10 pages.
DOI: 10.1145/3127041.3127058

1 INTRODUCTION

Hyperproperties were first proposed by Clarkson and Schneider to characterize properties of security policies that cannot be defined over individual traces, such as service level agreements and information-flow properties [15]. In this work, we extend the notion of hyperproperties to cover a broad range of requirements for cyber-physical systems (CPSs), and we present a taxonomy of hyperproperties used to address security and control design concerns for CPSs. Also, we provide practical techniques for automating the process of testing hyperproperties for CPSs.

In contrast to trace properties expressed over individual execution traces, hyperproperties are defined over multiple execution traces. For example, one execution of a system cannot be checked against a service level agreement property such as *"the average time elapsed between a user's request and response over all executions should be less than 1 second"*; the property can only be evaluated over all system execution traces. Moreover, we can consider an information-flow policy of *noninterference* specified as *"for all pairs of traces of a system that have the same low-level security inputs, they will also have the same low-level security output"* [22, 41]. This noninterference property is a hyperproperty as it is expressed over all pairs of traces of a system.

Hyperproperties generalize more traditional formal properties by specifying relationships between disparate execution traces, instead of behaviors of individual execution traces. Traditional logics that consider traces individually, such as LTL, cannot be used to specify hyperproperties, and thus, hyperproperties are more expressive. Logics such as CTL and CTL* allow properties over multiple paths of a computation tree, but they do not permit comparisons between the paths themselves. Instead, to express and efficiently check hyperproperties, Clarkson et al., introduced notions of HyperLTL and HyperCTL* [14]. Both logics directly extend LTL and allow us to reason about more than one execution trace at a time. The main difference between HyperLTL and HyperCTL* is that the former requires trace quantifiers appearing at the beginning of a formula, but the latter allows us to specify them within a formula.

Although hyperproperties are well studied in the context of security policies for software systems, hyperproperties have not been explored for CPSs. For a CPS that includes stochastic factors such as noise, environment disturbance, or transducer inaccuracies, it is realistic for design engineers to expect that the system has some acceptable performance in a probabilistic sense rather than requiring an absolute performance limit be met for all individual behaviors. Acceptable performances defined over the averages of settling time, overshoot, undershoot, or error bounds cannot be specified and checked using individual execution traces; they must be quantified over all execution traces.

Recently, security-aware function modeling of CPSs has emerged as an important research topic in computer science and system verification. A CPS, which is an integration between cyber and physical subcomponents, can be vulnerable to both cyber-based and physical-based attacks [5, 19, 39, 48]. For instance, consider a

MEMOCODE '17, Vienna, Austria

DOI: 10.1145/3127041.3127058

modern automobile, which is a complex CPS composed of many computer units such as an Engine Control Unit (ECU), the Transmission Control Module (TCM), and an Electronic Brake Control Module (EBCM), all interacting with the physical world via sensors and actuators. Cyber-based attackers can gain access to the communication channels of a modern automobile through wireless or in-vehicle networks. As a result, attackers can infiltrate an ECU or EBCM to stall the engine or disable the brake system [30, 45]. An alternative method of attack involves gaining physical access to the system, for example by manipulating the signals processed by the sensors (known as sensor *spoofing*), to compromise secure information or to alter system behaviors [5, 46]. Instances of active physical-based attacks include vehicle braking system attacks, where faulty data is injected into the wheel speed sensor of a vehicle to disrupt the braking function [48], and insulin delivery device attacks, where glucose level sensor data is corrupted to compromise the function of the insulin delivery service [31]. A passive physical-based attack, also called a side-channel attack, is based on physically observing the system behavior and using leaked information to gain insights into the system implementation [26, 28, 42]. Some well-known side channel attacks are power analysis attacks [27], timing attacks [29], electromagnetic attacks [43] and differential fault analysis attacks [10].

Designing a safety-critical CPS that is entirely secure from both cyber-based and physical-based attacks is challenging or impossible. A reasonable approach is to iteratively improve a CPS control design using a *falsification* technique. Falsification is an automated best-effort approach to identify system behaviors that violate a given formal specification [40]. The design approach would be to first formally specify safety properties of a CPS that protect the system against possible cyber-based and physical-based attacks using formalisms such as temporal logic and to then iteratively improve the design using falsification, which would automatically identify vulnerabilities in the design. Despite the attractiveness of falsification techniques, attacks for CPSs often need to be defined over multiple execution traces of the system, which is something that cannot be expressed or falsified using existing temporal logics such as LTL, MTL, and STL. Thus we propose an extension to these logics that would be compatible with the appropriate specifications. In this work, we present a study of hyperproperties including stability, security and safety, as applied to CPSs. We introduce several instances of hyperproperties capturing relationships (e.g input-output relationships) between multiple traces of a CPS. We extend the syntax and semantics of STL [17] to specify hyperproperties over dense-time real-valued signals, which results in a new logic called HyperSTL. Basically, we add quantifiers at the beginning of an STL formula to express relationships between multiple traces. We also introduce a testing algorithm based on a fragment of HyperSTL and apply it to find falsifying traces for hyperproperties of industrial Simulink models. Moreover, we provide a discussion on the feasibility of falsifying or verifying various classes of hyperproperties for CPSs.

Related work. The study of hyperproperties for CPSs evaluated in this paper was inspired by the previous work of Clarkson and Schneider, who introduced hyperproperties to express security policies such as secure information flows and service level agreements [15]. In [13], Bryans et. al. presented a general formalization of opacity policies that prevent observers from deducing the truth value of a predicate; those opacity policies require behaviors to be specified over multiple paths of a system. In earlier work [37], McLean showed that some "possibilistic" security properties like restrictiveness [35], noninterference[22] and nondeducibility [49] are closure properties that cannot be expressed by individual execution traces. In [37], those properties are specified with respect to different sets of trace contractors called selective interleaving functions.

Following the introduction of hyperproperties [15], Clarkson et al. introduced HyperLTL and HyperCTL*, which are extensions to existing temporal logics, to express and check classes of information-flow hyperproperties [14]. These logics extended LTL and CTL* by adding the path quantifiers that permit specifications involving multiple paths in the system. Model checking algorithms and complexity of fragments of HyperLTL and HyperCTL* were also given in [14], which were then further exploited and applied to check some classes of information-flow hyperproperties in [41].

Prototype implementations of model checkers for HyperLTL and HyperCTL*, which assume the system is modeled as a Kripke structure, can verify some information-flow hyperproperties of a discrete-time system, but extending that work to check hyperproperties defined over continuous traces is a challenging endeavor. For complex CPS models or for models built in frameworks with proprietary or otherwise obfuscated semantics, such as Simulink®, formal verification of hyperproperties is effectively impossible, as no corresponding Kripke structure may be obtained from those models[1]. Alternatively, an easier but still difficult task is to develop an efficient testing framework, which could be used to check hyperproperties for finite collections of traces or could be used to falsify hyperproperties of a CPS model; this is the contribution of the work presented herein.

In [50], Xu et al. introduced a notion of CensusSTL that utilizes STL by adding an *outer* logic to quantify the number of individual agents of a multiagent system whose behaviors satisfy an *inner* STL formula. CensusSTL is similar to the HyperSTL proposed in this paper; however, the former is only able to specify group behaviors from different components of an individual trace while the latter allows us to express relationships between multiple traces.

The remainder of the paper is organized as follows. Section 2 reviews relevant background. Section 3 introduces several examples of hyperproperties of CPSs including stability, security and safety. Section 4 presents the syntax and semantics of HyperSTL. Section 5 and Section 6 describe the testing algorithm for two fragments of HyperSTL. Section 7 applies the proposed approach to find falsifying traces for some hyperproperties of industrial Simulink models, and Section 8 concludes the paper.

[1] Some have created their own translation of Simulink models to modeling languages with well-defined formal semantics (for example, see [3, 52]), but these translations necessarily only handle a subset of the Simulink/Stateflow modeling language. This is due to the fact that some Simulink constructs correspond to behaviors that cannot be modeled using standard frameworks for hybrid systems. One such construct is the Variable Transport Delay block, which, roughly speaking, corresponds to a delay differential equation, a construct that is not handled by standard modeling frameworks for hybrid systems.

2 PRELIMINARIES

In this section, we review the concepts of signal, system, trace property, falsification, and verification.

Signal. We define a signal w as a function $w : \mathbb{T} \to \mathbb{D}$, where $\mathbb{T} \subseteq \mathbb{R}_{\geq 0}$ is the time domain. If $\mathbb{D} = \mathbb{B}$, w is a Boolean signal whose value is either true or false, and if $\mathbb{D} = \mathbb{R}$, then we say that the signal is real-valued. A *trace*, $\mathbf{w} : \mathbb{T} \to \mathbb{D}_1 \times \ldots \times \mathbb{D}_n$, is a collection of n signals, where $\forall t \in \mathbb{T}, \mathbf{w}(t) \triangleq (w_1(t), w_2(t), ..., w_n(t))$. Intuitively, we can consider $\mathbf{w}$ as one execution trace of a continuous-time system with n variables that describes an evolution of the system. In what follows, we reserve the use of bold letters like $\mathbf{w}$, $\mathbf{w}'$ for traces (i.e., tuples of signals), while we use lowercase italicized letters such as w_i to represent signals.

System. We define a deterministic or nonstochastic[2] cyber-physical system Σ as a function mapping a given input trace in $(\mathbb{T} \to \mathbb{D}^m)$ to an output trace in $(\mathbb{T} \to \mathbb{D}^n)$. We denote by $[\![\Sigma]\!]$ the set of traces $\mathbf{w}$ such that the first m components of $\mathbf{w}$ correspond to the m input signals for $[\![\Sigma]\!]$, and the next n components correspond to the n output signals.

Trace properties. A trace property φ is a finite or infinite set of individual traces. A trace property is either satisfied or violated by any given set of traces [6, 41]. A set of traces W satisfies the trace property φ if $W \subseteq \varphi$. As noted above, an individual trace can have several components, for example, a trace could contain m input signals and n output signals of a given system Σ. We say that the trace property φ holds for a system Σ (denoted as $\Sigma \models \varphi$) if the set of input-output traces compatible with the system description is contained in the trace property, i.e., $[\![\Sigma]\!] \subseteq \varphi$.

Falsification. Given a trace property φ and a CPS Σ, the falsification problem is to find a non-empty set $W \subseteq [\![\Sigma]\!]$ such that $W \nsubseteq \varphi$.

Verification. Given a trace property ϕ, the verification problem of a CPS Σ with respect to ϕ is to show that $[\![\Sigma]\!] \subseteq \phi$.

3 HYPERPROPERTIES OF REAL-VALUED SIGNALS

Hyperproperties generalize formal properties of a system by considering sets of sets of execution traces, instead of only sets of execution traces.

Definition 3.1 (Hyperproperty). Let S denote the set of all traces. Let the power set of S be written as $P \triangleq \mathcal{P}(S)$. A *hyperproperty* is any subset of $\mathcal{P}(S)$.

We say a set of traces W satisfies a hyperproperty $\phi \subseteq P$ if $W \in \phi$. Given a hyperproperty ϕ and a system Σ, the falsification task is to find a non-empty set $W \subseteq [\![\Sigma]\!]$ such that $W \notin \phi$. Similarly, given a hyperproperty ϕ and a system Σ, the verification task is to show that $[\![\Sigma]\!] \in \phi$.

[2]Note the contrast with *stochastic* systems. In stochastic systems, one or more parts of the system have randomness associated with them; for instance, the value of a particular system parameter may be drawn from a probability distribution. The key difference is that the stochastic system may not produce the same output for a given input. Unless otherwise specified, all the systems that we consider in this paper are deterministic.

In this section, we introduce hyperproperties for deterministic systems to characterize properties such as security, safety, and stability. We focus on a class of hyperproperties capturing relationships (*e.g.*, the input-output relationship) between multiple traces of a system, and we show several examples of hyperproperties related to stability and security for CPSs. In rest of this section, we use $d_{sup}(\mathbf{w}, \mathbf{w}')$ to denote the sup-norm distance between traces $\mathbf{w}$ and $\mathbf{w}'$, where $d_{sup}(\mathbf{w}, \mathbf{w}') = \sup_{t \in \mathbb{R}_{\geq 0}} ||\mathbf{w}(t) - \mathbf{w}'(t)||$.

- *Robust behavior* is a requirement that guarantees that small differences in system inputs result in small differences in system outputs. Consider the following property: "*For all pairs of traces of a system with an input difference less than ϵ_1, the output difference should be bounded by ϵ_2*". Such a property is a hyperproperty as it requires at least two execution traces to check. This hyperproperty can be formally written as:

$$\phi_1 \triangleq \{W \in P \mid \forall \mathbf{w}, \mathbf{w}' \in W : d_{sup}(\mathbf{w}_{in}, \mathbf{w}'_{in}) \leq \epsilon_1 \implies d_{sup}(\mathbf{w}_{out}, \mathbf{w}'_{out}) \leq \epsilon_2\}. \quad (1)$$

  This type of property is related to certain stability notions, such as bounded input, bounded output (BIBO) stability and the $\mathcal{L}_2$ gain, as these notions also bound the variation in the output, based on bounded variation in the input. We note, however, that the robust behavior hyperproperty differs from BIBO stability and the $\mathcal{L}_2$ gain, as the robust behavior hyperproperty is specified over all pairs of execution traces while the BIBO and $\mathcal{L}_2$ properties are defined based on individual traces. The robust behavior hyperproperty is also related to *bisimulation relations* [18] and *conformance-closeness* [2] for a dynamical system, as all three of these properties are based on some constraints on the distances between multiple traces. In fact, we may specify bisimulation or conformance-closeness functions in terms of hyperproperties. Lastly, we note that the robust behavior hyperproperty is perhaps most closely related to Lipschitz Robustness of systems [23], which bounds differences in output behaviors based on bounded differences in input behaviors, though Lipschitz Robustness was originally developed for timed input/output systems as opposed to general CPS models.

- *Side-channel attacks* are attacks against cryptographic devices based on studying leaking information about the operations they process, such as power consumption, heat generation, and execution time. The side channel attack is an instance of an inactive physical-based attack that can be used against a CPS in which some physical behaviors are observable. Attackers can deduce the working principle of a system without either access to the system itself or an understanding of the internal operation of the system. For example, attackers can analyze an abnormal change in the power consumption of an integrated circuit while an encryption process is being executed and then reconstruct the encryption key to access secret data [27, 28]. The following property permits side-channel attacks:

$$\phi_2 \triangleq \{W \in P \mid \exists \mathbf{w} \in W : \forall \mathbf{w}' \in W : (d_{sup}(\mathbf{w}, \mathbf{w}') > 0 \wedge Power(\mathbf{w}(t)) > c_1) \implies Power(\mathbf{w}'(t)) < c_2\}, \quad (2)$$

  where $Power(\mathbf{w}(t))$ represents the power consumption corresponding to $\mathbf{w}$ over time, and c_1, c_2 are arbitrary constants such that $c_1 > c_2$. A system that satisfies this property allows an

attacker to detect that a particular behavior has occurred ($\mathbf{w}$ in Formula 2) by monitoring the power associated with the behavior. The property is a hyperproperty as it is expressed in terms of multiple traces. To ensure the safety of a system from the power-monitoring attack, the system should satisfy $\neg\phi_2$. We note that other classes of side-channel attacks such as timing attacks, electromagnetic attacks, and differential fault analysis attacks can be specified using properties similar to Formula 2.

- *Robust control invariance* is a property that can be used to synthesize safe controllers, or more to the point, can be utilized to determine whether a safe controller exists for systems with disturbances [11]. Informally, the property states that, for a given set of behaviors that is deemed safe, a control action exists, such that the system remains within the safe set for any allowable disturbance input. This can be stated formally as follows:

$$\phi_3 \triangleq \{W \in P \mid \exists \mathbf{w} \in W : \forall \mathbf{w}' \in W : (\mathbf{w}, \mathbf{w}') \models \phi\}, \quad (3)$$

where $(\mathbf{w}, \mathbf{w}') \models \phi$ means that the pair $(\mathbf{w}, \mathbf{w}')$ satisfies some property ϕ. In this formulation, $w_u(t)$ is the component of $\mathbf{w}$ that represents the controller action, $w_d(t)$ is a disturbance input, $w_y(t)$ is a system output, and $(\mathbf{w}, \mathbf{w}') \models \phi$ enforces both that $w_u = w'_u$ and $w'_y(t) \in \Omega$, where Ω is the set of safe behaviors. The robust control invariance property is related to fault data injection (FDI) attacks, which are active physical-based attacks where attackers try to input faulty data into a system to corrupt the behavior of the controller. For example, attackers can spoof the sensors of DC microgrids by injecting false data such as the past outputs of the sensors at previous time instants. This instance of FDI attack is also well known as a *replay* attack [8, 31, 48]. FDI attacks have been studied widely for CPS, and many techniques have been proposed to efficiently detect those attacks in the early stages [8, 32, 34]. However, the optimal solution is to design a system that can defend itself against FDI attacks [38]. To guarantee that a system can defend against a sensor attack, given a specification ϕ, it must be possible to choose a controller that ensures that the output of the system always satisfies ϕ, i.e. ϕ_3 must hold.

3.1 Beyond Hyperproperties?

A hyperproperty is more expressive than a trace property as it is defined over a set of sets of traces and requires multiple traces to check. If a system is modeled as trace sets, one interesting question to ask is whether there are system properties inexpressible as hyperproperties. For security policies, all properties of trace sets can be considered as hyperproperties, so the answer may be *negative* [6, 15]. For CPSs, there may exist some properties that are challenging to classify.

Consider the following property specifying the *Lyapunov stability* of a dynamical control system:

$$\phi_{Ly} \triangleq \{\forall \epsilon \in [0, \infty), \exists \delta \in [0, \epsilon), \forall \mathbf{w} \in W : \\ ||\mathbf{w}(0)|| < \delta \implies (t > 0 \wedge ||\mathbf{w}(t)|| < \epsilon)\}. \quad (4)$$

Intuitively, this property indicates that a system is Lyapunov stable if for any ϵ-*ball* around the origin, there exists a δ-*ball* around the origin ($\delta < \epsilon$) such that if the system starts within the δ-*ball*, then it will never leave the ϵ-*ball* [9]. The illustration of a Lyapunov stable system is shown in Figure 1.

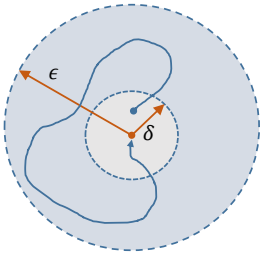

Figure 1: Illustration of a Lyapunov stable system.

Lyapunov stability is specified over the space of parameters and execution traces, and involves two alternations between universal and existential quantifiers. As we cannot check the Lyapunov stability with individual traces, it is not a trace property; so is it a hyperproperty? Consider the parameters δ and ϵ as constant signals, and then rewrite Lyapunov stability as follows:

$$\phi'_{Ly} \triangleq \{W \in P \mid \forall \mathbf{w} \in W : \exists \mathbf{w}' \in W : \forall \mathbf{w}'' \in W : \\ ||w''_{out}(0)|| < w'_{\delta}(0) \implies (t > 0 \wedge ||w''_{out}(t)|| < w_{\epsilon}(t))\}, \quad (5)$$

where a trace $\mathbf{w}$ is composed of two constant input signals w_δ, w_ϵ and an output signal w_{out}. By mapping parameters into constant signals, we can express interesting properties of the system as hyperproperties. Then Lyapunov stability is a hyperproperty that requires multiple traces to check; and it can be formally specified using the HyperSTL introduced in the next section. As to the original question of whether all system properties of interest can be specified as hyperproperties, we leave this open.

Remark 3.2 Although we focus on describing hyperproperties defined over real-valued signals, we note that there are other hyperproperties that can be specified in the context of CPSs as well. For instance, the *nondeducibility property* is an important information-flow security policy that prevents a low-level observer with sufficient knowledge of a target CPS from deducing high-level (confidential) information. The nondeducibility property is defined such that for each low-level input trace, there are more than one possible high-level input traces that produce the same output. Intuitively, an attacker should not be able to distinguish between permissible high-level behaviors based on low-level behaviors [20, 36]. On the other hand, the *noninterference property* is another important information-flow security policy that requires that high-level security users should not interfere with low-level security users. Intuitively, the outputs observed by the low-level security users remain unchanged despite the actions of the high-level security users [22]. Other variants of the noninterference property such as *noninference* [37], *observational determinism* [51], *declassification* [44], and *quantitative nonterinference* [47] are also hyperproperties that need to be specified over multiple traces. Though the nondeducibility and noninterference properties are relevant for CPS, in many cases their impact on and from real-valued signals is tenuous, and so we do not treat them further herein.

4 HYPERSTL

In this section, we introduce HyperSTL, a temporal logic that can be used to specify a class of hyperproperties of real-valued signals. The syntax and semantics of HyperSTL are naturally extended from those of STL by adding existential and universal trace quantifiers into STL's syntax to relate multiple execution traces [17].

Syntax. Let $\mathbf{v}$ be a *trace variable* from an infinite set of trace variables $\mathcal{V}$. The syntax of HyperSTL is then defined as follows:

$$\phi := \exists \mathbf{v}.\phi \mid \forall \mathbf{v}.\phi \mid \varphi$$
$$\varphi := true \mid \mu_{\mathbf{v}} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \mathbf{U}_I \varphi$$

Here, we add a universal quantifier $\forall$ and an existential quantifier $\exists$ to the syntax to indicate whether we want to specify that a formula holds over all traces or over at least one trace, respectively. For instance, $\forall \mathbf{v}.\exists \mathbf{v}'.\phi$ means that for any trace $\mathbf{w}$ assigned to trace variable $\mathbf{v}$, there exists a trace $\mathbf{w}'$ that can be assigned to trace variable $\mathbf{v}'$ such that ϕ holds on these two traces. We define $\Pi : \mathcal{V} \rightarrow S$ as a *trace assignment* (i.e., a valuation), which is a partial function mapping trace variables to traces, and S is a set of all infinite traces. Let v_i be the projection of a trace variable $\mathbf{v}$ along its i^{th} component, the projection of a trace assignment $\Pi(v_i)$ maps v_i to the i^{th} component of a trace $\mathbf{w}$ (i.e., w_i). Also, we abuse the subscript notation of a trace's component to write its corresponding trace variable's component in a HyperSTL formula, e.g., w_{out} is represented by v_{out}. A trace $\mathbf{w}$ can be Booleanized through atomic predicates of the form $\mu_{\mathbf{w}} \triangleq f(w_1(t), w_2(t), ..., w_n(t)) > 0$, where f is a real-valued function. Then, $\mu_{\mathbf{v}} = f(\Pi(\mathbf{v})(t)) > 0$ represents a Booleanized atomic predicate $\mu_{\mathbf{w}}$ if $\mathbf{v}$ is instanced by $\mathbf{w}$. Also, I is an interval over $\mathbb{R}_{\geq 0}$ such as $[a, b)$, (a, b), $(a, b]$, $[a, b]$, $(a, +\infty)$, or $[a, +\infty)$, where a, b are real numbers and $0 \leq a < b$. If I is not specified, we assume that $I = [0, \infty)$. We also allow Boolean operators $\vee$ and $\implies$ with their standard meaning. Temporal operators used in HyperSTL formulas include *always* ($\Box$), *eventually* ($\Diamond$), and *until* (U), respectively, where $\Diamond_I \varphi = true \mathbf{U}_I \varphi$, and $\Box_I \varphi = \neg\Diamond_I \neg\varphi$. Note that we use trace variables such as $\mathbf{v}$, $\mathbf{v}'$ to express HyperSTL formula and the corresponding traces represented by these trace variables like $\mathbf{w}$ $\mathbf{w}'$ to interpret the formula. Consider the HyperSTL formula $\phi := \exists \mathbf{v}.\forall \mathbf{v}'.\Box_{[0,1]}(||\mathbf{v} - \mathbf{v}'|| < 1)$. This property says that there is always a trace $\mathbf{w}$, such that for all times in the interval $[0, 1]$, every other trace $\mathbf{w}'$ is at a bounded distance of 1 from $\mathbf{w}$.

Boolean Semantics. A HyperSTL formula satisfied by a set of traces W at a time t is written as $\Pi, t \models_W \phi$, The validity judgment of a HyperSTL formula at a given time t is specified according to the following recursive semantics:

$$\begin{aligned}
\Pi, t \models_W \exists \mathbf{v}.\phi \quad &\text{iff} \quad \text{exists } \mathbf{w} \in W \ : \ \mathbf{w} \models \phi \text{ and } \Pi(\mathbf{v}) = \mathbf{w} \\
\Pi, t \models_W \forall \mathbf{v}.\phi \quad &\text{iff} \quad \text{forall } \mathbf{w} \in W \ : \ \mathbf{w} \models \phi \text{ and } \Pi(\mathbf{v}) = \mathbf{w} \\
\Pi, t \models_W \mu_{\mathbf{v}} \quad &\text{iff} \quad f(\Pi(\mathbf{v})(t)) > 0 \\
\Pi, t \models_W \neg\varphi \quad &\text{iff} \quad \Pi, t \not\models_W \varphi \\
\Pi, t \models_W \varphi_1 \wedge \varphi_2 \quad &\text{iff} \quad \Pi, t \models_W \varphi_1 \text{ and } \Pi, t \models_W \varphi_2 \\
\Pi, t \models_W \varphi_1 \mathrm{U}_I \varphi_2 \quad &\text{iff} \quad \exists t_1 \in t + I \text{ s.t } \Pi, t_1 \models_W \varphi_2 \\
&\quad \text{and } \forall t_2 \in [t, t_1] \text{ s.t } \Pi, t_2 \models_W \varphi_1
\end{aligned}$$

Using HyperSTL, we can express the hyperproperties described in Section 3 over some time interval $[t_1, t_2]$ as follows[3].

- The robust behavior in Formula 1 can be specified as:

$$\phi_1' \triangleq \forall \mathbf{v}.\forall \mathbf{v}'.\ \Box_{[t_1,t_2]}(d_{sup}(\mathbf{v}_{in}, \mathbf{v}'_{in}) \leq \epsilon_1 \implies d_{sup}(\mathbf{v}_{out}, \mathbf{v}'_{out}) \leq \epsilon_2). \tag{6}$$

- The power-monitoring attack in Formula 2 can be written as:

$$\phi_2' \triangleq \exists \mathbf{v}.\forall \mathbf{v}'.\ \Box_{[t_1,t_2]}((d_{sup}(\mathbf{v}, \mathbf{v}') > 0 \wedge Power(\mathbf{v}) > c_1) \implies Power(\mathbf{v}') < c_2). \tag{7}$$

Furthermore, we can rewrite the Lyapunov stability specified in Formula 5 as the following HyperSTL formula

$$\phi_{Ly}'' \triangleq \forall \mathbf{v}.\exists \mathbf{v}'.\forall \mathbf{v}''.\ (v''_{out} < v'_{\delta} \implies \Box_{(0,\infty)} v''_{out} < v_{\epsilon}). \tag{8}$$

According to the possible alternation of quantifiers in a HyperSTL's syntax, we classify the above HyperSTL formulae into two fragments:

(a) *alternation-free* HyperSTL formulae including one type of quantifier, and
(b) *k-alternation* HyperSTL formulae that have k number of alternations between existential and universal quantifiers.

Thus, the robust behavior property can be expressed using alternation-free HyperSTL while the power-monitoring attack property can be specified using 1-alternation HyperSTL. The Lyapunov stability property is more complex as it must be expressed using 2-alternation HyperSTL.

Falsification or Verification of Hyperproperties? We have introduced several classes of hyperproperties for CPSs and a temporal logic approach to express them. Next, we investigate whether we can falsify or verify those hyperproperties using existing methods. Hyperproperties are more complex and expressive than traditional properties, and performing falsification and verification for hyperproperties is harder, in many cases. Despite this, we observe that certain classes of hyperproperties can be falsified or verified. For instance, we can falsify an alternation-free HyperSTL formula that contains a universal quantifier (e.g., the robust behavior hyperproperty), and we can verify an alternation-free HyperSTL formula that contains an existential quantifier. For the class of hyperproperties that includes alternating quantifiers, falsification or verification are often undecidable unless we impose some assumption about the sets of execution traces (e.g., quantified over some finite set of traces with bounded time).

4.1 t-HyperSTL

We introduce t-HyperSTL as a fragment of HyperSTL in which a nesting structure of temporal logic formulas involving different traces is not allowed. For example, a formula $\forall \mathbf{v}.\exists \mathbf{v}'.\Box_{[0,2]}\mathbf{v} > 1 \implies \Diamond_{[1,2]}\mathbf{v}' > 2$ is allowed but a formula $\forall \mathbf{v}.\exists \mathbf{v}'.\Box_{[0,2]}(\mathbf{v} > 1 \implies \Diamond_{[1,2]}\mathbf{v}' > 2)$ is not allowed. Also, t-HyperSTL restricts the *until* operator to be specified over an individual trace, e.g., t-HyperSTL does not allow the formula $\forall \mathbf{v}.\exists \mathbf{v}'.(\mathbf{v} > 1)\mathrm{U}_{[0,1]}(\mathbf{v}' > 2)$.

Inherited from the syntax of HyperSTL, t-HyperSTL formulae are also classified into alternation-free and k-alternation types.

[3] For a robust control invariance hyperproperty, an instance of the corresponding HyperSTL formula will be shown in Section 7.2.

t-HyperSTL suffices to express the class of hyperproperties formulated in Section 3, and its corresponding semantics, which is more restrictive than that of HyperSTL, allow us to perform falsification for these hyperproperties.

Quantitative Semantics. The quantitative semantics of t-HyperSTL reflects the *robustness satisfaction* of a t-HyperSTL formula. It is a natural extension of those for STL [17, 33]. Given χ is a real-valued function of a formula φ, a trace assignment Π, a trace variable $\mathbf{v}$, and a time t, the quantitative semantics of t-HyperSTL is defined inductively as follows:

$$\begin{aligned}
\chi(\varphi, \Pi, \exists \mathbf{v}, t) &= \max_{\mathbf{w} \in W} \chi(\varphi, \Pi(\mathbf{v}) = \mathbf{w}, t) \\
\chi(\varphi, \Pi, \forall \mathbf{v}, t) &= \min_{\mathbf{w} \in W} \chi(\varphi, \Pi(\mathbf{v}) = \mathbf{w}, t) \\
\chi(\mu_{\mathbf{v}} > 0, \Pi, \mathbf{v}, t) &= \mu_{\mathbf{v}} \\
\chi(\neg\varphi, \Pi, \mathbf{v}, t) &= -\chi(\varphi, \Pi, \mathbf{v}, t) \\
\chi(\varphi_1 \wedge \varphi_2, \Pi, \mathbf{v}, t) &= \min\left(\chi(\varphi_1, \Pi, \mathbf{v}, t), \chi(\varphi_2, \Pi, \mathbf{v}, t)\right) \\
\chi(\varphi_1 \mathrm{U}_I \varphi_2, \Pi, \mathbf{v}, t) &= \sup_{t_1 \in t+I} \min\Big(\chi(\varphi_2, \Pi, \mathbf{v}, t_1), \\
&\qquad \inf_{t_2 \in [t, t_1]} \chi(\varphi_1, \Pi, \mathbf{v}, t_2)\Big)
\end{aligned}$$

5 FALSIFYING ALTERNATION-FREE T-HYPERSTL

We first consider the falsification of alternation-free t-HyperSTL formulae. This fragment of HyperSTL is expressive enough to capture a broad range of hyperproperties specifying input-output relationships over all pairs of execution traces. We use a translation scheme called self-composition [7], which allows us to falsify an alternation-free t-HyperSTL formula that includes only universal quantifiers using a robust testing method for a normal STL formula. Then, given an alternation-free t-HyperSTL that includes universal quantifiers, we attempt to find a set of falsifying traces for CPSs corresponding to this formula.

Falsification algorithm. The procedure that addresses the falsification problem of a system Σ with respect to a given hyperproperty φ_h over a time duration T is shown in Algorithm 1, and further interpreted as follows.

- We first transform the alternation-free t-HyperSTL formula φ_h into the equivalent STL formula φ_{STL}.
- We then call a function NewSystemGen to generate a new model that contains copies of the original system. The number of copies is equal to the number of quantifiers of the formula φ_h.
- Then, we apply existing falsification mechanisms for an STL formula such as Breach[4] [16] to compute the minimum robustness value χ_{min} of the system Σ' according to φ_{STL}. Breach allows us to parametrically generate different input signals over a parameter space. For example, parameters can represent control points, and an input signal can be created using interpolation between these points. If χ_{min} is negative we return the optimal set of parameters $\Theta_f \in \Theta$ that produces a falsifying behavior.

[4]Breach [16] is a tool that applies a best-effort approach to automatically check whether a system satisfies a given STL formula.

Algorithm 1 Falsification of alternation-free t-HyperSTL

```
Require: a system Σ, a parameter space Θ,
a t-HyperSTL formula φ_h, a time duration T,
a maximum number of simulations N
begin
  φ_STL ← HyperSTL2STL(φ_h) // transform specification
  Σ' ← NewSystemGen(Σ, φ_h) // transform model
  χ_min, Θ_f ← FalsifySTL(Σ', φ_STL, Θ, T, N)
  if χ_min < 0 then
    return Θ_f
  end
end
```

We note that, unlike formal verification, performing falsification cannot ensure a system is always safe; even if falsification fails to identify a falsifying behavior, a counter-example may still exist.

Example 5.1. Consider a mechanical mass-spring damper system whose dynamics are defined by the second-order ordinary differential equation:

$$\ddot{x}(t) + 2\dot{x}(t) + 5x(t) = 3F(t), \tag{9}$$

where x is the vertical position of the mass, and F is the random external force. The robust behavior hyperproperty of the system is specified as follows: for all pairs of traces of the system with the external force difference less than ϵ_1 , the output difference should be bounded by ϵ_2; here $\epsilon_1 = 0.2$ and $\epsilon_2 = 0.3$. We apply the Algorithm 1 to falsify the robust behavior hyperproperty for the system with a duration $T = 10$ seconds. Formula 6 can be reduced to the normal STL formula as follows:

$$\phi_M \triangleq \square_{[0,10]}(\rho_{in} \leq \epsilon_1 \implies \rho_{out} \leq \epsilon_2), \tag{10}$$

where a trace $\mathbf{p} \triangleq (\rho_{in}, \rho_{out})$ of the system Σ' captures the input-output difference between two traces $\mathbf{w}, \mathbf{w}'$ of the original system Σ', e.g., $\rho_{in}(t) = ||w_{in}(t) - w'_{in}(t)||$. Here, the system Σ' contains two copies of the mechanical mass-spring damper system Σ. The falsification result shown in Figure 2 illustrates the inductive checking procedure for the satisfaction of Formula 10 using Breach, where $alw_{[0,10]}$ is equivalent to $\square_{[0,10]}$, and the left y-axis denotes robustness degree. Here, we observe that the violation of the robust behavior hyperproperty of the mechanical mass-spring damper system occurs during the overshoot period of the outputs of the system.

Remark 5.2 There is a duality between addressing the falsification problem of an alternation-free t-HyperSTL that only contains universal quantifiers and solving the verification problem of an alternation-free t-HyperSTL that only contains existential quantifiers. Given an alternation-free t-HyperSTL such as $\exists \mathbf{v}.\exists \mathbf{v}'.\phi_e$, our purpose is to extensively simulate a system and find a single pair of execution traces of the system that satisfies ϕ_e. Here, we do not attempt to falsify the system, but verify the system. Thus, this process is dual to finding the falsifying traces of the system corresponding to the formula $\forall \mathbf{v}.\forall \mathbf{v}'.\neg\phi_e$.
Also, we note that we can leverage Algorithm 1 such that it includes a parameter synthesis approach to mine hyperproperties for CPSs, as in [24, 25]. For instance, we could use a requirement mining

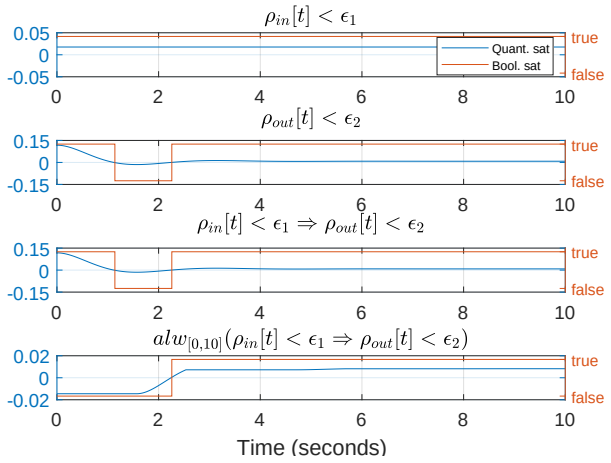

Figure 2: Falsification result of the mass-spring damper system. The counterexample pair of traces found by Breach for the robust behavior hyperproperty.

approach to automatically infer appropriate values for the ϵ_1 and ϵ_2 variables in Formula 10.

6 FALSIFYING K-ALTERNATION T-HYPERSTL

Falsifying k-alternation t-HyperSTL formulas is a challenging task, as it requires us to examine all execution traces of a system. Consider a 1-alternation t-HyperSTL formula such as $\exists \mathbf{v}.\forall \mathbf{v}'.\phi$; falsifying a system for this property is as hard as verifying the system, since we need to show that for all traces $\mathbf{w} \in S$, there exists a trace $\mathbf{w}'$ that the formula ϕ is violated, where S is an infinite set of traces. It is even more difficult to perform falsification for CPSs whose dynamics evolve continuously over time. Furthermore, if a hyperproperty contains more than one alternation of quantifiers (e.g. the Lyapunov stability property), the falsifying algorithm may suffer an exponential growth in complexity. Despite this, if we assume a CPS can be modeled by a finite set of traces, we can develop a falsifying algorithm for the system that can prove or disprove ϕ.

In general, there may not exist a unique answer to the question of whether we can verify or falsify a system with respect to the formula $\exists \mathbf{v}.\forall \mathbf{v}'.\phi$ using finite simulations. We can consider several possible answers for that question as follows.

- **Case 1**: if both $\mathbf{w}, \mathbf{w}'$ belong to some infinite set of traces, then we can neither verify nor falsify ϕ.
- **Case 2**: if $\mathbf{w}$ belongs to an infinite set of traces and $\mathbf{w}'$ belongs to a finite set of traces, then we cannot falsify but we can verify ϕ.
- **Case 3**: if $\mathbf{w}$ belongs to a finite set of traces and $\mathbf{w}'$ belongs to an infinite set of traces, then we cannot verify but we can falsify ϕ.
- **Case 4**: If both $\mathbf{w}$ and $\mathbf{w}'$ belong to a finite set of n traces, we are able to verify the system with n simulations as well as falsify the system with $\frac{n(n-1)}{2}$ simulations.

We note that in all of the cases that we are able to falsify the system corresponding to the formula $\exists \mathbf{v}.\forall \mathbf{v}'.\phi$ with finite simulations, we can apply Algorithm 1 to transform the falsification problem to another equivalent problem that uses a traditional STL specification.

Table 1: Feasibility of solving the falsification and verification problems for properties and hyperproperties expressed using STL and k-alternation t-HyperSTL under two assumptions: A1) using finite simulation and A2) applying a verification oracle that can do reachability analysis with respect to the last quantifier.

Type	A1: Finite Simulation		A2 : Verification Oracle
	Falsification	Verification	on the Last Quantifier
$\forall$	Yes	No	-
$\exists$	No	Yes	-
$\forall\exists$	No	No	$\forall$
$\exists\forall$	No	No	$\exists$
$\forall\exists\forall$	No	No	$\forall\exists$
$\exists\forall\exists$	No	No	$\exists\forall$

The falsification procedure is similar to solving the falsification problem of alternation-free t-HyperSTL.

For the case that both execution traces of a system, $\mathbf{w}$ and $\mathbf{w}'$, belong to some infinite sets, and if we have a verification oracle to address the last quantifier (e.g., by conservatively estimating the set of possible system behaviors, under certain conditions), we can either falsify or verify the system. Given a set of initial states, a verification oracle can be a method that mathematically overapproximates the reachable set of the system or a simulation-based technique [1, 21] that may verify the system with finite simulations.

Alternatively, for a hyperproperty that requires two or more alternations of quantifiers to express, even if we have a verification oracle corresponding to the last quantifier, we can neither falsify nor verify a system. Using a verification oracle, the feasibility of addressing the falsification and verification problems associated with a k-alternation t-HyperSTL formula is equivalent to that of a $(k-1)$-alternation t-HyperSTL formula; this is shown in Table 1. We emphasize that any hyperproperties for general CPSs that are as complex as, or more complicated than Lyapunov stability, are not verifiable or falsifiable without reasonable restrictions on sets of execution traces.

7 CASE STUDY

In this section, we introduce two proof-of-concept case studies in the domain of automotive control systems: a) an industrial-scale Simulink model of a closed-loop airpath control (APC) system and b) a Simulink model of a fault-tolerant fuel (FTF) control system. We will demonstrate how to apply the testing framework of HyperSTL built on top of Breach to falsify the robust behavior hyperproperty of the APC system, and the robust control invariance hyperproperty of the FTF system under FDI attacks.

7.1 Airpath Control Model

We use a prototype APC system to evaluate the capability of our proposed method on an industrial control system. The APC is a key subsystem for a hydrogen Fuel-Cell (FC) vehicle powertrain. The

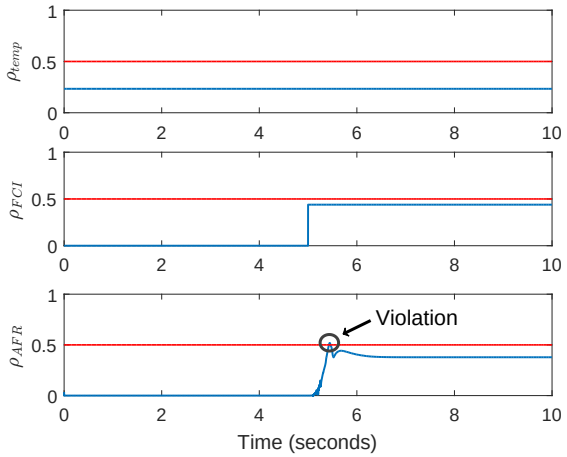

Figure 3: Falsification result of the APC system. The counterexample pair of traces found by Breach for the robust behavior hyperpropperty.

purpose of the APC is to regulate the air flow rate into the FC stack using multiple actuators. The FC stack generates electrical power for the vehicle using a mixture of air and hydrogen. The FC stack only operates under restricted conditions, such as temperature, pressure and moisture level within the stack. An excess of moisture in the stack will impede the performance while moisture deficiency could permanently damage the FC stack. Thus, to achieve high performance while still operating the system in a safe regime, the controller is required to accurately regulate the air flow rate.

The closed-loop Simulink model of the APC system is complex; it contains more than 7,000 Simulink blocks such as integrators, saturations, S-Function blocks, lookup tables, and data store memory blocks. The model has two input signals including i) the ambient temperature and ii) the fuel cell current request (FCI). Details of the system, such as units and expected signal ranges, are suppressed due to proprietary concerns. Intuitively, an FCI value is proportional to the desired torque requested by the driver, which is ultimately based on the accelerator pedal angle. The output of the APC system is an air flow rate (AFR). The purpose of the controller model is to regulate the AFR to some desirable reference value. To ensure the APC system works properly, for some small perturbations of the ambient temperature and FCI values, the differences in AFR values should be bounded within a desirable range. In other words, to avoid unexpected changes in the air flow rate at the inlet of an FC stack, which may cause undesirable behavior, the system should satisfy the robust behavior hyperproperty. The robust behavior hyperproperty of the APC system can be formalized as follows,

$$\begin{aligned}\phi_{APC} \triangleq \{W \in P \mid \forall \mathbf{w}, \mathbf{w}' \in W : \\ (d_{sup}(w_{temp}, w'_{temp}) \leq \epsilon_1 \wedge d_{sup}(w_{FCI}, w'_{FCI}) \leq \epsilon_2) \\ \implies d_{sup}(w_{AFR}, w'_{AFR}) \leq \epsilon_3)\},\end{aligned} \tag{11}$$

which can be translated to the following STL formula using Algorithm 1 to perform the falsification task,

$$\phi'_{APC} \triangleq \square_{[0,T]}((\rho_{temp} \leq \epsilon_1 \wedge \rho_{FCI} \leq \epsilon_2) \implies \rho_{AFR} \leq \epsilon_3), \tag{12}$$

where a trace $\mathbf{w}$ is composed of the temperature and FCI input signals w_{temp} and w_{FCI} respectively, and the AFR output signal w_{AFR}. Here, we create a new model including two copies of the original APC system; and a trace $\mathbf{p} \triangleq (\rho_{temp}, \rho_{FCI}, \rho_{AFR})$ of the new model captures the input-output difference between two traces $\mathbf{w}, \mathbf{w}'$ of the original model, for instance, $\rho_{temp}(t) = ||w_{temp}(t) - w'_{temp}(t)||$.

The result of falsification of the robust behavior hyperpropety of the APC system is shown in Figure 3, where the blue lines present the distance signals ρ_{temp}, ρ_{FCI}, ρ_{AFR} respectively, and the red lines demonstrate their corresponding bounds. Here, the parameter values selected by a design engineer are normalized to 0.5. That is, $\epsilon_1 = 0.5$, $\epsilon_2 = 0.5$, and $\epsilon_3 = 0.5$. The sampling time is 0.001024 seconds and the simulation time T is 10 seconds. For proprietary reasons, we normalize the quantities and suppress the units for the data shown in the figure. The counterexample pairs of traces reported by Breach demonstrate a behavior where the output difference exceeds its allowed bounds when the input differences are still less than their given thresholds, which is a violation of Formula 12. Finding this counter-example is significant, as it can help automotive control engineers to improve the controller design to eliminate such an undesirable behavior of the APC system.

7.2 Fault-tolerant Fuel Model

We consider a fault-tolerant fuel (FTF) model that includes both Simulink blocks and Stateflow charts[5]. The model has two external input signals, engine speed and throttle command, and one output signal, which is the effective air-fuel ratio inside the combustion chamber. The model also contains four sensors measuring throttle angle, engine speed, the amount of residual oxygen in the exhaust gas (EGO), and the manifold absolute pressure (MAP). The controller has three different control strategies: a normal operation mode, which is used when no sensor faults are present, a fault mitigation mode, which is used when one sensor fault has occurred, and a mode that disables fuel control, which is used when two or more sensor faults are detected. We only consider the normal and fault mitigation modes for this example. The goal of the controller is to regulate the air-fuel ratio output, denoted as λ, so that it remains within a desirable range, despite a failure in at most one sensor.

In this case study, we evaluate the ability of the FTF controller to tolerate an engine speed sensor fault. In the original version of the model, a speed sensor fault consists of the speed sensor output being set to 0.0 rad/sec; the controller detects the fault when the sensor reading equals 0.0. In the modified version that we use, we do not fix the controller mode based on the sensor reading, but instead we evaluate the controller performance when either the normal or fault mitigation modes are selected. In the modified version of the model that we use, a speed sensor fault consists of a sensor output producing a fixed but randomly selected value in the sensor range $[0, 620]$ rad/sec. This kind of sensor fault could occur when an attacker uses a sensor spoofing approach to inject incorrect measurements into the sensor readings or when a real fault occurs in the speed sensor. We use the robust control invariance property to specify desired controller performance in the presence of the

[5]We use a modified version of the FTF model available at https://www.mathworks.com/help/simulink/examples/modeling-a-fault-tolerant-fuel-control-system.html

indicated class of sensor faults:

$$\phi_{FTF} \triangleq \exists \mathbf{v}.\forall \mathbf{v}'.\Box_{[\tau,\infty]}(d_{sup}(v_u, v'_u) = 0 \implies 0.8\lambda_{ref} \leq v'_\lambda \leq 1.2\lambda_{ref}), \quad (13)$$

where λ_{ref} is the reference value of the air-fuel ratio λ, and τ is the settling time. Here, a trace variable $\mathbf{v}$ can be mapped to a trace $\mathbf{w}$ composed of the controller input w_u corresponding to a controller mode decision, a disturbance w_d representing the fixed random sensor input injected into the speed sensor, and an output w_λ.In general, we cannot falsify Formula 13 according to the discussion shown in Table 1; however, for systems like the FTF model that have a finite set of control strategies, we can effectively perform falsification by creating a new model that contains copies of the original system, one copy for each control mode (two copies, in this case). The external input (the speed sensor reading) is connected to each of the copies of the model. The specification ϕ_{FTF} is converted to the following equivalent formula in standard STL:

$$\hat{\phi}_{FTF} \triangleq \forall w_d.\Box_{[\tau,\infty]}(0.8\lambda_{ref} \leq w_{\lambda_1} \leq 1.2\lambda_{ref} \vee 0.8\lambda_{ref} \leq w_{\lambda_2} \leq 1.2\lambda_{ref}), \quad (14)$$

where w_{λ_1} and w_{λ_2} are the air-fuel ratios of the first and second copies of the model. We note that Formula 14 is arrived at by applying the quantitative semantics provided in Sec. 4; the disjunction in Formula 14 appears due to the $\exists$ quantifier in Formula 13, which effectively applies a max operator over the two available control modes. The formula $\hat{\phi}_{FTF}$ can be tested using the falsification methods for traditional STL available in Breach.

Figure 4 illustrates the falsification result of the FTF model. The blue lines correspond to a simulation trace representing the falsifying behavior, the green line illustrates an instance of the correct speed, and the red lines represent the error bound of λ, where $\tau = 10$ seconds, $T = 50$ seconds, and $\lambda_{ref} = 14.6$. Based on the results, we can conclude that there exists a trace, which includes outputs w_{λ_1} and w_{λ_2} that both evolve beyond the tolerance bound regardless of whether the controller operates in the normal mode or the fault mitigation mode (i.e., the performance requirement is violated despite which control mode is used). This experiment demonstrates the capability of using a falsification approach to automatically test hyperproperties for CPSs.

8 CONCLUSION AND FUTURE WORK

In this paper, we represented the first study of the hyperproperties of CPSs. We defined a new temporal logic, called HyperSTL, to express several hyperproperties including stability, security, and safety for CPSs. HyperSTL allows us to effectively specify more general requirements of CPS rather than STL as it can express the relationships between multiple execution traces. The testing framework of t-HyperSTL, a fragment of HyperSTL, was also given and applied to falsify the robust behavior hyperproperty of a hydrogen fuel-cell powertrain model, and the robust control invariance hyperproperty of the fuel control model under a fault data injection attack. We also discuss the feasibility of performing the falsification and verification for various classes of hyperproperties for CPSs.

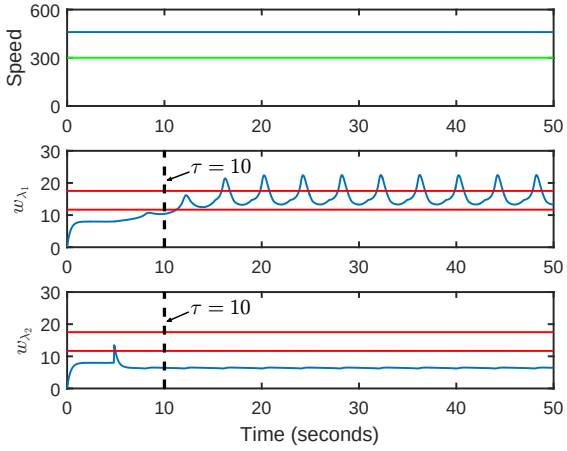

Figure 4: A pair of falsifying traces found by Breach illustrating the FTF model cannot tolerate the fault under a speed sensor fault.

Future Work. We first plan to introduce a library of HyperSTL fomulae that encapsulates different general classes of hyperproperties of CPS including those presented in this paper. Second, the falsification algorithm of HyperSTL proposed in the paper is incomplete as it relies on self-composition (i.e. making copies of a system) and only falsifies a restricted class of hyperproperties. Thus, extending the falsification algorithm to bypass self-composition to falsify more interesting hyperproperties is planned. Also, the monitoring algorithms of HyperLTL recently proposed in [4, 12] could be applied to HyperSTL.

9 ACKNOWLEDGMENTS

The authors would like to acknowledge Borzoo Bonakdarpour and his research group for insightful comments that helped us refine our definition of HyperSTL. We would also like to thank the anonymous reviewers for their feedback. The material presented in this paper is based upon work supported by the National Science Foundation (NSF) under grant numbers CNS 1464311, EPCN 1509804, and SHF 1527398, the Air Force Research Laboratory (AFRL) through contract numbers FA8750-15-1-0105, and FA8650-12-3-7255 via subcontract number WBSC 7255 SOI VU 0001, and the Air Force Office of Scientific Research (AFOSR) under contract numbers FA9550-15-1-0258 and FA9550-16-1-0246. The U.S. government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of AFRL, AFOSR, or NSF.

REFERENCES

[1] Houssam Abbas, Bardh Hoxha, Georgios Fainekos, and Koichi Ueda. 2014. Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems. In *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2014 IEEE 4th Annual International Conference on.* IEEE, 1–6.

[2] Houssam Abbas, Hans Mittelmann, and Georgios Fainekos. 2014. Formal property verification in a conformance testing framework. In *Formal methods and models for codesign (memocode), 2014 twelfth acm/ieee international conference on.* IEEE, 155–164.

[3] Aditya Agrawal, Gyula Simon, and Gabor Karsai. 2004. Semantic Translation of Simulink/Stateflow Models to Hybrid Automata Using Graph Transformations. *Electronic Notes in Theoretical Computer Science* 109 (2004), 43 – 56.

[4] Shreya Agrawal and Borzoo Bonakdarpour. 2016. Runtime verification of k-safety hyperproperties in HyperLTL. In *Computer Security Foundations Symposium (CSF), 2016 IEEE 29th*. IEEE, 239–252.

[5] Mohammad Al Faruque, Francesco Regazzoni, and Miroslav Pajic. 2015. Design methodologies for securing cyber-physical systems. In *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis*. IEEE Press, 30–36.

[6] Mack W Alford, Jean-Pierre Ansart, Günter Hommel, Leslie Lamport, Barbara Liskov, Geoff P Mullery, and Fred B Schneider. 1985. *Distributed systems: methods and tools for specification. An advanced course*. Springer-Verlag New York, Inc.

[7] Gilles Barthe, Pedro R D'Argenio, and Tamara Rezk. 2004. Secure information flow by self-composition. In *Computer Security Foundations Workshop, 2004. Proceedings. 17th IEEE*. IEEE, 100–114.

[8] Omar Beg, Taylor Johnson, and Ali Davoudi. 2017. Detection of False-data Injection Attacks in Cyber-Physical DC Microgrids. *IEEE Transactions on Industrial Informatics* (2017).

[9] Nam Parshad Bhatia and Giorgio P Szegö. 2002. *Stability theory of dynamical systems*. Springer Science & Business Media.

[10] Eli Biham and Adi Shamir. 1997. Differential fault analysis of secret key cryptosystems. In *Annual International Cryptology Conference*. Springer, 513–525.

[11] F. Blanchini. 1999. Survey Paper: Set Invariance in Control. *Automatica* 35, 11 (Nov. 1999), 1747–1767.

[12] Noel Brett, Umair Siddique, and Borzoo Bonakdarpour. 2017. Rewriting-Based Runtime Verification for Alternation-Free HyperLTL. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 77–93.

[13] Jeremy W Bryans, Maciej Koutny, Laurent Mazaré, and Peter YA Ryan. 2008. Opacity generalised to transition systems. *International Journal of Information Security* 7, 6 (2008), 421–435.

[14] Michael R Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K Micinski, Markus N Rabe, and César Sánchez. 2014. Temporal logics for hyperproperties. In *International Conference on Principles of Security and Trust*. Springer, 265–284.

[15] Michael R Clarkson and Fred B Schneider. 2010. Hyperproperties. *Journal of Computer Security* 18, 6 (2010), 1157–1210.

[16] Alexandre Donzé. 2010. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification*. Springer, 167–170.

[17] Alexandre Donzé, Thomas Ferrere, and Oded Maler. 2013. Efficient robust monitoring for STL. In *Computer Aided Verification*. Springer, 264–279.

[18] Georgios E Fainekos, Antoine Girard, and George J Pappas. 2006. Temporal logic verification using simulation. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 171–186.

[19] Thoshitha T Gamage, Bruce M McMillin, and Thomas P Roth. 2010. Enforcing information flow security properties in cyber-physical systems: A generalized framework based on compensation. In *Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual*. IEEE, 158–163.

[20] Thoshitha T. Gamage, Bruce M. McMillin, and Thomas P. Roth. 2010. Enforcing Information Flow Security Properties in Cyber-Physical Systems: A Generalized Framework Based on Compensation.. In *COMPSAC Workshops*. IEEE Computer Society, 158–163.

[21] Antoine Girard and George J Pappas. 2006. Verification using simulation. In *International Workshop on Hybrid Systems: Computation and Control*. Springer, 272–286.

[22] Joseph A Goguen and José Meseguer. 1982. Security policies and security models. In *Security and Privacy, 1982 IEEE Symposium on*. IEEE, 11–11.

[23] Thomas A. Henzinger, Jan Otop, and Roopsha Samanta. 2016. *Lipschitz Robustness of Timed I/O Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 250–267.

[24] Bardh Hoxha, Adel Dokhanchi, and Georgios Fainekos. [n. d.]. Mining parametric temporal logic properties in model-based design for cyber-physical systems. *International Journal on Software Tools for Technology Transfer* ([n. d.]), 1–15.

[25] Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V Deshmukh, and Sanjit A Seshia. 2015. Mining requirements from closed-loop control models. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 34, 11 (2015), 1704–1717.

[26] John Kelsey, Bruce Schneier, David Wagner, and Chris Hall. 1998. Side channel cryptanalysis of product ciphers. In *European Symposium on Research in Computer Security*. Springer, 97–110.

[27] Paul Kocher, Joshua Jaffe, and Benjamin Jun. 1999. Differential power analysis. In *Annual International Cryptology Conference*. Springer, 388–397.

[28] Paul Kocher, Ruby Lee, Gary McGraw, Anand Raghunathan, and Srivaths Moderator-Ravi. 2004. Security as a new dimension in embedded system design. In *Proceedings of the 41st annual Design Automation Conference*. ACM, 753–760.

[29] Paul C Kocher. 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Annual International Cryptology Conference*. Springer, 104–113.

[30] Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, et al. 2010. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 447–462.

[31] Chunxiao Li, Anand Raghunathan, and Niraj K Jha. 2011. Hijacking an insulin pump: Security attacks and defenses for a diabetes therapy system. In *e-Health Networking Applications and Services (Healthcom), 2011 13th IEEE International Conference on*. IEEE, 150–156.

[32] Lanchao Liu, Mohammad Esmalifalak, Qifeng Ding, Valentine A Emesih, and Zhu Han. 2014. Detecting false data injection attacks on power grid by sparse optimization. *IEEE Transactions on Smart Grid* 5, 2 (2014), 612–621.

[33] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 152–166.

[34] Kebina Manandhar, Xiaojun Cao, Fei Hu, and Yao Liu. 2014. Detection of faults and attacks including false data injection attack in smart grid using kalman filter. *IEEE transactions on control of network systems* 1, 4 (2014), 370–379.

[35] Daryl McCullough. 1987. Specifications for multi-level security and a hook-up. In *Security and Privacy, 1987 IEEE Symposium on*. IEEE, 161–161.

[36] John McLean. 1990. Security models and information flow. In *Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on*. IEEE, 180–187.

[37] John McLean. 1994. A general theory of composition for trace sets closed under selective interleaving functions. In *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*. IEEE, 79–93.

[38] Shaunak Mishra, Yasser Shoukry, Nikhil Karamchandani, Suhas Diggavi, and Paulo Tabuada. 2015. Secure state estimation: optimal guarantees against sensor attacks in the presence of noise. In *Information Theory (ISIT), 2015 IEEE International Symposium on*. IEEE, 2929–2933.

[39] Yilin Mo, Tiffany Hyun-Jin Kim, Kenneth Brancik, Dona Dickinson, Heejo Lee, Adrian Perrig, and Bruno Sinopoli. 2012. Cyber–physical security of a smart grid infrastructure. *Proc. IEEE* 100, 1 (2012), 195–209.

[40] Truong Nghiem, Sriram Sankaranarayanan, Georgios Fainekos, Franjo Ivancić, Aarti Gupta, and George J. Pappas. 2010. 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 (HSCC '10)*. ACM, New York, NY, USA, 211–220.

[41] Markus N Rabe. 2016. A temporal logic approach to information-flow control. (2016).

[42] Srivaths Ravi, Anand Raghunathan, and Srimat Chakradhar. 2004. Tamper resistance mechanisms for secure embedded systems. In *VLSI Design, 2004. Proceedings. 17th International Conference on*. IEEE, 605–611.

[43] Pankaj Rohatgi. 2009. Electromagnetic attacks and countermeasures. In *Cryptographic Engineering*. Springer, 407–430.

[44] Andrei Sabelfeld and David Sands. 2005. Dimensions and principles of declassification. In *Computer Security Foundations, 2005. CSFW-18 2005. 18th IEEE Workshop*. IEEE, 255–269.

[45] Florian Sagstetter, Martin Lukasiewycz, Sebastian Steinhorst, Marko Wolf, Alexandre Bouard, William R Harris, Somesh Jha, Thomas Peyrin, Axel Poschmann, and Samarjit Chakraborty. 2013. Security challenges in automotive hardware/software architecture design. In *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 458–463.

[46] Yasser Shoukry, Paul Martin, Paulo Tabuada, and Mani Srivastava. 2013. Non-invasive Spoofing Attacks for Anti-lock Braking Systems. In *Proceedings of the 15th International Conference on Cryptographic Hardware and Embedded Systems (CHES'13)*. Springer-Verlag, Berlin, Heidelberg, 55–72.

[47] Geoffrey Smith. 2009. On the foundations of quantitative information flow. In *International Conference on Foundations of Software Science and Computational Structures*. Springer, 288–302.

[48] Jiang Wan, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. 2015. Security-aware functional modeling of cyber-physical systems. In *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*. IEEE, 1–4.

[49] J Todd Wittbold and Dale M Johnson. 1990. Information flow in nondeterministic systems. In *Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on*. IEEE, 144–161.

[50] Zhe Xu and A Agung Julius. 2016. Census signal temporal logic inference for multiagent group behavior analysis. *IEEE Transactions on Automation Science and Engineering* (2016).

[51] Steve Zdancewic and Andrew C Myers. 2003. Observational determinism for concurrent program security. In *Computer Security Foundations Workshop, 2003. Proceedings. 16th IEEE*. IEEE, 29–43.

[52] Liang Zou, Naijun Zhan, Shuling Wang, and Martin Fränzle. 2015. Formal Verification of Simulink/Stateflow Diagrams. In *Automated Technology for Verification and Analysis - 13th International Symposium, Shanghai, China*. 464–481.