Statistical Verification of Hyperproperties for Cyber-Physical Systems*

YU WANG, Duke University
MOJTABA ZAREI, Duke University
BORZOO BONAKDARPOUR, Iowa State University
MIROSLAV PAJIC, Duke University

Many important properties of cyber-physical systems (CPS) are defined upon the relationship between multiple executions simultaneously in continuous time. Examples include probabilistic fairness and sensitivity to modeling errors (i.e., parameters changes) for real-valued signals. These requirements can only be specified by *hyperproperties*. In this article, we focus on verifying probabilistic hyperproperties for CPS. To cover a wide range of modeling formalisms, we first propose a general model of probabilistic uncertain systems (PUSs) that unify commonly studied CPS models such as *continuous-time Markov chains* (CTMCs) and *probabilistically parametrized Hybrid I/O Automata* (P^2HIOA). To formally specify hyperproperties, we propose a new temporal logic, *hyper probabilistic signal temporal logic* (HyperPSTL) that serves as a hyper and probabilistic version of the conventional signal temporal logic (STL). Considering the complexity of real-world systems that can be captured as PUSs, we adopt a *statistical model checking* (SMC) approach for their verification. We develop a new SMC technique based on the direct computation of significance levels of statistical assertions for HyperPSTL specifications, which requires no *a priori* knowledge on the indifference margin. Then, we introduce SMC algorithms for HyperPSTL specifications on the joint probabilistic distribution of multiple paths, as well as specifications with nested probabilistic operators quantifying different paths, which cannot be handled by existing SMC algorithms. Finally, we show the effectiveness of our SMC algorithms on CPS benchmarks with varying levels of complexity, including the Toyota Powertrain Control System.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; • **Theory of computation → Logic and verification**; • **Security and privacy → Formal security models**; • **Mathematics of computing** → *Hypothesis testing and confidence interval computation*; • **Software and its engineering** → *Software verification and validation*;

Additional Key Words and Phrases: Cyber-physical systems, hyperproperties, statistical model checking, embedded control software.

ACM Reference Format:
Yu Wang, Mojtaba Zarei, Borzoo Bonakdarpour, and Miroslav Pajic. 2019. Statistical Verification of Hyperproperties for Cyber-Physical Systems. *ACM Trans. Embedd. Comput. Syst.* 1, 1 (August 2019), 23 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*This article appears as part of the ESWEEK-TECS special issue and was presented at the International Conference on Embedded Software (EMSOFT) 2019.

Authors' addresses: Yu Wang, Duke University, 100 Science Dr, Hudson Hall Rm 220, Durham, NC, 27708, yu.wang094@duke.edu; Mojtaba Zarei, Duke University, 100 Science Dr, Hudson Hall Rm 220, Durham, NC, 27708, mojtaba.zarei@duke.edu; Borzoo Bonakdarpour, Iowa State University, 207 Atanasoff Hall, Ames, IA, 50011, borzoo@iastate.edu; Miroslav Pajic, Duke University, 100 Science Dr, Hudson Hall Rm 130, Durham, NC, 27708, miroslav.pajic@duke.edu.

1539-9087/2019/8-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Ensuring safety of controllers in embedded and cyber-physical systems (CPS) using closed-loop system verification is a challenging problem, due to the inherent uncertainties in system dynamics and the environment. In systems where the uncertainties can be captured in a probabilistic manner, two prominent verification approaches are *exhaustive* [3] and *statistical* [21]. The exhaustive approach, with full knowledge of a system model, computes the satisfying probability of the desired properties arithmetically; on the other hand, the statistical approach estimates the probabilities from sampling, and makes assertions with a certain significance level (an upper bound of the probability of returning a wrong answer). Accordingly, *statistical model checking* (SMC) is more capable of handling "black-box", high-dimension or large-scale system models.

Properties of interest for such systems, from 'simple' probabilistic safety or reachability, to the ones that capture dynamical responses under complex conditions, are usually formally defined by probabilistic temporal logic specifications. Conventional probabilistic temporal logics, such as the probabilistic computational tree logic (PCTL) [15], as well as its extension PCTL* [3], can only specify probabilistic properties without explicitly quantifying over different paths of the system; that is, they cannot simultaneously and explicitly quantify over multiple distinctive paths to fully express their inter-relations. This prevents them from capturing important safety/performance hyperproperties [2, 7] that involve multiple execution paths, such as *sensitivity* to model errors, *detectability* of system anomalies, and *fairness* when more than once process/client are controlled/serviced.

For example, consider embedded controllers such as the Toyota Powertrain control system benchmark [19], for which both exhaustive (e.g., [11]) and statistical (e.g., [24]) verification techniques have been introduced. However, all these techniques are restricted to the use on a dynamical system model obtained for fixed system parameters. In general, such parameters are experimentally derived and thus, should be considered as random variables with unknown probability distributions, instead of the fixed values. In addition, some of the system parameters might change to a degree, due to system 'wear-and-tear'. Hence, it is critical to enable analysis of system sensitivity to model errors, by providing a formal logic to capture such properties, as well as methods to verify how system evolution (execution) changes for different parameters of the system model.

Specifying these properties, such as sensitivity to change of parameters, involves probabilistic quantification over multiple paths (a path and its deviation), and thus can only be captured as *hyperproperties* [7]. For example, as illustrated in Figure 1, for sensitivity analysis we can check whether the deviation π_2 of a path π_1 under probabilistic uncertainty stays close probabilistically, such that there is limited variation in the hitting time τ to a desired working region. Although the sensitivity may be expressed as a non-hyperproperty if 'expected' hitting times are known in advance (as a reference) for any entry into the desired operating region, such information is usually unavailable for complex systems.

Consequently, in this work, we first introduce a probabilistic temporal logic for hyperproperties expressed on real-valued continuous-time signals, referred to as *Hyper Probabilistic Signal Temporal Logic* (HyperPSTL). HyperPSTL can be viewed as a hyper extension and generalization of the probabilistic signal/metric-interval temporal logic [25, 32], a probabilistic version of HyperSTL [23], and a continuous-time extension and generalization of HyperPCTL [2]. HyperPSTL extends those logics by enabling (1) reasoning about the probability of paths by adding a probability operator, and (2) reasoning about multiple paths simultaneously, i.e., hyperproperties specifying the relationship between different paths by associating path variables to the atomic propositions.

To allow us to cover a range of modeling formalisms, we introduce a very general system model – *probabilistic uncertain systems* (PUS) – and define the semantics of HyperPSTL on it. Generally,

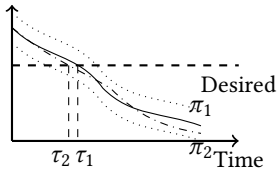

Fig. 1. Sensitivity of paths π_1 and π_2, drawn in solid and dashed lines respectively.

they are 'black-box' probabilistic dynamical systems with unknown dynamics on a given state space. A PUS incorporates nondeterminism as its input and probabilism as its parameters, both of which are time functions of values of general types, including real, integer or categorical/Boolean. Given the values of the input and the parameters, we can draw a time-dependent sample path from the PUS, which can also be of general types. We note that this general model captures commonly studied models such as CTMCs and hybrid I/O automata with probabilistic parameters – referred to as probabilistically parameterized hybrid I/O automata (P^2HIOA).

We define the semantics of HyperPSTL on a set of paths of the PUS through pre-defined labels or predicates on the state spaces. We show by concrete examples that this allows for the capturing of properties such as (i) anomaly detectability, or (ii) sensitivity to model errors due to probabilistic uncertainty of parameters of hybrid I/O automata, and (iii) workload fairness in queueing networks modeled as continuous-time Markov chains (CTMC). To verify HyperPSTL specifications on a PUS for a given input, we introduce new SMC algorithms using Clopper-Pearson (CP) bounds [8].

Unlike most previous SMC methods based on the sequential probability ratio test (SPRT) [20, 21], this approach requires no *a priori* knowledge on the indifference margin. Our SMC algorithm can verify a HyperPSTL specification to arbitrarily small (non-zero) significance levels, which is also different from [4] on using CP bounds to estimate satisfying probability for given samples. The conservativeness of the CP significance level ensures that the desired significance level is strictly achieved even in the worst case. Specifically, we iteratively draw samples from the objective model and compute the significance level derived from the CP confidence interval. The algorithm stops if the CP significance level is smaller than the desired significance levels. Further improvement in sample efficiency is also possible using the upper bounds derived from Wilson scores, Jeffreys interval or Agresti-Coull interval, at the cost of yielding only asymptotic correctness (see [6] for a review of these confidence intervals).

To the best of our knowledge, this work is the first to enable SMC on probabilistic temporal logic for hyperproperties in continuous time. Compared to common non-hyper temporal logics, HyperPSTL allows uniquely for (1) probability quantification over multiple paths, (2) defining specifications on their joint probabilistic distribution, such as comparison of probabilities, and (3) nesting of probabilistic operators that quantify over different sets of paths. We address these challenges by deriving SMC methods capable of handling these cases, while providing provable significance levels. We also show that HyperPSTL subsumes several existing signal temporal logics for probabilistic properties [24, 32] (see Section 3), and thus, the SMC algorithms for HyperPSTL introduced in this work, also apply to them.

Finally, we show the effectiveness of our SMC methods on embedded case studies with different complexity and modeling formalism. Specifically, we statistically verified sensitivity of a thermostat and the Toyota Powertrain System with uncertain parameters, as well as fairness in queueing networks of different sizes; we generated probabilistic guarantees with high significance levels. For example, for the Toyota Powertrain and the large queueing networks, where exhaustive verification is not possible, we construct upper bounds on the 0.95 and 0.99 percentiles of their sensitivity

and fairness, respectively, with a confidence level of 0.99 using only a few hundred samples. This achieves, for the first time, statistical verification of probabilistic hyperproperties on continuous signals, which can be used even for real-world sized CPS.

Organization. We introduce probabilistic uncertain systems in Section 2, followed by the syntax and semantics of HyperPSTL in Section 3. In Section 4, we show how HyperPSTL can be used to capture desired properties of CPS. Our SMC algorithms are presented in Section 5 with an emphasis on the use of CP significance levels and the handling of probabilities involving multiple paths. We apply the SMC techniques to three embedded case studies (e.g., Toyota Powertrain System) in Section 6, before concluding in Section 7.

Notation. We denote the sets of integers, rational, real, and non-negative real numbers by $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{R}_{\geq 0}$, respectively. The domain and image of a function is denoted by $\text{Dom}(\cdot)$ and $\text{Image}(\cdot)$, respectively. Let $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$ and $\mathbb{Q}_\infty = \mathbb{Q} \cup \{\infty\}$. For $n \in \mathbb{N}$, let $[n] = \{1, \ldots, n\}$. The indicator function is denoted by $\mathbf{I}$. We denote the (Borel) measure of a measurable set by $\mu_{\text{Borel}}(\cdot)$. The cardinality and the power set of a set are denoted by $|\cdot|$ and $2^{\cdot}$. For any set $I \subseteq \mathbb{R}^n$, we denote its boundary, interior, and closure by ∂I, I°, and $\bar{I}$, respectively. The empty set is denoted by $\emptyset$. With a slight abuse of notation, given a map $V : A \to B$ and $A' \subseteq A$, let $V(A') = \cup_{a \in A'} V(a)$.

We refer to a function of time $\sigma : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ as a *signal*, and denote by $\sigma^{(t_1)}$ its t_1-time shift defined as $\sigma^{(t_1)}(t_2) = \sigma(t_1 + t_2)$ for $t_1 \in \mathbb{R}_{\geq 0}$. We denote the binomial distribution by $\text{Binom}(n, p)$, the exponential distribution parametrized by rate by $\text{Exp}(\lambda)$, and the beta distributions with shape parameters by $\text{Beta}(\alpha, \beta)$. A random variable X drawn from probability distribution μ is denoted by $X \sim \mu$.

2 PROBABILISTIC UNCERTAIN SYSTEMS

To allow for defining HyperPSTL for a large class of commonly used modeling formalisms in a way that facilitates design of SMC techniques, we introduce a very general system model, which we refer to as *probabilistic uncertain systems* (PUS) (Figure 2). Such model can be viewed as a "black-box" probabilistic dynamical system with an explicit state space, where the randomness only comes from the time-dependent parameters drawn from random processes. The PUS generalizes models such as CTMCs and P^{2}HIOA, and provides a unified framework for the SMC of probabilistic hyperproperties.

A PUS is a 'black-box' probabilistic dynamical system on a given *state space*, which we denote by $\mathcal{X}$. Its probabilistic uncertainty comes from a set of *n time-dependent* parameters $D(t) = \Big(d_1(t), ..., d_n(t)\Big)$ where $t \in \mathbb{R}_{\geq 0}$, drawn from some probability distribution $\mu(\mathcal{D})$ on its domain $\mathcal{D}$. That is, the *parameters* of the system are drawn from an n-dimensional random process. The *input* to the system $I(t) = \Big(i_1(t), \ldots, i_m(t)\Big) \in \mathcal{I}$ is an m-dimensional function of time t. Given the input $I(t)$, the values of the parameters $D(t)$, and an initial state $X^{\text{init}} = \Big(x_1^{\text{init}}, \ldots, x_l^{\text{init}}\Big) \in \mathcal{X}$, the system deterministically generates a time-dependent *path* $X : \mathbb{R}_{\geq 0} \to \mathcal{X}$ with $X(t) = \Big(x_1(t), \ldots, x_l(t)\Big)$. That is, the randomness in system evolution only comes from system parameters.

The values of the system inputs, parameters, and states can be of mixed types: real, discrete or categorical, depending on the system formulation. Here, we make no assumption on the system dynamics (Markovian, causal, etc); rather, the system should be viewed just as a general deterministic map from the functions $I(t)$ and $D(t)$ to the function $X(t)$. Without knowing the value of the system parameters and given the initial state, the map from the input $I(t)$ to the path $X(t)$ is only probabilistic, i.e., the probability distribution for all uncertain variables are given.

Given the finite set of *atomic propositions* AP, the labeling function on the state space $\mathsf{L} : \mathcal{X} \to 2^{\mathsf{AP}}$ defines for each state, a set of atomic propositions that hold. Alternatively, the labeling of each

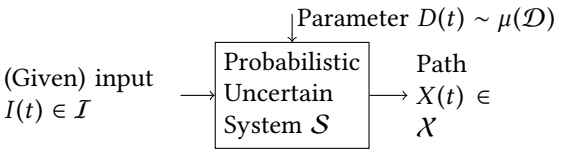

Fig. 2. Probabilistic Uncertain System $\mathcal{S}$.

atomic proposition in AP can be represented by a predicate/Boolean function that indicates the subset of states $\mathcal{X}$ that are labeled by that atomic proposition. Then, a path of the system induces a signal $\sigma(t) = \mathsf{L}(X(t)) : \mathbb{R}_{\geq 0} \to 2^{\mathsf{AP}}$. This signal indicates the set of properties that is satisfied by the system $\mathcal{S}$ at each time $t \in \mathbb{R}_{\geq 0}$ during system evolution. In Section 3, we will define hyper temporal properties on sets of such signals.

A PUS $\mathcal{S}$ is both probabilistic in the value of its parameters and nondeterministic in its input. We note that the distinction between the parameters and the input of the system is only mathematical, not physical. In practice, we can model the probabilistic input of a real system as a parameter of its PUS model, and a nondeterministic parameter of the system as an input of the PUS. In this work, we focus on SMC related to the probabilistic nature of a PUS; i.e., we consider the cases, where the input of the PUS is given such as when it is determined by the actions of the controller used to control the system (PUS). Before we introduce HyperPSTL and SMC methods for PUS, in the rest of this section, we show how our abstract notion of PUS captures two prominent computing models: *continuous-time Markov chains* (CTMCs) that mathematically model queuing networks and probabilistic hybrid I/O automata that model the Toyota Powertrain system mentioned in Section 1.

2.1 Probabilistically Parameterized Hybrid I/O Automata

Probabilistically Parameterized Hybrid I/O Automata (P^{2}HIOA) are extensions of Hybrid I/O Automata (HIOA) [16, 19, 22] with probabilistic system parameters. Given the variety of mathematical models for HIOA, we build the one used in [19] to describe the dynamics of hybrid systems with fully observable states. Specifically, an HIOA is a tuple $\mathcal{A} = (\mathtt{M}_{\mathcal{A}}, \mathtt{X}_{\mathcal{A}}, \mathtt{I}_{\mathcal{A}}, \mathtt{U}_{\mathcal{A}}, \mathtt{Inv}_{\mathcal{A}}, \mathtt{T}_{\mathcal{A}}, \mathtt{Q})$ where

- $\mathtt{M}_{\mathcal{A}}$ is a finite set of *modes*;
- $\mathtt{X}_{\mathcal{A}}$ is a set of *n state variables* of real values, i.e., $\mathtt{X}_{\mathcal{A}} \subseteq \mathbb{R}^n$;
- $\mathtt{U}_{\mathcal{A}}$ is a set of *m input variables*; their valuations can be of different types, such as $\mathbb{N}$, $\mathbb{R}$, and Boolean;
- $\mathtt{F}_{\mathcal{A}} : \mathtt{M}_{\mathcal{A}} \times \mathtt{X}_{\mathcal{A}} \times \mathtt{U}_{\mathcal{A}} \to \mathbb{R}^n$ defines a deterministic *flow* capturing system evolution in each mode, i.e., a differential equation:
$$\frac{\mathrm{d}\mathtt{x}_{\mathcal{A}}}{\mathrm{d}t} = \mathtt{F}_{\mathcal{A}}\left(\mathtt{m}_{\mathcal{A}}, \mathtt{x}_{\mathcal{A}}, \mathtt{u}_{\mathcal{A}}\right);$$
- For any mode $\mathtt{m}_{\mathcal{A}} \in \mathtt{M}_{\mathcal{A}}$, $\mathtt{Inv}_{\mathcal{A}}(\mathtt{m}_{\mathcal{A}}) = \mathrm{Dom}(\mathtt{F}_{\mathcal{A}}(\mathtt{m}_{\mathcal{A}}, \cdot, \cdot))$ defines the *invariant* of the mode;
- $\mathtt{T}_{\mathcal{A}} : \mathtt{M}_{\mathcal{A}} \times \mathtt{X}_{\mathcal{A}} \times \mathtt{U}_{\mathcal{A}} \to \mathtt{M}_{\mathcal{A}} \times \mathtt{X}_{\mathcal{A}}$ defines deterministic *jumps* triggered by $\mathtt{X}_{\mathcal{A}} \in \partial\mathtt{Inv}_{\mathcal{A}}(\mathtt{m}_{\mathcal{A}})$, i.e., when the flow hits the boundary of the invariant.
- $\mathtt{I}_{\mathcal{A}} \in \mathtt{M}_{\mathcal{A}} \times \mathtt{X}_{\mathcal{A}}$ is the *initial* condition;
- $\mathtt{Q} \subseteq \mathtt{U}_{\mathcal{A}} \times \mathtt{X}_{\mathcal{A}} \to \{0, 1\}$ is a finite set of *predicates*.

The P^{2}HIOA extend Hybrid I/O Automata by allowing system parameters, used to capture $\mathtt{F}_{\mathcal{A}}, \mathtt{T}_{\mathcal{A}}, \mathtt{Inv}_{\mathcal{A}}$, to be probabilistic instead of fixed. This differs from the Probabilistic Hybrid Automata introduced in [28, 34], where the randomness comes only from the probabilistic jumps. In standard hybrid models, the deterministic and nonlinear flow, invariant, and jump functions $\mathtt{F}_{\mathcal{A}}, \mathtt{T}_{\mathcal{A}}, \mathtt{Inv}_{\mathcal{A}}$, which define its dynamics, are parameterized by quantities that are estimated from

physical experiments. For example, in the Toyota Powertrain model, the mass flow rate of intake air is determined by the RPM of the engine and the pressure of the intake manifold through a polynomial, whose five parameters are fit from experimental data. Due to experimental errors, these parameters are better represented as random variables with unknown probability distributions (e.g., Gaussian or uniform with means and variances inferred from experimental data) than real numbers as in [19]. Thus, we denote the parameters for $\mathtt{F}_{\mathcal{A}}$, $\mathtt{T}_{\mathcal{A}}$, and $\mathtt{Inv}_{\mathcal{A}}$ by $D = (d_{\mathtt{F}_{\mathcal{A}}}, d_{\mathtt{T}_{\mathcal{A}}}, d_{\mathtt{Inv}_{\mathcal{A}}})$. To simplify our presentation, in this work, we assume that these parameters for P^{2}HIOA are time-invariant.

Consequently, a P^{2}HIOA can be represented by a PUS by treating (i) $I = \mathtt{U}_{\mathcal{A}}$ as the input, (ii) $X = \mathtt{M}_{\mathcal{A}} \times \mathtt{X}_{\mathcal{A}}$ as the state (the input variables $\mathtt{U}_{\mathcal{A}}$ are encoded as part of the state), (iii) $X^{\text{init}} = \mathtt{I}_{\mathcal{A}}$ as the initial state, (iv) $D = (d_{\mathtt{F}_{\mathcal{A}}}, d_{\mathtt{T}_{\mathcal{A}}}, d_{\mathtt{Inv}_{\mathcal{A}}})$ as the parameters, and (v) the predicate $\mathtt{Q}$ as the labeling function. The probabilistic hyperproperties related to P^{2}HIOA, as the one discussed in Sections 1 and 4, will be statistically verified on this PUS.

2.2 Continuous-Time Markov Chains

Another example of PUS are CTMCs, which are commonly used to model queuing and task scheduling in embedded computing and communication systems with uncertainties. Consider a CTMC with

- the *states* $[n]$,
- the *initial state* $X_0 \in [n]$,
- the *labeling function* on the states $\mathtt{L} : [n] \to 2^{\mathsf{AP}}$ for a given set of labels AP,
- the *probability transition rate matrix* $M \in \mathbb{R}^{n \times n}$, such that $\sum_{j \in [n]} M_{ij} = 0$, where M_{ij} is the transition rate from a state i to a state j.

The CTMC can be represented by a PUS with (i) the states $\mathcal{X} = [n]$, (ii) the initial state X_0, (iii) the empty inputs (i.e., the PUS has no input), (iv) the labels AP, and (v) the labeling function $\mathtt{L} : [n] \to 2^{\mathsf{AP}}$. For $i \in \mathbb{N}$, we draw d_{1i} and d_{2i} from $[0, 1]$ uniformly and independently. Using Gillespie algorithm [14], with the convention $\sum_{k=1}^{0} \cdot = 0$, the state X of the PUS representing the CTMC evolves by:

$$X(t) = X_i \;\text{ if }\; t \in [T_i, T_{i+1}),$$

where the state jumps are determined by d_{1i} as

$$X_{i+1} = j \text{ when } X_i = l \;\text{ if }\; \sum_{k=1}^{j-1} M_{lk} \leq d_{1i} < \sum_{k=1}^{j} M_{lk},$$

and the time lapses are determined by d_{2i} using

$$T_{i+1} = T_i + \mathbf{F}^{-1}_{\text{Exp}\left(\sum_{k \in [n], k \neq l} M_{lk}\right)}\left(d_{2i}\right),\; T_0 = 0,$$

with $\text{Exp}(\cdot)$ denoting the exponential distribution parameterized by rate and $\mathbf{F}^{-1}_{\cdot}$ the inverse function of the cumulative distribution function. Treating the two sequences $\{d_{1i}\}_{i \in \mathbb{N}}$ and $\{d_{2i}\}_{i \in \mathbb{N}}$ as the two parameters of the PUS, provides a PUS presentation of the CTMC. Although they are discrete sequences, we can easily represent them as continuous functions of time to fit into the definition of the parameters of the PUS. Then, the probabilistic hyperproperties (e.g., fairness) for the initial CTMC, should be statistically verified on the aforementioned PUS.

3 HYPER PROBABILISTIC SIGNAL-TEMPORAL LOGIC

To formally express and reason about probabilistic hyperproperties on real-valued signals, we introduce the logic HyperPSTL, which can be viewed as a probabilistic extension of the *signal temporal logic* (STL) for hyperproperties. We introduce the syntax and semantics of HyperPSTL for

PUS in Sections 3.1 and 3.2, before presenting its use to capture relevant properties of embedded systems in Section 4.

3.1 Syntax

We define HyperPSTL formulas inductively as:

$$\varphi ::= \mathsf{a}^{\pi} \mid \varphi^{\pi} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \, \mathcal{U}_{[t_1,t_2]} \, \varphi \mid p \bowtie p \tag{1}$$

$$p ::= \mathbb{P}^{\Pi}\varphi \mid \mathbb{P}^{\Pi} p \mid f(p, \ldots, p) \tag{2}$$

where

- $\mathsf{a} \in \mathsf{AP}$, and AP is the finite set of *atomic propositions*,
- $t_1 < t_2$ with $t_1, t_2 \in \mathbb{Q}_{\infty}$,
- π is a path variable, and Π is a set of path variables,
- $\mathbb{P}$ is the probability operator,
- $\bowtie \in \{<, >, =, \leq, \geq\}$,
- $f : \mathbb{R}^n \to \mathbb{R}$ is a n-ary measurable function, constants are viewed as 0-ary functions,
- $\mathsf{fv}(\cdot)$ denotes the set of free path variables in φ, i.e., the path variables not quantified by a probability operator through (2), and $\mathsf{fv}(\varphi) = \emptyset$ in the 2nd rule of (1). This is recursively defined by:

$$\mathsf{fv}(\mathsf{a}^{\pi}) = \{\pi\},\ \mathsf{fv}(\varphi^{\pi}) = \{\pi\},\ \mathsf{fv}(\neg\varphi) = \mathsf{fv}(\varphi),$$
$$\mathsf{fv}(\varphi_1 \wedge \varphi_2) = \mathsf{fv}(\varphi_1) \cup \mathsf{fv}(\varphi_2),\ \mathsf{fv}(\varphi_1 \, \mathcal{U}_{[t_1,t_2]} \, \varphi_2) = \mathsf{fv}(\varphi_1) \cup \mathsf{fv}(\varphi_2),$$
$$\mathsf{fv}(p_1 \bowtie p_2) = \mathsf{fv}(p_1) \cup \mathsf{fv}(p_2),\ \mathsf{fv}(f(p_1, \ldots, p_n)) = \textstyle\bigcup_{i \in [n]} \mathsf{fv}(p_i),$$
$$\mathsf{fv}(\mathbb{P}^{\Pi}(\varphi)) = \mathsf{fv}(\varphi) \backslash \Pi,\ \mathsf{fv}(\mathbb{P}^{\Pi}(p)) = \mathsf{fv}(p) \backslash \Pi.$$

Other common logic operators can be derived as follows: $\varphi \vee \varphi' \equiv \neg(\neg\varphi \wedge \neg\varphi')$, $\mathtt{True} \equiv \varphi \vee \neg\varphi$, $\varphi \Rightarrow \varphi' \equiv \neg\varphi \vee \varphi'$, $\Diamond_{[t_1,t_2]}\varphi \equiv \mathtt{True} \ \mathcal{U}_{[t_1,t_2]} \ \varphi$, and $\Box_{[t_1,t_2]}\varphi \equiv \neg\Diamond_{[t_1,t_2]}\neg\varphi$. In addition, we denote $\mathcal{U}_{[0,\infty)}$, $\Diamond_{[0,\infty)}$, and $\Box_{[0,\infty)}$ by $\mathcal{U}$, $\Diamond$, $\Box$, respectively.

A formula with $\mathsf{fv}(\varphi) = \emptyset$ is called a *state formula* and requires no instantiation of free path variables, thus can be evaluated on a state of the PUS. Therefore, we can associate another path variable π to it. All other formulas are referred to as *path formulas*, as their correctness depends on the instantiation of their free path variables. It specifies on which path a state formula should be satisfied, so that a HyperPSTL formula can reason simultaneously on multiple paths. Finally, for the first two rules of (2), we assume that Π is contained in $\mathsf{fv}(\varphi)$ or $\mathsf{fv}(p)$ for the probability quantification to be non-trivial. Observe that HyperPSTL can be viewed as the probabilistic version of HyperSTL [23] by replacing the existential and universal quantifiers over signals with probabilistic quantifiers over paths in (2).

HyperPSTL has the following unique features. It allows for the simultaneous probability quantification over several paths (as we show for the sensitivity analysis of powertrain controllers in Section 4). It also allows for the arithmetics and comparison of probabilities, and the nesting of probability operators quantifying different paths (as shown in the queueing fairness analysis described in Section 4).

HyperPSTL reduces to a non-hyper probabilistic signal temporal logic (PSTL) if it only has one path variable in it. PSTL can still define probability satisfaction of single atomic propositions, so it subsumes the MITL for probability distributions from [32]. But, PSTL (and thus HyperPSTL) does not subsume PrSTL in [25], since it does not allow a time-varying probability threshold, as is allowed in PrSTL. Still, augmenting HyperPSTL syntax to allow time-varying functions is straight-forward.

Finally, note that to simplify our presentation of HyperPSTL syntax and semantics, while allowing for verification of complex systems such as P^{2}HIOA, we only include simultaneous or consecutive probabilistic quantification (e.g., $\mathbb{P}^{\{\pi_1,\pi_2\}}$ or $\mathbb{P}^{\pi_1}\mathbb{P}^{\pi_2}$) over the paths from a single initial state. HyperPSTL can be augmented by allowing nested existential and universal quantification over multiple states in the same way as [2, 31]. Specifically, in addition to the probabilistic quantification over the paths, one can add extra state quantification of these paths to specify from which state the path starts, like $\exists X_1^{\pi_1}.\forall X_2^{\pi_2}.\mathbb{P}^{\{\pi_1,\pi_2\}}$. However, verifying such formulas generally requires exhaustive iteration over all the states, which is challenging, if not impossible, on systems with infinite state spaces like P^{2}HIOA. Therefore, in this paper, we do not include state quantification which in our logic can be done as presented in [2, 31].

3.2 Semantics

We define the satisfaction relation for HyperPSTL state formulas on a PUS $\mathcal{S}$ by

$$(\mathcal{S}, X) \models \varphi \quad \Leftrightarrow \quad \mathcal{S} \models [\![\varphi]\!]_{V_X}, \tag{3}$$

where V_X is an assignment of path variables to the paths of the PUS $\mathcal{S}$ starting from a state X, and $[\![\varphi]\!]_{V_X}$ is the instantiation of the assignment V_X on φ. Here, the instantiation specifies the initial state for the formula φ.

The satisfaction relation for the HyperPSTL path formulas is defined with respect to the assignment V_X by:

$$\begin{array}{lcl}
\mathcal{S} \models [\![p \bowtie p]\!]_{V_X} & \Leftrightarrow & \mathcal{S} \models [\![p]\!]_{V_X} \bowtie [\![p]\!]_{V_X} \\
\mathcal{S} \models [\![f(p,\ldots,p)]\!]_{V_X} & \Leftrightarrow & \mathcal{S} \models f\Big([\![p]\!]_{V_X},\ldots,[\![p]\!]_{V_X}\Big) \\
\mathcal{S} \models [\![\mathbb{P}^{\Pi}(\varphi)]\!]_{V_X} & \Leftrightarrow & \mathcal{S} \models \mathbf{Pr}_{\boldsymbol{\sigma}\sim \mathrm{Path}^{|\Pi|}(X)}\Big(\big(\mathcal{S}, V_X[\Pi \to \boldsymbol{\sigma}]\big) \models \varphi\Big) \\
(\mathcal{S}, V_X) \models \mathsf{a}^{\pi} & \Leftrightarrow & \mathsf{a} \in \mathsf{L}(V_X(\pi)(0)) \\
(\mathcal{S}, V_X) \models \Phi^{\pi} & \Leftrightarrow & (\mathcal{S}, V_X(\pi)) \models \Phi \\
(\mathcal{S}, V_X) \models \neg\varphi & \Leftrightarrow & (\mathcal{S}, V_X) \not\models \varphi \\
(\mathcal{S}, V_X) \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow & (\mathcal{S}, V_X) \models \varphi_1 \text{ and } (\mathcal{S}, V_X) \models \varphi_2 \\
(\mathcal{S}, V_X) \models \varphi_1\, \mathcal{U}_{[t_1,t_2]}\, \varphi_2 & \Leftrightarrow & \exists t \in [t_1,t_2].\ \Big(\forall t' < t.\Big(\mathcal{S}, V_X^{(t')}\Big) \models \varphi_1\Big) \wedge \Big(\mathcal{S}, V_X^{(t)}\Big) \models \varphi_2
\end{array} \tag{4}$$

where

- $\mathsf{L}(X)$ is the set of labels of the PUS state X,
- $\mathrm{Path}^{|\Pi|}(X)$ is the collection of all $|\Pi|$-tuples of paths starting from state X,
- $V_X[\Pi \to \boldsymbol{\sigma}]$ is a revision of the assignment V_X by assigning the set $\boldsymbol{\sigma}$ of paths to the set Π of path variables, respectively,
- $V_X^{(t)}$ is the t-shift of the assignment V_X, defined by $\Big(V_X^{(t)}(\pi)\Big) = (V_X(\pi))^{(t)}$ for all path variables π in assignment V.

Finally, we note the equivalence $\mathsf{a} \in \mathsf{L}(V_X(\pi)(0)) \Leftrightarrow \mathsf{a} \in \mathsf{L}(X)$ and $(\mathcal{S}, V_X(\pi)) \models \Phi \Leftrightarrow (\mathcal{S}, X) \models \Phi$ for the 4$^{\text{th}}$ and 5$^{\text{th}}$ rules in (4). The following example illustrates the semantics of HyperPSTL.

Example 1. *As shown in Figure 3, consider a CTMC $\mathcal{S}$ that models a queue that is initially empty and is of buffer size 2. The transition rate matrix of $\mathcal{S}$ is the following:*

$$\begin{bmatrix} -1 & 1 & 0 \\ 2 & -3 & 1 \\ 0 & 2 & -2 \end{bmatrix}$$

The CTMC satisfies the following HyperPSTL formula:

$$\varphi = \mathbb{P}^{\pi_1}\Big((\neg s_1^{\pi_1})\ \mathcal{U}\ (s_1^{\pi_1}\ \mathcal{U}_{[0,1]}\ s_0^{\pi_1})\Big) - \mathbb{P}^{\pi_2}\Big((\neg s_2^{\pi_2})\ \mathcal{U}\ (s_2^{\pi_2}\ \mathcal{U}_{[0,1]}\ s_1^{\pi_2})\Big) > 0.05,$$

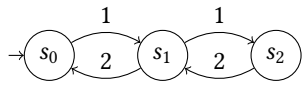

Fig. 3. CTMC Model of A Queue HyperPSTL.

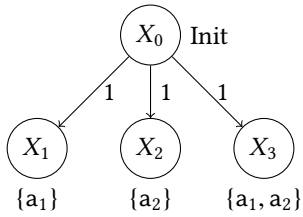

Fig. 4. HyperPSTL on CTMC.

The formula asserts that the probability difference between (i) finishing a task within 1 *time delay after first having* 1 *in queue and (ii) finishing a task within* 1 *time delay after first having* 2 *in queue, is greater than* 0.05. *The is because the probability for (i) is* $\frac{2}{3}(1 - e^{-3}) \approx 0.633$ *and that for (ii) is* $1 - e^{-2} \approx 0.865$. ◁

3.3 Expressivity of HyperPSTL

Theorem 3.1. *HyperPSTL subsumes PSTL on CTMCs.*

Proof. It suffices to show that there exist formulas in HyperPSTL that cannot be expressed in PSTL. The general idea is to show that PSTL cannot express conditional probabilities, while HyperPSTL can. Consider the CTMC $\mathcal{S}$ given in Figure 4, and the HyperPSTL state formula

$$\varphi = \Big(\frac{\mathbb{P}^{\pi}\left(\text{Init}^{\pi} \Rightarrow \Diamond(\text{a}_1^{\pi} \wedge \text{a}_2^{\pi})\right)}{\mathbb{P}^{\pi}\left(\text{Init}^{\pi} \Rightarrow \Diamond \text{a}_2^{\pi}\right)} = \frac{1}{2} \Big),$$

expressing a conditional probability for a path π of $\mathcal{S}$. Clearly, it is satisfied for the state Init.

We claim that φ cannot be expressed in PSTL. By the syntax and semantics of PSTL [25], it suffices to show that φ cannot be expressed by a formula $\mathbb{P}(\psi)$, where ψ is a PSTL path formula derived by concatenating a set of PSTL state formulas $\varphi_1, \ldots, \varphi_n$ with $\wedge$, $\neg$, or temporal operators. These state formulas are either `True` or `False` on the states X_0, X_1, X_2, and X_3. Thus, whether a path satisfies ψ, defines a subset of the paths from the state X_0 in the CTMC, Since the probability of a path ended up in any X_i is $1/3$ for $i = 1, 2, 3$, the formula $\mathbb{P}(\psi)$ can only take values in $\{0, 1/3, 2/3, 1\}$. However, by the semantics of HyperPSTL, the fractional probability on the right side of the equation has value $1/2$, thus φ evaluates to true and cannot be expressed by $\mathbb{P}(\psi)$ in PSTL. □

4 HyperPSTL IN ACTION

In this section, we demonstrate how HyperPSTL can be used to capture relevant properties of CPS.

4.1 Sensitivity to Modeling Errors in P^{2}HIOA

A typical example of a CPS that can be modeled as a P^{2}HIOA is the automotive powertrain, where the response to the change in driving behaviors is of key interest. Note that the dynamical system parameters, even for the same type of powertrains, vary across different systems. Thus, it is critical to analyze if *in most cases*, the change in dynamical response of the controlled system stays within

permitted amount δ when the system parameters change; i.e., if the dynamical response deviation is within δ with probability of at least $1-\varepsilon$.

Consider as an example, the sensitivity of the Toyota Powertrain Controller [19] under probabilistic uncertainty in its dynamical parameters (i.e., the system model). As shown in Figure 1, we consider statistically verifying the probabilistic boundedness of the sensitivity of the (first) hitting time τ to a desired working region (where the error to the desired AF ratio is less than 5%) under the probabilistic uncertainty in RPM. Mathematically, this can be represented by

$$\mathbf{Pr}_{\pi_1,\pi_2}\Big(|\tau^{\pi_1}-\tau^{\pi_2}|\leq\delta\Big)>1-\varepsilon,$$

for some given values $\delta,\varepsilon>0$, where π_1 and π_2 are two statistically independent sample paths of the system. To express this formally using HyperPSTL, we introduce a predicate $\mathtt{Q}$ for the desired working region of the system (i.e., under the dashed line in Figure 1). If $0\leq\tau^{\pi_2}-\tau^{\pi_1}\leq\delta$, the switch from $\neg\mathtt{Q}^{\pi_2}$ to $\mathtt{Q}^{\pi_2}$ for the path π_2 happens within time δ after $\neg\mathtt{Q}^{\pi_1}$ changes to $\mathtt{Q}^{\pi_1}$ for the path π_1.[1] This can be equivalently expressed as $(\neg\mathtt{Q}^{\pi_1}\wedge\neg\mathtt{Q}^{\pi_2})\,\mathcal{U}\,(\mathtt{Q}^{\pi_1}\wedge\Diamond_{[0,\delta]}\mathtt{Q}^{\pi_2})$, and accordingly, the probabilistic boundedness of sensitivity is expressed as

$$\mathbb{P}^{\{\pi_1,\pi_2\}}\Big((\neg\mathtt{Q}^{\pi_1}\wedge\neg\mathtt{Q}^{\pi_2})\;\mathcal{U}\;\big((\mathtt{Q}^{\pi_1}\wedge\Diamond_{[0,\delta]}\mathtt{Q}^{\pi_2})\vee(\mathtt{Q}^{\pi_2}\wedge\Diamond_{[0,\delta]}\mathtt{Q}^{\pi_1})\big)\Big)\geq 1-\varepsilon. \tag{5}$$

Note that (5) involves probability quantification over a set of paths, which cannot be expressed in non-hyper temporal logics.

4.2 Probabilistic Anomaly Detectability

An important feature of CPS is detectability of system anomalies, independently of the type of used *sound* detector; this can be captured as probabilistic overshoot observability on system outputs, where the input overshoot captures that an anomaly has occurred. Specifically, we require that with probability of at least $1-\varepsilon$ it holds that: if (i) in one execution, a signal π steps (i.e., anomaly starts) and then stays bounded (e.g., within the modeled noise bound) for some time interval I; and (ii) in another execution, signal π' steps and then overshoots (i.e., beyond the noise bound); then (iii) the distance between the two signals is greater than a predefined threshold (i.e., the anomaly overshoot can be observed by a detector on system output). This is captured as the following HyperPSTL formula

$$\mathbb{P}^{\{\pi,\pi'\}}\Bigg(\Big(\Box\Big(step^{\pi}\Rightarrow\Box_I(x^{\pi}<c)\Big)\wedge\Diamond\Big(step^{\pi'}\wedge\Diamond_I(x^{\pi'}>c)\Big)\Big)\Rightarrow\;\Big(\Diamond_I d(y^{\pi},y^{\pi'})>c'\Big)\Bigg)>1-\varepsilon, \tag{6}$$

where x is the input and y is the output. As (5), the formula in (6) also involves probability quantification over a set of paths.

4.3 Workload Fairness in Queueing Networks

As shown in Figure 5, consider an embedded processing system with n front servers and m back servers (as in e.g., [29]). The requests (e.g., task, packets) arrive at each front-end queue, probabilistically over time, into buffers of different sizes. For each queue, the requests are preprocessed with probabilistic execution times, and delivered to back servers with different buffer sizes, following some scheduling policy. We can probabilistically model the arrival and processing of requests by Markov Modulated Poisson Processes (MMPP) of different parameters across all the servers [5]. In the general case, this setup yields no easy exhaustive solution.

Our goal is to check if a request-delivering policy between the front and back servers is fair [12, 13]; i.e., if a back server i is more likely to be overloaded than another back server j. Let τ_i and τ_j

[1] Meanwhile, $\mathtt{Q}^{\pi_2}$ may switch back to $\neg\mathtt{Q}^{\pi_2}$ for π_2, but the first hitting time τ^{π_2} will not change.

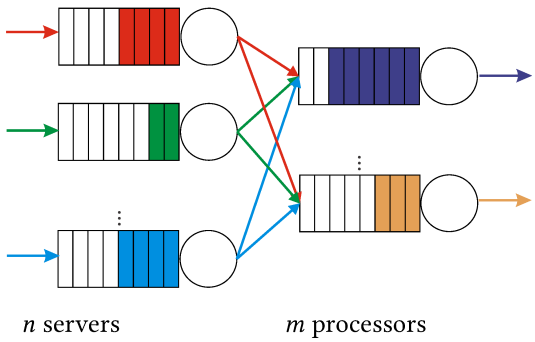

Fig. 5. Queueing Network.

be the overload times of the back server i and j, respectively. We define a fairness property that with probability of at least $1-\varepsilon$, given the overload time for the back server i, the back server j is overloaded much earlier or later (more than some $t>0$ than that time), with approximately equal probability (i.e., difference less than some δ)

$$\mathbf{Pr}_{\pi_1}\Big(\Big|\mathbf{Pr}_{\pi_2}(\tau_i^{\pi_1}-\tau_j^{\pi_2}>t)-\mathbf{Pr}_{\pi_2}(\tau_j^{\pi_2}-\tau_i^{\pi_1}>t)\Big|<\delta\Big)>1-\varepsilon.$$

To express this property in HyperPSTL, for $i\in[1,m]$, let $\mathtt{Q}_i$ be the predicate of the overload of the buffer of the i^{th} back server. The event that the back server i is overloaded earlier than the back server j more than time $\tau>0$, can be expressed as $(\neg\mathtt{Q}_i^{\pi_1}\wedge\neg\mathtt{Q}_j^{\pi_2})\,\mathcal{U}\,(\mathtt{Q}_i^{\pi_1}\wedge\Diamond_{[\tau,\infty)}\mathtt{Q}_j^{\pi_2})$. Hence, a fairness policy requirement is captured by a HyperPSTL formula

$$\begin{aligned}&\mathbb{P}^{\pi_1}\Big(\Big|\,\mathbb{P}^{\pi_2}\Big((\neg\mathtt{Q}_i^{\pi_1}\wedge\neg\mathtt{Q}_j^{\pi_2})\,\mathcal{U}\,(\mathtt{Q}_i^{\pi_1}\wedge\Diamond_{[\tau,\infty)}\mathtt{Q}_j^{\pi_2})\Big)\\&-\mathbb{P}^{\pi_2}\Big((\neg\mathtt{Q}_i^{\pi_1}\wedge\neg\mathtt{Q}_j^{\pi_2})\,\mathcal{U}\,(\mathtt{Q}_j^{\pi_2}\wedge\Diamond_{[\tau,\infty)}\mathtt{Q}_i^{\pi_1})\Big)\Big|\le\delta\Big)\ge1-\varepsilon.\end{aligned}\tag{7}$$

Note that (7) involves both comparison of probabilities and nesting of probability operators quantifying different paths, which are not allowable in common non-hyper temporal logics.

5 STATISTICAL VERIFICATION OF HYPERPSTL PROPERTIES

In this section, we study the SMC of HyperPSTL on a PUS with given inputs. As with previous works, we focus on handling the probability operators in HyperPSTL by sampling, which is the main issue for the SMC of probabilistic temporal logic. Through Sections 5.1 – 5.3, we propose SMC algorithms based on Clopper-Pearson (CP) significance level calculation for all the ways the probability operators can be used or nested in HyperPSTL. Accordingly, any nested HyperPSTL can be verified recursively by applying these SMC algorithms. The handling of temporal operators is similar to that of hyperSTL [23] and is thus not fully discussed due to the space limitations. For a PUS, verifying bounded-time properties is straightforward; verifying unbounded-time properties is more involving, and will be part of the future work.

5.1 SMC via CP Significance Level

A new feature of HyperPSTL compared to common temporal logics is the simultaneous probability quantification over multiple path variables. We now illustrate the idea of SMC for such formulas. Consider HyperPSTL formula $\Phi=(\mathbb{P}^{\Pi}\varphi<p)$, where (i) $\Pi=\{\pi_1,\ldots,\pi_K\}=\mathrm{fv}(\varphi)$ is the set of free path variables of φ, (ii) $p\in[0,1]$ is a probability threshold, and (iii) φ contains no probability

operator. The semantics of Φ is

$$p_\varphi = \mathbf{Pr}_{\boldsymbol{\sigma}\sim \mathrm{Path}^{|\Pi|}(X)}\Big(\big(\mathcal{S}, V_X[\Pi \to \boldsymbol{\sigma}]\big) \models \varphi\Big) < p.$$

The truth value of φ can be evaluated on a set of concrete sample paths $\boldsymbol{\sigma} = \{\sigma_1, \ldots, \sigma_K\}$ by assigning the concrete sample path σ_i to the free path variable σ_i for $i \in [K]$. Hence, with a slight abuse of notation, we denote

$$\varphi(\boldsymbol{\sigma}) = \begin{cases} 1, & \text{if } \varphi \text{ is true on } \boldsymbol{\sigma}, \\ 0, & \text{otherwise.} \end{cases} \tag{8}$$

Previous SMC approaches have used the sequential probability ratio test (SPRT) to evaluate Φ with the specification of an indifference margin [21]. Specifically, assuming that

$$|p_\varphi - p| > \delta \tag{9}$$

for some $\delta > 0$, to evaluate Φ, it suffices to test the two most indistinguishable cases, i.e., a Simple Hypothesis Testing (SHT) problem with two hypothesis,

$$H_0 : p_\varphi = p - \delta, \quad H_1 : p_\varphi = p + \delta, \tag{10}$$

which can then be solved by SPRT [17].

Since the choice of indifference margin is somewhat arbitrary, we propose an indifferent margin-free SMC approach via significance level calculation. For $i \in [N]$ and $K = |\Pi|$, let $(\sigma_1^{(i)}, ..., \sigma_K^{(i)})$ be a tuple of i.i.d. sample paths drawn from the PUS $\mathcal{S}$ starting from the state X. Checking the correctness of φ by (8) on each tuple gives the sum statistic

$$T = \sum\nolimits_{i\in[N]} \varphi\big(\sigma_1^{(i)}, \ldots, \sigma_K^{(i)}\big)$$

that obeys the binomial distribution $\mathrm{Binom}(n, p_\varphi)$. The average statistics T/N is a unbiased estimator for p_φ. Intuitively, when $T/N < p$, it is more likely that $p_\varphi < p$; and the same for the other case. Hence, we define the following statistical asserting function based on the samples

$$\mathcal{A}((\mathcal{S}, X) \models \Phi) = \begin{cases} 1, & \text{if } T/N < p \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

To ensure the asymptomatic correctness of the SMC algorithm, we assume that

$$p_\varphi \neq p, \tag{12}$$

which is a weaker assumption than (9). When (12) holds, as the number of samples increases, the samples will be increasingly concentrated on one side of p by the central limit theorem. Therefore, a statistical analysis based on the majority of the samples has an increasing accuracy. When (12) is violated, the samples would be evenly distributed on the two sides of p, regardless of the sample size. Thus, no matter how the sample size increases, the accuracy of any statistical test would not increase. This will be illustrated later in the proof of Theorem 5.1.

In general, the significance level for claiming $p_\varphi \in [a, b] \subseteq [0, 1]$ (i.e., an upper bound of the probability of making a wrong claim), when $T/N \in [a, b]$, can be computed using a method from Clopper and Pearson [8] by

$$\alpha_{\mathrm{CP}}(a, b \mid T, N) = 1 - \begin{cases} (1-a)^N - (1-b)^N & \text{if } T = 0 \\ b^N - a^N & \text{if } T = N \\ F_{\mathrm{Beta}}(b \mid T+1, N-T) - F_{\mathrm{Beta}}(a \mid T, N-T+1) & \text{otherwise.} \end{cases} \tag{13}$$

Algorithm 1 SMC of $(\mathcal{S}, X) \models \mathbb{P}^{\Pi}\varphi < p$.

Require: PUS $\mathcal{S}$, desired significance level α_d, batch size B.
1: $N \leftarrow 0$, $T \leftarrow 0$, $K \leftarrow |\Pi|$, initial significance level $\alpha_{\text{CP}} \leftarrow 1$
2: **while** $\alpha_{\text{CP}} > \alpha_d$ **do**
3: **for** $i \in [n]$ **do**
4: Draw $\sigma_{N+1}, \dots \sigma_{N+B}$ from X in $\mathcal{S}$.
5: $T \leftarrow T + \sum_{i=N+1}^{N+B} \varphi(\sigma_1^{(i)}, \dots, \sigma_K^{(i)})$; $N \leftarrow N + B$.
6: **end for**
7: Update $\mathcal{A}$ by (11) and α_{CP} by (13) and (14).
8: **end while**
9: **return** $\mathcal{A}$ and α_{CP}.

where $F_{\text{Beta}}(\cdot \mid T_1, T_2)$ is the cumulative probability function (CDF) of the beta distribution $\text{Beta}(T_1, T_2)$ with the shape parameters (T_1, T_2). For computing the significance level of the assertion $\mathcal{A}((\mathcal{S}, X) \models \Phi)$, we utilize

$$[a, b] = \begin{cases} [0, p], & \text{if } T/N < p, \\ [p, 1], & \text{if } T/N > p. \end{cases} \tag{14}$$

Remark 1. *The CP significance levels* (13) *have the following properties. First, $\alpha_{CP}(a, b \mid T, N)$ increases as the interval $[a, b]$ shrinks. That is, for any $[a', b'] \subseteq [a, b] \subseteq [0, 1]$, we have $\alpha_{CP}(a', b' \mid T, N) \leq \alpha_{CP}(a, b \mid T, N)$. In addition, for $T \notin \{0, N\}$ and $N \gg 1$, we have that*

$$\alpha_{CP}(a, b \mid T, N) \approx 1 - F_{Beta}(b \mid T, N - T) + F_{Beta}(a \mid T, N - T). \tag{15}$$

Since the beta distribution $Beta(T, N-T)$ has the mean T/N and the variance $T(N-T)/N^2(N+1)$, for fixed T/N, as the number of samples $N \to \infty$, the beta distribution becomes increasingly concentrated at T/N, and thus $\alpha_{CP}(a, b \mid T, N) \to 0$. This implies that the probability of making the wrong claim decreases as more samples are available.

Given a desired significance level α, we can design a new SMC algorithm by, at each iteration, collecting B new samples, computing the CP significance interval and stopping when the result is less than α, as summarized in Algorithm 1. Correctness of Algorithm 1 follows directly from the definition of significance level.

Theorem 5.1. *Algorithm 1 terminates with probability* 1 *and gives the correct statistical assertion with probability at least* $1 - \alpha_d$.

Proof. **Termination**: For $p_\varphi \in \{0, 1\}$, the proof is trivial. For $p_\varphi \in (0, 1)$, from assumption (12) and without loss of generality, let $p_\varphi < p$ and $\delta = p - p_\varphi$. Recalling the second half of Remark 1, for any $T/N \in [p_\varphi - \delta/2, p_\varphi + \delta/2] \subseteq (0, p)$, the variance of $\text{Beta}(T, N - T)$ is lower bounded by $\min_{x \in [p_\varphi - \delta/2, p_\varphi + \delta/2]} x(1-x)/(N+1)$. Therefore, as $N \to \infty$, it uniformly converges to 0. This implies that $\alpha_{\text{CP}}(0, p \mid T, N)$ uniformly converges to 0 – i.e., for any given $\alpha_d > 0$, there exists $N_0(p_\varphi, \delta) \in \mathbb{N}$, such that $\alpha_{\text{CP}}(0, p \mid T, N) < \alpha_d$ for any $N \geq N_0(p_\varphi, \delta)$ and $T/N \in [p_\varphi - \delta/2, p_\varphi + \delta/2] \subseteq (0, p)$.

With $N \geq N_0(p_\varphi, \delta)$, by the law of large numbers (or central limit theorem), we have $\mathbf{Pr}\Big(T/N \in [p_\varphi - \delta/2, p_\varphi + \delta/2]\Big) \to 1$, as the number of samples $N \to \infty$. Therefore, Algorithm 1 terminates with probability 1.

Correctness: Let τ be the step Algorithm 1 terminates and A be "the assertion $\mathcal{A}$ in (11) is correct", then $\mathbf{Pr}(A) = \sum_{i \in \mathbb{N}} \mathbf{Pr}(A \mid \tau = i)\mathbf{Pr}(\tau = i)$. By construction of the significance intervals, for any $i \in \mathbb{N}$, we have $\mathbf{Pr}(A \mid \tau = i) > 1 - \alpha_d$. In addition by **Termination**, we have $\sum_{i \in \mathbb{N}} \mathbf{Pr}(\tau = i) = 1$, Thus, $\Pr(A) \geq 1 - \alpha_d$. □

5.2 SMC of Joint Probabilities

Another new feature of HyperPSTL is the arithmetics and comparisons of the probabilities of multiple sub-properties. For example, we can compare the satisfaction probability of φ_1 and φ_2 by the HyperPSTL formula $p_1 < p_2$, where $p_1 = \mathbb{P}^{\Pi_1}\varphi_1$ and $p_2 = \mathbb{P}^{\Pi_2}\varphi_2$, according to the syntax (1), (2). For simplicity, let $\Pi_1 = \mathsf{fv}(\varphi_1)$ and $\Pi_2 = \mathsf{fv}(\varphi_2)$. Mathematically, this is equivalent to reasoning over the joint probabilities of these properties. Specifically, $p_1 < p_2$ can be equivalently expressed as a specification on the joint probability $(p_1, p_2) \in D$, where $D = \left\{(x_1, x_2) \in [0,1]^2 \mid x_1 < x_2\right\}$.

To formally capture this, we introduce an additional syntactic rule $(p_1, \ldots, p_n) \in D$ in (1), whose semantics is given by

$$(\mathcal{S}, X) \models (p_1, \ldots, p_n) \in D \Leftrightarrow \mathcal{S} \models \left([\![p_1]\!]_{V_X}, \ldots, [\![p_n]\!]_{V_X}\right) \in D, \tag{16}$$

where $D \subseteq [0,1]^n$ is measurable.

While the expressiveness of HyperPSTL is unchanged with the new rule (16), the conjunction and disjunction of several HyperPSTL formula can be simplified. For example, the HyperPSTL formula $\Phi_1 \wedge \Phi_2$ with

$$\begin{aligned} \Phi_1 &= \left(f_1(\mathbb{P}^{\Pi_1}\varphi_1, f_2(\mathbb{P}^{\Pi_2}\varphi_2, \mathbb{P}^{\Pi_3}\varphi_3)) > c_1\right) \\ \Phi_2 &= \left(f_3(\mathbb{P}^{\Pi_2}\varphi_2)) < c_2\right) \end{aligned} \tag{17}$$

can be equivalently written as $(\mathbb{P}^{\Pi_1}\varphi_1, \mathbb{P}^{\Pi_2}\varphi_2, \mathbb{P}^{\Pi_3}\varphi_3) \in D$, where

$$D = \left\{(x_1, x_2, x_3) \in [0,1]^3 \mid f_1(x_1, f_2(x_2, x_3)) > c_1, f_3(x_2) < c_2\right\}.$$

In addition, the new rule simplifies the SMC of the specification. Previously, it requires checking both Φ_1 and Φ_2 separately, and then a probabilistic composition of the two results. With the new rule, a single procedure of checking whether the joint probability $(\mathbb{P}^{\Pi_1}\varphi_1, \mathbb{P}^{\Pi_2}\varphi_2, \mathbb{P}^{\Pi_3}\varphi_3)$ is in D is sufficient.

We now demonstrate the idea of verifying the joint probability by checking a non-nested state formula

$$(\mathcal{S}, X) \models \left(\mathbb{P}^{\Pi_1}\varphi_1, \ldots, \mathbb{P}^{\Pi_n}\varphi_n\right) \in D,$$

where for $i \in [n]$, and φ_i contains no probability operator and $\Pi_i = \mathsf{fv}(\varphi_i)$. As a statistical approach is adopted, we assume that the exact probability of satisfying φ does not lie within the boundary of the test region D, as stated in Assumption 1.

Assumption 1. *To check $(\mathcal{S}, X) \models \left(\mathbb{P}^{\Pi_1}\varphi_1, \ldots, \mathbb{P}^{\Pi_n}\varphi_n\right) \in D$, we assume that (i) the test region D is a simply connected domain with $\mu_{\text{Borel}}(D) \neq 0$, and (ii)*

$$\left(\mathbf{Pr}_{\boldsymbol{\sigma}_1 \sim \text{Path}^{|\Pi_1|}(X)}\left(\left(\mathcal{S}, V_X[\Pi_1 \to \boldsymbol{\sigma}_1]\right) \models \varphi_1\right), \ldots, \mathbf{Pr}_{\boldsymbol{\sigma}_n \sim \text{Path}^{|\Pi_n|}(X)}\left(\left(\mathcal{S}, V_X[\Pi_n \to \boldsymbol{\sigma}_n] \models \varphi_n\right)\right) \notin \partial D.$$

Assumption 1 can be viewed as the multidimensional generalization of (12), and is necessary for asymptomatic correctness of the SMC algorithm, as discussed in Section 5.1.

Remark 2. *Compared to previous studies on SMC using sequential probability ratio tests (SPRT) [33, 35], Assumption 1 is weaker as it requires no a priori knowledge on the indifference margin.*

Remark 3. *From Assumption 1, we have that $\left(\mathbb{P}^{\Pi_1}\varphi_1, \ldots, \mathbb{P}^{\Pi_n}\varphi_n\right) \in D_1$ and $\left(\mathbb{P}^{\Pi_1}\varphi_1, \ldots, \mathbb{P}^{\Pi_n}\varphi_n\right) \in D_2$ are semantically equivalent if $\overline{D_1} = \overline{D_2}$. In addition, $\mathbb{P}^{\Pi} \in D$ and $\mathbb{P}^{\Pi}(\neg\varphi) \in D^c$ are semantically equivalent when $D^c = [0,1]\backslash D$.*

By the semantic rule (16), the SMC problem is converted to a composite hypothesis testing problem

$$\begin{cases} H_0: & (p_{\varphi_1}, \dots p_{\varphi_n}) \in D, \\ H_1: & (p_{\varphi_1}, \dots p_{\varphi_n}) \in [0,1]^n \backslash D, \end{cases} \tag{18}$$
$$p_{\varphi_i} = \mathbf{Pr}_{\boldsymbol{\sigma}_i \sim \mathrm{Path}^{|\Pi_i|}(X)}\Big(\big(\mathcal{S}, V_X[\Pi_i \to \boldsymbol{\sigma}_i]\big) \models \varphi_i\Big) \text{ for } i \in [n].$$

For each $i \in [n]$, let $\{\boldsymbol{\sigma}_j\}_{j\in[N_i]}$ be N_i tuples of i.i.d sample paths of the PUS $\mathcal{S}$ starting from the same state X that are used to estimate p_{φ_i}. Similar to Section 5.1 approach, we consider the statistics

$$T_i = \sum\nolimits_{j=1}^{N_i} \varphi_i(\boldsymbol{\sigma}_1^{(j)}, \dots \boldsymbol{\sigma}_{K_i}^{(j)}), \quad K_i = |\Pi_i|. \tag{19}$$

where $T_i \sim \mathrm{Binom}(N_i, p_{\varphi_i})$. We define the assertion using the average statistics T_i/N_i for $i \in [n]$ as

$$\mathcal{A}((\mathcal{S}, X) \models \Phi) = \begin{cases} 1 & \text{if } \left(\frac{T_1}{N_1}, \dots, \frac{T_n}{N_n}\right) \in D, \\ 0 & \text{otherwise.} \end{cases} \tag{20}$$

For this multi-dimensional case, to compute the exact CP significance level for a general domain D involves multi-dimensional integrations on it, which can be computationally intensive. Therefore, we compute an upper bound α_{CP} on the significance level of the assertion (20) by finding a hypercube $\prod_{i\in[n]}[a_i, b_i]$ such that

$$\left(\frac{T_1}{N_1}, \dots, \frac{T_n}{N_n}\right) \in \prod\nolimits_{i\in[n]}[a_i, b_i] \subset D. \tag{21}$$

Due to the monotonicity of significance levels, the significance level of $(T_1/N_1, ..., T_n/N_n) \in D$ is upper bounded by that of $(T_1/N_1, ..., T_n/N_n) \in \prod_{i\in[n]}[a_i, b_i]$, which can be computed directly by compositing the significance level of $T_i/N_i \in [a_i, b_i]$ for $i \in [n]$, using the results in Section 5.1. Thus, we compute an upper bound $\bar{\alpha}_{\mathrm{CP}}$ of the exact CP significance level as

$$\bar{\alpha}_{\mathrm{CP}} = 1 - \prod\nolimits_{i=1}^{n} \alpha_{\mathrm{CP}}(a_i, b_i \mid T_i, N_i), \tag{22}$$

where $\alpha_{\mathrm{CP}}(a_i, b_i \mid T_i, N_i)$ is defined in (13).

When implementing this verification approach, for each iteration, we look for a hypercube $\prod_{i\in[n]}[a_i, b_i]$ satisfying (21) with $a_i < b_i$ for $i \in [n]$. Although finding such a hypercube is only possible if $(T_1/N_1, ..., T_n/N_n) \notin \partial D$, this is guaranteed with probability 1 for large samples when Assumption 1 holds.

To minimize the upper bound of the significance level $\bar{\alpha}_{\mathrm{CP}}$ in (22), the hypercube should be preferably as large as possible. For a simple domain D, the analytic solutions of such a largest hypercube can be derived directly as a function of T_i, N_i for $i \in [n]$ and the functions defining ∂D More generally, especially if D is convex, the largest hypercube can be derived by solving the optimization problem of maximizing its volume while keeping it inside D. Admittedly, solving the optimization problem at every iteration can still be inefficient for some cases. To remedy for this, we can (1) reduce the frequency of computing the significance level by drawing samples in batches; and (2) search only for approximate maxima in optimization.

Finally, we note that the upper bound $\bar{\alpha}_{\mathrm{CP}}$ is asymptotically tight if a largest hypercube is used to compute it. This holds because, by the law of large numbers, as the number of samples increases, $(T_1/N_1, ..., T_n/N_n)$ concentrates near $(p_{\varphi_1}, \dots, p_{\varphi_n})$ and the largest hypercube converges to a constant one strictly containing $(p_{\varphi_1}, \dots, p_{\varphi_n})$. Thus, the probability of $(T_1/N_1, ..., T_n/N_n) \in \prod_{i\in[n]}[a_i, b_i]$ converges to that of that of $(T_1/N_1, ..., T_n/N_n) \in D$.

Based on the previous discussions, we derive Algorithm 2. Correctness of Algorithm 2 is given by Theorem 5.2 that can be proved in the same way as Theorem 5.1.

Algorithm 2 SMC of $(\mathcal{S}, X) \models \left(\mathbb{P}^{\Pi_1} \varphi_1, \ldots, \mathbb{P}^{\Pi_n} \varphi_n \right) \in D$.

Require: PUS $\mathcal{S}$, desired significance level α_d, batch size B.

$N_1, \ldots, N_n \leftarrow 0$, $\bar{\alpha}_{\text{CP}} \leftarrow 1$
for $i \in [n]$ **do**
  $K_i \leftarrow |\Pi_i|$, $T_i \leftarrow 0$
end for
while $\bar{\alpha}_{\text{CP}} < 1 - \alpha_d$ **do**
  for $i \in [n]$ **do**
    Draw $\sigma_{N_i+1}, \ldots \sigma_{N_i+B}$ from X in $\mathcal{S}$.
    $T_i \leftarrow T_i + \sum_{j=N_i+1}^{N_i+B} \varphi_i(\sigma_1^{(j)}, \ldots \sigma_{K_i}^{(j)})$; $N_i \leftarrow N_i + B$.
  end for
  Update $\mathcal{A}$ by (20) and $\bar{\alpha}_{\text{CP}}$ by (22).
end while
return $\mathcal{A}$ and $\bar{\alpha}_{\text{CP}}$.

Theorem 5.2. *Algorithm 2 terminates with probability* 1 *and gives the correct statistical assertion with probability at least* $1 - \alpha$.

5.3 SMC of Nested Probability Operators

Finally, we consider SMC of nested HyperPSTL formulas on the PUS $\mathcal{S}$. By the syntax of HyperPSTL in Section 3.1, a nested HyperPSTL formula is constructed iteratively in two ways: (i) replacing an atomic proposition with a general state formula, and (ii) consecutively nested quantification of free path variables in a non-nested formula. The former also appears in common temporal logics and the latter is unique to HyperPSTL.

For (i), we show the idea of SMC by checking the satisfaction on a state X of the nested formula $\Psi = \mathbb{P}^{\Pi} \psi[\rho] \in D_1$ derived by replacing an atomic proposition of ψ with a non-nested state formula $\rho = \left(\mathbb{P}^{\Pi_1} \varphi_1, \ldots, \mathbb{P}^{\Pi_n} \varphi_n \right) \in D_2$. In Ψ, we have $D_1 \subseteq [0,1]$, $D_2 \subseteq [0,1]^n$. If ρ is treated as an atomic proposition, Ψ becomes a non-nested state HyperPSTL formula as discussed in Section 5.1. The HyperPSTL state formulas nesting more probability operators in this fashion can be statistically verified in the same way.

To verify Ψ, we follow a compositional analysis similar to [26, 27]. If the state space $\mathcal{X}$ of the PUS is finite (e.g., a CTMC from Section 2.2), we can statistically verify the sub-formula ρ on each state X of the PUS $\mathcal{S}$ with significance level α_X, using Theorem 5.2 and Algorithm 2, and label the state with ρ if the assertion given by Algorithm 2 is $\mathcal{A}\left(V \models \rho\right) = 1$. Then, we can statistically verify the full formula Ψ on the relabeled PUS as a non-nested formula with significance level α_0 using Theorem 5.2. The overall significance level is $\alpha = \sum_{X \in \mathcal{X}} \alpha_X + \alpha_0$, which is only bounded when $\mathcal{X}$ is finite. The SMC for Ψ on infinite-state PUS provides an avenue for future work. For this work, it brings no limitation as we only verify such nested formulas on finite-state PUS (e.g., queueing in Section 6), while for P^{2}HIOA such formulas are not used to capture properties of interest.

To implement the SMC algorithm for Ψ on finite-state PUS, given the overall significance level α, we can split it into the summation $\sum_{X \in \mathcal{X}} \alpha_X + \alpha_0$. The simplest way is $\alpha_X = \alpha_0 = \alpha/(|\mathcal{X}| + 1)$ for $X \in \mathcal{X}$, where $\mathcal{X}$ is the number of states of the PUS $\mathcal{S}$. Then, we can employ Algorithm 2 to verify ρ on each state X with significance level α_X and then assert Ψ with significance level α_0 using Algorithm 2 again. This is summarized by Algorithm 3.

Algorithm 3 SMC of $(\mathcal{S}, X) \models \mathbb{P}^{\Pi}\psi[\rho] \in D_1$ with $\rho = \left(\mathbb{P}^{\Pi_1}\varphi_1, \ldots, \mathbb{P}^{\Pi_n}\varphi_n\right) \in D_2$.

Require: PUS $\mathcal{S}$, desired significance level α_d.

1: Split α_d into $\sum_{X\in\mathcal{X}} \alpha_X + \alpha_0$
2: **for** $X \in \mathcal{X}$ **do**
3: Verify $(\mathcal{S}, X) \models \rho$ on $\mathcal{S}$ by Algorithm 2 with significance level α_X and label X with ρ if the assertion is positive.
4: **end for**
5: Verify $(\mathcal{S}, X) \models \mathbb{P}^{\Pi}\psi[\rho] \in D_1$ on the relabeled $\mathcal{S}$ by Algorithm 2 with significance level α_0.

For (ii), we show the idea of SMC by checking the satisfaction on a state X of a HyperPSTL formula $\Psi = \mathbb{P}^{\Pi_1}(\mathbb{P}^{\Pi_2}\varphi < p_2) < p_1$, where the sub-formula φ contains no probability operator and all its path variables are probabilistically quantified by $\mathbb{P}^{\Pi_1}\mathbb{P}^{\Pi_2}$, i.e., $p_1, p_2 \in [0,1]$, $\Pi_1 \cap \Pi_2 = \emptyset$, and $\mathrm{fv}(\varphi) = \Pi_1 \cup \Pi_2$.

The SMC can be similarly done for HyperPSTL state formulas that nest more probability operators in this fashion. The formula Ψ says that with probability at most p_1, we can find a set of paths Π_1 such that the probability to find another set of paths Π_2 to satisfy φ is at most p_2. This formula can be equivalently expressed by $\Psi = \mathbb{P}^{\Pi_1}\left(\mathbb{P}^{\Pi_2} \in [0, p_2]\right) \in [0, p_1]$ using the rule (16). We note that for (ii), unlike (i), the state space $\mathcal{X}$ of the PUS can be infinite.

The main idea is as follows. For $i \in [N]$, let $\boldsymbol{\sigma}_i$ be a $|\Pi_1|$-tuple of i.i.d. sample paths starting from X in the PUS; define the indicator

$$T_i = \mathbf{I}\left((\mathcal{S}, X) \models [\![\mathbb{P}^{\Pi_2}\varphi < p_2]\!]_{V[\Pi_1 \to \boldsymbol{\sigma}_i]}\right) \tag{23}$$

of whether the partly instantiated formula $(\mathcal{S}, X) \models [\![\mathbb{P}^{\Pi_2}\varphi < p_2]\!]_{V[\Pi_1 \to \boldsymbol{\sigma}_i]}$ is true under this instantiating. Although T_i is not directly accessible, we can estimate it statistically using the assertion A_i of Algorithm 2 for any given significance level $\alpha_1 > 0$.

To verify the full formula Ψ, we only need to estimate the total number of positive instantiation $T = \sum_{i\in[N]} T_i$. We estimate it using $A = \sum_{i\in[N]} A_i$. Since $A_i \neq T_i$ with probability at most α_1 for all $i \in [N]$, we have $|T - A| < \Delta$ with the significance level

$$\alpha_2 = 1 - F_{\mathrm{Binom}}(\Delta \mid N, \alpha_1), \tag{24}$$

where F_{Binom} is the Binomial cumulative distribution function. Since

$$T \in [T_1, T_2] = [\min\{0, A - \Delta\}, \max\{A + \Delta, N\}], \tag{25}$$

we can check the full formula Ψ using these minimal and maximal estimations of T. Intuitively, if $T_2/N < p$ (hence $T_1/N < p$), it is more likely that Ψ is true; if $T_1/N > p$ (hence $T_2/N > p$), it is more likely that Ψ is false; otherwise, further sampling is needed. Thus, we define the following statistical asserting function by

$$\mathcal{A}((\mathcal{S}, X) \models \Psi) = \begin{cases} 1 & \text{if } T_2/N < p_1 \\ 0 & \text{if } T_1/N > p_1 \\ \text{undecided}, & \text{otherwise.} \end{cases} \tag{26}$$

When a final assertion is possible, its significance level is the larger one between plugging T_1 and T_2 into (13) and (14). Accordingly, the overall significance level α is

$$\alpha = \alpha_2 + \begin{cases} \alpha_{\mathrm{CP}}(0, p_1 \mid T_2, N) & \text{if } T_2/N < p_1 \\ \alpha_{\mathrm{CP}}(p_1, 1 \mid T_1, N) & \text{if } T_1/N > p_1 \end{cases} \tag{27}$$

Algorithm 4 SMC of $\mathbb{P}^{\Pi_1}(\mathbb{P}^{\Pi_2}\varphi < p_2) < p_1$.

Require: PUS $\mathcal{S}$, desired significance level α_d.
1: Set initial significance levels α_1 for $i \in [N]$.
2: $T \leftarrow 0$, $N \leftarrow 0$, $T_i \leftarrow 0$ for $i \in [N]$.
3: $c \leftarrow 1$, $\alpha \leftarrow 1$, $\alpha_1 \leftarrow \alpha_d$.
4: **while** $\alpha > \alpha_d$ **do**
5: $N \leftarrow N + 1$. Draw $\boldsymbol{\sigma}_{N+1}$ staring from X in $\mathcal{S}$.
6: **for** $i \in [N]$ **do**
7: Update A_i with significance level α_1 by Algorithm 2.
8: **end for**
9: $A \leftarrow \sum_{i \in N} A_i$, $\Delta \leftarrow c\alpha_1 N$.
10: Update α_2 by (24), T_1, T_2 by (25) and α by (27).
11: **if** $\alpha_2 > \alpha/2$ **then** $c \leftarrow c + 1$,
12: **else** $\alpha_1 \leftarrow \alpha_1/2$.
13: **end if**
14: **end while**
15: **return** Assertion given by (26).

where α_{CP} is given by (13).

To implement the SMC algorithm, given the overall significance level α, we need to simultaneously decrease both the significance level α_1 for making assertions on the partly instantiation of ψ for given values of Π_1, and the significance level α_2 for estimating the sum of those assertions. We start from $\Delta = c\alpha_1 N$ with $c = 1$ and $\alpha_1 = \alpha_d$. When α_2 is the main source of statistical error, i.e., $\alpha_2 > \alpha/2$, we decrease α_2 by increasing the parameter c by 1; otherwise, we decrease α_1 by reducing it by half. This is summarized in Algorithm 4.

6 EVALUATION

We numerically evaluate our SMC algorithms on several benchmarks with different complexity levels. Other probabilistic hyperproperties on different systems are handled in a similar manner, but due to space constraints here we focus on the discussed properties/systems. All benchmarks are implemented in Matlab/Simulink and are available in [9]. Specifically, the Toyota powertrain model is derived from [18]; and the queueing networks are implemented in Simulink using the SimEvents Toolbox [30]. Evaluations are performed on a laptop with 16 GB RAM and Intel Xeon E-2176 CPU.

For each benchmark, we evaluate the proposed SMC algorithms in different setups by changing the desired significance level α, as well as the parameters (e.g., δ, ε, and t) in the objective HyperPSTL specifications. The proposed SMC algorithms are executed repeatedly on each setup for 100 times. This is to check whether the probability for the proposed SMC algorithms to return the correct assertion is at least $1 - \alpha$: i.e., we repeat the SMC algorithm for each setup for 100 times, and check if it makes the correct assertion for at least $100(1 - \alpha_d)$ times.

For each setup, to compare it with the assertions of the proposed SMC algorithms, the truth value of the HyperPSTL specification of interest is derived by estimating the probabilities involved in it using an analytic solution or numerous sampling. Specifically, for the thermostat, we derive an analytic solution for the left hand side of (5) from its dynamics; this can be done due to its simplicity. For the powertrain, we estimate the left hand side of (5) by sampling 10^5 pairs of (π_1, π_2), for which the standard error is less than 0.01. For the queueing networks, we estimate the left hand side of (7) by drawing 500 samples for π_1. For each sample of π_1, we draw 500 samples for π_2 to

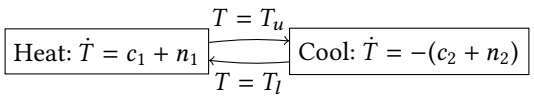

Fig. 6. Dynamical Model of a Thermostat.

δ	ε	α	Acc.	Sam.	Time (s)	Ans.
0.9	0.05	0.05	1.00	1.8e+02	2.1e+02	False
0.9	0.05	0.01	1.00	5.0e+02	6.0e+02	False
0.9	0.01	0.05	1.00	2.8e+01	3.4e+01	False
0.9	0.01	0.01	1.00	4.6e+01	5.5e+01	False
1.1	0.05	0.05	0.99	3.0e+02	3.5e+02	True
1.1	0.05	0.01	0.99	6.1e+02	7.3e+02	True
1.1	0.01	0.05	1.00	1.3e+02	1.6e+02	False
1.1	0.01	0.01	1.00	2.2e+02	2.6e+02	False

Table 1. Accuracy (Acc.), average number of samples (Sam.), average execution time (Time), and SMC results (Ans.) for sensitivity (5) with parameters δ and ε of Thermostat under significance level α (note the $1-\alpha$ accuracy guarantee by our SMC method).

evaluate the truth value of $\left|\mathbf{Pr}_{\pi_2}(\tau_i^{\pi_1} - \tau_j^{\pi_2} > t) - \mathbf{Pr}_{\pi_2}(\tau_j^{\pi_2} - \tau_i^{\pi_1} > t)\right| < \delta$. The total standard error for estimating the left hand side of (7) is less than 0.05.

Results for all considered setups are shown in Tables 1 to 4; as can be seen, the estimated accuracy of our SMC algorithms are very close to 1, showing the conservativeness of the CP significance level. We also report the average number of sample paths and the execution times for each setup based on the 100 repetitions, rounded up to their standard errors. In all simulations, the number increases when the desired significance level decreases, showing the trade-off between accuracy and the sampling cost. In addition, the execution time is mainly consumed by drawing samples from the Simulink models, and is approximately propositional to the number of samples.

6.1 Thermostat

A thermostat can be modeled as a simple P^{2}HIOA with two modes Heat and Cool (Figure 6), and one state variable T that varies within the temperature interval $[T_l, T_h] = [15, 40] \subseteq \mathbb{R}$. The mean heating and cooling rates are $c_1 = c_2 = 5$; they are subject to time-invariant but random Gaussian error $n_1, n_2 \sim \mathbf{N}(0, 0.5^2)$. The thermostat starts from $\big(T = T_l, \text{Heat}\big)$. We verify the sensitivity of the running period of a heat and cool cycle under the noise, which is represented in HyperPSTL by (5) with $\mathtt{Q} := \big(T = T_l, \text{Cool}\big)$. We statistically verified the sensitivity specification using Algorithm 2, with specification (5) parameters $\delta \in \{0.9, 1.1\}$ and $\varepsilon \in \{0.05, 0.01\}$, under the desired significance levels $\alpha \in \{0.01, 0.05\}$.

The derived results in Table 1 give accurate estimations (with significance level as low as $\alpha = 0.01$) on the probability distribution of the sensitivity, with a relatively small number of samples (at most a few hundred samples for each setup). We verified that the sensitivity of the running period of the thermostat is less than $\delta = 1.1$ with probability $1 - \varepsilon = 0.95$, but not less than $\delta = 0.9$ with the same probability, showing that the **0.95** percentile is between $[\mathbf{0.9}, \mathbf{1.1}]$. Also, as the sensitivity specification is false for $1 - \varepsilon = 0.99$ for both $\delta = 0.9$ and $\delta = 1.1$, showing that the **0.99** percentile is in $[\mathbf{1.1}, \infty]$.

δ	ε	α	Acc.	Sam.	Time (s)	Ans.
0.15	0.95	0.05	1.00	5.9e+01	8.1e+00	True
0.15	0.95	0.01	1.00	9.0e+01	1.3e+01	True
0.15	0.99	0.05	0.99	6.6e+01	9.1e+00	False
0.15	0.99	0.01	1.00	9.7e+01	1.4e+01	False
0.20	0.95	0.05	0.98	5.9e+01	8.1e+00	True
0.20	0.95	0.01	1.00	9.0e+01	1.2e+01	True
0.20	0.99	0.05	1.00	3.0e+02	4.2e+01	True
0.20	0.99	0.01	0.99	4.6e+02	1.8e+02	True

Table 2. Accuracy (Acc.), average number of samples (Sam.), average execution time (Time), and SMC results (Ans.) for sensitivity (5) with parameters δ and ε of Toyota Powertrain under significance level α (note the $1-\alpha$ accuracy guarantee by our SMC method).

6.2 Toyota Powertrain Control System

We use the Simulink model for the Toyota Powertrain with a four-mode embedded controller from [19]. It can be considered as a P^2HIOA with four modes and 15 state variables. We consider the sensitivity of the recovery time after start of the deviation percentage of the air/fuel (A/F) ratio μ to the level $|\mu| < 0.05$, under the mean RPM 2500 subjecting to Gaussian noise $\mathbf{N}(0, 25^2)$. The sensitivity specification is formally expressed by HyperPSTL formula (5) with $\mathtt{Q} = \big(|\mu| < 0.05\big)$; we statistically verified it using Algorithm 2 with parameters $\varepsilon \in \{0.05, 0.01\}$ and $\delta \in \{0.06, 0.07\}$, under the desired significance levels $\alpha \in \{0.01, 0.05\}$.

The results shown in Table 2 give accurate estimations (with significance level as low as 0.01) on the probability distribution of the sensitivity under the given embedded controller; this is achieved with a relatively small number of samples (at most a few hundred samples for each setup). We verified that the sensitivity of the recovery time of the powertrain is less than $\delta = 0.15s$ with probability $1-\varepsilon = 0.99$, but not less than $\delta = 0.20s$ with the same probability, showing that the **0.99** percentile is between $[\mathbf{0.15s}, \mathbf{0.20s}]$. Also, as the sensitivity specification is true for $1-\varepsilon = 0.95$ for both $\delta = 0.15s$ and $\delta = 0.20s$, showing that the **0.95** percentile is in $[\mathbf{0}, \mathbf{0.15s}]$.

6.3 Queueing Networks

We consider two queuing networks with different sizes m and n, as shown in Figure 5: the **Small** has 1 front-end servers and 2 back-end servers, the **Large** has 25 front-end servers and 20 back-end servers. For the small model, the package arrival and processing are modeled by exponential distribution, while in the second model, the package arrival and processing are modeled by Markov Modulated Poisson Processes with different parameters over all queues. Note that the parameters for different front-end servers are not identical, so deriving an exhaustive solution is virtually impossible for large sizes. We consider the fairness of workloads in queuing networks under the policy that the front servers deliver to the back server with the shortest queue. The fairness specification is formally defined in HyperPSTL by (7) with $\mathtt{Q}_i$ representing the overloading of the back server i. We set $i = 1$, $j = 2$, and statistically verify the specification with Algorithm 4 with $t \in \{0.1, 5.0\}$, $\delta \in \{0.1, 0.5\}$ and $\varepsilon \in \{0.1, 0.5\}$, for the significance level $\alpha = 0.95$.

From the first row of Tables 3 and 4, for both the small and large queueing network, with probability $1-\varepsilon = 0.9$ for the back server 1, we have that $|p_1 - p_2| < 0.1$, where p_1 and p_2 are the probabilities that back server 2 overloads $t = 5$ earlier/later than back server 1, respectively. This shows that for 90% cases, the shortest-queue-delivery policy is roughly fair for back-end server 1, in the sense that its overload time is not significantly sooner or later (for $t = 5$) than back-end

t	δ	ε	α	Acc.	Sam.	Time (s)	Ans.
0.1	0.1	0.1	0.05	1.00	1.5e+02	8.0e+01	False
0.1	0.5	0.5	0.05	1.00	1.4e+02	8.2e+01	False
5.0	0.1	0.1	0.05	1.00	7.3e+02	3.9e+02	True
5.0	0.5	0.5	0.05	1.00	3.1e+01	1.9e+01	True

Table 3. Accuracy (Acc.), average number of samples (Sam.), average execution time (Time), and SMC results (Ans.) for fairness (7) with parameters t, δ and ε of a Small Queueing Network (with 1 front-end server and 2 back-end servers) under significance level α.

t	δ	ε	α	Acc.	Sam.	Time (s)	Ans.
0.1	0.1	0.1	0.05	1.00	2.1e+02	3.2e+02	False
0.1	0.5	0.5	0.05	1.00	3.3e+02	4.9e+02	False
5.0	0.1	0.1	0.05	1.00	6.8e+02	1.1e+03	True
5.0	0.5	0.5	0.05	1.00	4.2e+01	6.6e+01	True

Table 4. Accuracy (Acc.), average number of samples (Sam.), average execution time (Time), and SMC results (Ans.) for fairness of a Large Queueing Network (with 25 front-end servers and 20 back-end servers) with parameters defined as in Table 3.

server 2. On the other hand, from the third row of Tables 3 and 4, with probability $1 - \varepsilon = 0.9$ for back-end server 1, we have $|p'_1 - p'_2| > 0.5$, where p'_1 and p'_2 are the probabilities that back server 2 overloads $t = 0.1$ earlier/later than back server 1, respectively. Thus, for 90% cases, the shortest-queue-delivery policy is not exactly fair for back server 1, in the sense that its overload time can be moderately sooner or later (for $t = 0.1$) than for back-end server 2.

7 CONCLUSION

In this work, we have studied statistical verification of hyperproperties for cyber-physical systems (CPS). We have first defined a general model of probabilistic uncertain systems (PUS) that unify commonly studied modeling formalisms such as continuous-time Markov chains and hybrid I/O automata with probabilistic dynamical parameters. We have then introduced hyperproperties, such as fairness and sensitivity, that involve relationships between multiple paths simultaneously in continuous time. To formally specify such hyperproperties, we have introduced Hyper Probabilistic Signal Temporal Logic (HyperPSTL), which is a hyper and probabilistic version of the conventional the signal temporal logic (STL). To simplify our presentation of the HyperPSTL syntax and semantics, while considering SMC of specific hyperproperties of complex CPS, in the proposed HyperPSTL, we have disallowed nested existential and universal quantifies on states to avoid exhaustive iteration on the possibly infinite state space. Still, this logic can be augmented (as done in detail in [31]) by allowing nested existential and universal quantifications over multiple states.

To verify HyperPSTL specifications on the PUS, we have developed statistical model checking (SMC) algorithms with three new features: (1) the significance level of the HyperPSTL specifications are computed directly using the Clopper-Pearson significance level; (2) statistically verifying HyperPSTL specifications on the joint probabilistic distribution of multiple paths, and (3) HyperPSTL specifications with nested probabilistic operators quantifying different paths are allowed. Finally, we have evaluated the introduced SMC algorithms on different CPS benchmarks with varying levels of complexity.

The work in this paper opens many new avenues for further research. Our work has direct application in software doping [10] and in particular, identifying whether two black-box systems are statistically identical. This problem is in particular challenging in the CPS domain, since the physical environment may make the behavior of cyber components more unpredictable. Another application area of our work is in conformance-based testing [1]. We also plan to expand our work to analyzing information-flow security and in particular differential privacy.

ACKNOWLEDGMENTS

This work is sponsored in part by the ONR under agreements N00014-17-1-2504, AFOSR under award number FA9550-19-1-0169, as well as the NSF CNS-1652544 and NSF SaTC-1813388 grants.

REFERENCES

[1] H. Abbas, H. D. Mittelmann, and G. E. Fainekos. 2014. Formal property verification in a conformance testing framework. In *Proceedings of the 12th ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*. 155–164.

[2] Erika Ábrahám and Borzoo Bonakdarpour. 2018. HyperPCTL: A Temporal Logic for Probabilistic Hyperproperties. In *Quantitative Evaluation of Systems*. 20–35.

[3] Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. The MIT Press.

[4] Benoît Barbot, Béatrice Bérard, Yann Duplouy, and Serge Haddad. 2017. Statistical Model-Checking for Autonomous Vehicle Safety Validation. In *SIA Simulation Numérique*.

[5] Gunter Bolch, Stefan Greiner, Hermann De Meer, and Kishor S Trivedi. 2006. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons.

[6] Lawrence D. Brown, T. Tony Cai, and Anirban DasGupta. 2001. Interval Estimation for a Binomial Proportion. *Statist. Sci.* 16, 2 (2001), 101–117.

[7] Michael R. Clarkson and Fred B. Schneider. 2008. Hyperproperties. In *2008 21st IEEE Computer Security Foundations Symposium*. 51–65.

[8] Charles J Clopper and Egon S Pearson. 1934. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika* (1934), 404–413.

[9] CPSL@Duke. 2019. HyperPSTL Case Studies. http://cpsl.pratt.duke.edu/research/statistical-model-checking-hyperpstl. Accessed: 2019-7-15.

[10] P. R. D'Argenio, G. Barthe, S. Biewer, B. Finkbeiner, and H. Hermanns. 2017. Is Your Software on Dope? - Formal Analysis of Surreptitiously "enhanced" Programs. In *Proceedings of the 26th European Symposium on Programming Languages and Systems (ESOP)*. 83–110.

[11] Parasara Sridhar Duggirala, Chuchu Fan, Sayan Mitra, and Mahesh Viswanathan. 2015. Meeting a powertrain verification challenge. In *International Conference on Computer Aided Verification*. Springer, 536–543.

[12] Atilla Eryilmaz and R Srikant. 2006. Joint congestion control, routing, and MAC for stability and fairness in wireless networks. *IEEE Journal on Selected Areas in Communications* 24, 8 (2006), 1514–1524.

[13] Leonidas Georgiadis, Michael J Neely, Leandros Tassiulas, et al. 2006. Resource allocation and cross-layer control in wireless networks. *Foundations and Trends® in Networking* 1, 1 (2006), 1–144.

[14] Daniel T Gillespie. 1976. A General Method for Numerically Simulating the Stochastic Time Evolution of Coupled Chemical Reactions. *J. Comput. Phys.* 22, 4 (1976), 403–434.

[15] Hans Hansson and Bengt Jonsson. 1994. A logic for reasoning about time and reliability. *Formal aspects of computing* 6, 5 (1994), 512–535.

[16] Thomas A Henzinger. 2000. The theory of hybrid automata. In *Verification of Digital and Hybrid Systems*. Springer, 265–292.

[17] Robert V Hogg, Joseph McKean, and Allen T Craig. 2005. *Introduction to mathematical statistics*. Pearson Education.

[18] Xiaoqing Jin, Jyotirmoy V Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. 2014. Benchmarks for Model Transformations and Conformance Checking. In *1st International Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH)*.

[19] Xiaoqing Jin, Jyotirmoy V. Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. 2014. Powertrain Control Verification Benchmark. In *The 17th International Conference on Hybrid Systems: Computation and Control*. 253–262.

[20] Kim G. Larsen and Axel Legay. 2016. Statistical Model Checking: Past, Present, and Future. In *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques*. 3–15.

[21] Axel Legay and Mahesh Viswanathan. 2015. Statistical Model Checking: Challenges and Perspectives. *International Journal on Software Tools for Technology Transfer* 17, 4 (2015), 369–376.

[22] Nancy Lynch, Roberto Segala, and Frits Vaandrager. 2003. Hybrid i/o automata. *Information and computation* 185, 1 (2003), 105–157.
[23] Luan Viet Nguyen, James Kapinski, Xiaoqing Jin, Jyotirmoy V. Deshmukh, and Taylor T. Johnson. 2017. Hyperproperties of Real-Valued Signals. In *15th ACM-IEEE Int. Conf. on Formal Methods and Models for System Design*. 104–113.
[24] Nima Roohi, Yu Wang, Matthew West, Geir E Dullerud, and Mahesh Viswanathan. 2017. Statistical Verification of the Toyota Powertrain Control Verification Benchmark. In *20th Int. Conf. on Hybrid Systems: Computation and Control*. 65–70.
[25] Dorsa Sadigh and Ashish Kapoor. 2016. Safe Control Under Uncertainty with Probabilistic Signal Temporal Logic. In *Robotics: Science and Systems Conference*.
[26] Koushik Sen, Mahesh Viswanathan, and Gul Agha. 2005. On Statistical Model Checking of Stochastic Systems. In *Computer Aided Verification*. 266–280.
[27] K. Sen, M. Viswanathan, and G. Agha. 2005. VESTA: A Statistical Model-Checker and Analyzer for Probabilistic Systems. In *Second International Conference on the Quantitative Evaluation of Systems*. 251–252.
[28] Jeremy Sproston. 2000. Decidable Model Checking of Probabilistic Hybrid Automata. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*. 31–45.
[29] Yi Tang and Neil W Bergmann. 2015. A hardware scheduler based on task queues for FPGA-based embedded real-time systems. *IEEE Trans. Comput.* 64, 5 (2015), 1254–1267.
[30] The MathWorks, Inc. 2019. SimEvents. https://www.mathworks.com/products/simevents.html. Accessed: 2019-7-15.
[31] Yu Wang, Siddhartha Nalluri, Borzoo Bonakdarpour, and Miroslav Pajic. 2019. Statistical Model Checking for Probabilistic Hyperproperties. *arXiv preprint arXiv:1902.04111* (2019).
[32] Yu Wang, Nima Roohi, Matthew West, Mahesh Viswanathan, and Geir E Dullerud. 2016. Verifying Continuous-Time Stochastic Hybrid Systems via Mori-Zwanzig Model Reduction. In *55th Conference on Decision and Control (CDC)*. 3012–3017.
[33] Yu Wang, Nima Roohi, Matthew West, Mahesh Viswanathan, and Geir E Dullerud. 2018. Statistical Verification of PCTL Using Stratified Samples. *IFAC-PapersOnLine* 51, 16 (2018), 85–90.
[34] Lijun Zhang, Zhikun She, Stefan Ratschan, Holger Hermanns, and Ernst Moritz Hahn. 2010. Safety Verification for Probabilistic Hybrid Systems. In *Computer Aided Verification*. 196–211.
[35] Paolo Zuliani. 2015. Statistical Model Checking for Biological Applications. *Int. Journal on Software Tools for Technology Transfer* 17, 4 (2015), 527–536.