information

Review

Monte Carlo Based Statistical Model Checking of Cyber-Physical Systems: A Review

Angela Pappagallo *, Annalisa Massini * and Enrico Tronci *

Computer Science Department, Sapienza University of Rome, Via Salaria 113, 00198 Rome, Italy
* Correspondence: pappagallo@di.uniroma1.it (A.P.); massini@di.uniroma1.it (A.M.); tronci@di.uniroma1.it (E.T.)

Received: 2 November 2020; Accepted: 7 December 2020; Published: 21 December 2020

Abstract: The ever-increasing deployment of autonomous Cyber-Physical Systems (CPSs) (e.g., autonomous cars, UAV) exacerbates the need for efficient formal verification methods. In this setting, the main obstacle to overcome is the huge number of scenarios to be evaluated. Statistical Model Checking (SMC) is a simulation-based approach that holds the promise to overcome such an obstacle by using statistical methods in order to sample the set of scenarios. Many SMC tools exist, and they have been reviewed in several works. In this paper, we will overview Monte Carlo-based SMC tools in order to provide selection criteria based on Key Performance Indicators (KPIs) for the verification activity (e.g., minimize verification time or cost) as well as on the environment features, the kind of system model, the language used to define the requirements to be verified, the statistical inference approach used, and the algorithm implementing it. Furthermore, we will identify open research challenges in the field of (SMC) tools.

Keywords: statistical model checking; cyber-physical systems; monte carlo sampling

1. Introduction

The ever-increasing deployment of autonomous Cyber-Physical Systems (CPSs) [1] (e.g., autonomous cars, Unmanned Autonomous Vehicles) exacerbates the need for efficient formal verification methods [2]. In this setting, the huge number of scenarios to be evaluated (*scenario explosion*) is the main obstacle to overcome. Statistical Model Checking (SMC) [3] holds the promise to overcome this obstacle by using statistical methods to sample the set of scenarios.

SMC algorithms use a simulation-based approach and statistical inference over observations in order to establish the correctness of the System Under Verification (SUV), within statistical confidence bound. Statistical inference can be applied for two different purposes: (i) *Hypothesis Testing (HT)*, allowing to infer whether the observations provide statistical evidence of the satisfaction of a property defined over the system; (ii) *estimation*, allowing to compute approximated values for some system parameters, whose probability distribution has to be known *a priori*.

Many SMC tools have been developed, suggesting how to carry out simulations and how many simulations to perform. The simulation runs affect the performance of each tool, which significantly varies according to several features. As a consequence, it is hard to know a priori the best tool suited for the verification problem at hand. In this work, we will overview a set of the Monte Carlo based SMC tools, currently available for research purposes, in order to provide selection criteria based on Key Performance Indicator (KPI) about: the verification activity (e.g., minimize verification time or cost); the environment; the kind of system model; the property specification language; the statistical inference approach; and, the algorithm that implements it. For example, if the verification KPI is to ease the parallelization of the simulation activity, our analysis allows us to identify the tools that compute beforehand the number of simulation runs needed to attain given statistical confidence. On the other hand, if the main verification KPI is the number of simulation runs, our analysis allows for us to identify the tools that minimize the number of scenarios to be sampled.

Of course, many SMC reviews are available. For example, Agha et al. in [4] give a very extensive survey of most of the tools available to the academic community, giving details about the kind of input models, the logic used to define properties, which emphasizes current limitations and trade-offs between precision and scalability. In [5], Reijsbergen et al. present a comprehensive overview of different algorithms performing HT (that will be discussed in Section 4.1). For each algorithm, the work in [5] focuses on its characteristics and performance. In [6], some of the most popular and well-maintained SMC tools are reviewed. The tools are compared based on their modelling and property specification languages, capabilities, and performances, in order to help users to select the most suitable tool to use.

Summing up, the output of our analysis is the following. First, providing tool selection criteria based on the five-tuple consisting of: SMC tool, environment model, SUV model (e.g., Discrete-Time Markov Chain (DTMC), Continuous-Time Markov Chain (CTMC), etc.), property specification, and statistical inference approach. On the other hand, making a comparison with the above-mentioned reviews: [4] focuses on tool selection criteria that are based on the pair consisting of SMC tool, SUV application domain (e.g., CPS, biological system, etc.); [5] focuses on identifying which SMC tool implements a given verification algorithm; and, [6] focuses on tool selection criteria based on the triple consisting of SMC tool, SUV model, property definition language.

Second, identifying open research challenges in the field of SMC tools, namely suggesting the deployment of SMC tools for the unbounded verification of hybrid systems with discrete or continuous time.

This paper is structured, as follows. In Section 2, we give a brief introduction to CPSs. In Section 3, we give some background information on the models that are used to represent the system to be verified and on the logics used to define properties. In Section 4, we discuss the main differences between the two approaches to statistical inference in SMC, i.e., HT, and Estimation. We also give some hints about *Bayesian Analysis* (see, e.g., [7]), although we will not go into details. In Section 5, we provide a taxonomy of the tools implementing Monte Carlo based SMC techniques. In Section 6, we analyze our taxonomy and discuss open research topics.

2. Cyber-Physical Systems

Our verification setting consists of three main components: the environment, the System Under Verification (SUV), and the specifications (see Figure 1).

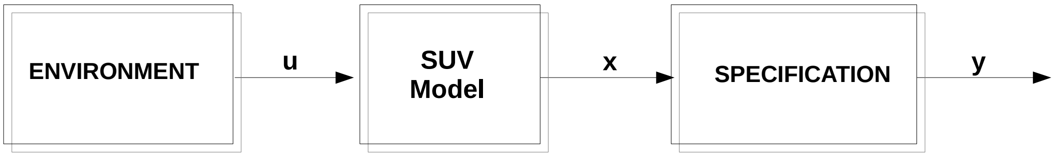

Figure 1. Simulation-based verification setting. Signal u models exogenous inputs, signal x models the SUV state, and signal y models the verification output.

For verification purposes, we need to model both the set of operational scenarios (SUV environment) as well as our SUV. This can be done by considering deterministic systems with stochastic inputs (the typical Control Engineering approach) or by considering stochastic systems (the typical Computer Science approach). When considering that from a uniformly distributed random variable in the real interval [0, 1], we can obtain a random variable with any specific (possibly dependent on the SUV state) distribution (e.g., as in Chapter 2. of [8]), and we can easily transform one modelling approach into the other. Accordingy, for example, a *Discrete-Time Markov Chain* $\mathcal{M}$ can be modeled with a *hlDiscrete-Time Dynamical System* $\mathcal{S}$ with a uniformly random stochastic input, from which we get $\mathcal{M}$ state-dependent transition probability while using techniques, like the those in [8]. Finally, from a practical point of view, we note that simulators will generate random numbers while using a pseudo-random number generator. Thus, both modelling approaches lead to the same implementation. Accordingly, for our purposes, we will consider them equivalent.

Many statistical model checkers work in a *black box* fashion by interfacing with widely used commercial as well as open source simulators for the design and the verification of CPSs in many application domains (e.g., automotive, avionics, IoT). Examples of such simulators are: Simulink [9], Dymola [10], SimulationX [11], Wolfram SystemModeler [12], MWorks [13], and Open Modelica [14]. Because the SUV model is not visible to such *black box* statistical model checkers (they will just run the simulator and check its outputs), we will not make any distinction between synchronous or asynchronous systems for them. Of course, the the situation is quite different for *white box* model checkers. They typically address this problem by defining probabilistic or non-deterministic schedulers, as done, for example, in PRISM [15] or SMV [16], e.g., following the approach in Chapter 2 of [17]. Using such approach, *white box* model checkers can also model both synchronous as well as asynchronous systems.

Because most statistical model checkers work in a *black box* fashion with simulators, here, we will follow the Control Engineering approach and model our SUV as a deterministic system with stochastic inputs.

2.1. Modelling the Environment

The *environment* (Figure 1) for the SUV defines the set of *operational scenarios* our SUV is supposed to withstand. In other words, an environment defines the set of *admissible inputs* for our SUV. Such inputs are usually named *exogenous* or *uncontrollable* inputs, since they are not under the control of the SUV. Typical examples of uncontrollable inputs are faults, change in system parameters, inputs from users, or from other systems.

In this section, we formalize the notion of *space of input functions* for a system. Such a space defines the system *environment*, i.e., the set of *admissible operational scenarios* for our SUV. To this end, first of all, we define the set of time instants, *time set* (Definition 1). In the following, as usual, the notation B^A denotes the set of functions from A to B.

Definition 1 (Time set)**.** *A* time set *T is a subgroup of $(\mathbb{R}, +)$.*

For example, when considering discrete-time systems T will be the set $\mathbb{Z}$ of all integer numbers, whereas, for continuous-time systems, T will be the set $\mathbb{R}$ of all real numbers.

Definition 2 (Space of input functions)**.** *Given a set U (of* input values*) and a time set T, a* space of input functions *$\mathcal{U}$ on (U, T) is a subset of $U^T = \{u \mid u : T \to U.\}$*

This allows for us to easily model application domain-dependent probability distributions on the SUV input functions as well as stochastic sampling strategies on the SUV input functions.

The above approach is quite convenient, since, in a simulation-based verification, setting the simulation model is typically deterministic and stochastic behaviors stems from uncontrollable events.

Of course, the more liberal the environment model, the larger the set of operational scenarios under which our SUV will be verified. On the other hand, the more liberal the environment model, the heavier the computational load entailed by the verification activity. Accordingly, a trade off is typically sought by verification experts between the *degree of assurance* (i.e., to which extent all meaningful operational scenarios have been considered) and the computational cost (and, thus, time cost) of the verification activity.

2.2. Modelling the SUV

We mainly focus on the case, where the System Under Verification (SUV) is a Cyber-Physical System (CPS), i.e., a system consisting of both hardware and software components. CPSs can be found basically in any domain. Examples are: biomedical devices, aerospace (e.g., airplanes, Unmanned Autonomous Vehicle (UAV), satellites), automotive, smart grid, etc. Indeed, many CPSs are safety or mission-critical systems, which, through software components, are endowed with some degree of *autonomous* behavior. From this stems the need for thorough verification of correctness of the software controlling the CPS. This motivates our focus on formal verification of CPSs.

From a formal point of view, a CPS is basically a dynamical system whose state variables can undergo continuous as well as discrete changes. Typically, continuous dynamics model the physical components of a CPS, whereas the discrete dynamics model the software components.

To formalize our notion of system, the following definition will be useful.

Definition 3 (Restriction)**.** *Let $\mathcal{I}$ be a time interval (i.e., an interval $\mathcal{I} \subseteq T$). Given a function $u \in U^{\mathcal{I}}$ (see Definition 2) and two positive real numbers $t_1 \leq t_2$, we denote with $u \mid_{[t_1,t_2)}$ the restriction of u to the interval $[t_1, t_2)$, i.e., the function $u \mid_{[t_1,t_2)}$: $[t_1, t_2) \to \mathcal{U}$, such that $u \mid_{[t_1,t_2)} (t) = u(t)$ for all $t \in [t_1, t_2)$. We denote $\mathcal{U}^{[t_1,t_2)}$ the* restriction *of $\mathcal{U}^{\mathcal{I}}$ to the domain $[t_1, t_2)$. That is, $\mathcal{U}^{[t_1,t_2)} = \{u \mid_{[t_1,t_2)} \mid u \in \mathcal{U}^{\mathcal{I}}\}$.*

The above considerations lead us to model (Definition 4) our SUV as a dynamical system, along the lines of [18].

Definition 4 (Dynamical System)**.** *A* Dynamical System*, $\mathcal{H}$ is a tuple $(X, U, Y, T, \mathcal{U}, \varphi, \psi)$, where:*

- *X, the* space of state values *of $\mathcal{H}$, is a non-empty set whose elements are said* states *of $\mathcal{H}$;*
- *U, the* space of input values *of $\mathcal{H}$, is a non-empty set whose elements are said* input values *for $\mathcal{H}$;*
- *Y, the* space of output values *of $\mathcal{H}$, is a non-empty set whose elements are said* output values *for $\mathcal{H}$;*
- *T is a time set;*
- *$\mathcal{U}$, the* space of input functions *of $\mathcal{H}$, is a non-empty subset of U^T;*
- *$\varphi : T \times T \times X \times \mathcal{U} \to X$, is the* transition map *of $\mathcal{H}$;*
- *$\psi : T \times X \times U \to Y$ is the* observation function *of $\mathcal{H}$.*

Function φ satisfies the following properties:

- Causality*. For all $t_0 \in T$, $t \geq t_0$, $x_0 \in X$, $u, u' \in \mathcal{U}$:*

$$u \mid_{[t_0,t)} = u' \mid_{[t_0,t)} \Rightarrow \varphi(t, t_0, x_0, u \mid_{[t_0,t)}) = \varphi(t, t_0, x_0, u' \mid_{[t_0,t)})$$

- Consistency*. For all $t \in T$, $x_0 \in X$, $u, \in \mathcal{U}$:*

$$\varphi(t, t, x_0, u) = x_0$$

- Semigroup*. For all $t_0 \in T$, $t > t_1 > t_0$, $x_0 \in X$, $u \in \mathcal{U}$:*

$$\varphi(t, t_0, x_0, u \mid_{[t_0,t)}) = \varphi(t, t_1, \varphi(t_1, t_0, x_0, u \mid_{[t_0,t_1)}), u \mid_{[t_1,t)})$$

In the following, unless otherwise stated, $\mathcal{H}$ denotes the tuple $(X, U, Y, T, \mathcal{U}, \varphi, \psi)$.

From a formal point of view, a CPS is just a dynamical system, where T is the set of nonnegative real numbers and state variables may undergo continuous (to model the physical part of the system) as well as discrete (to model the software part of the system) changes. This typically means that φ is not continuous (but, typically, may be assumed continuous *almost everywhere*). Definition 4 is general enough to encompass discrete as well as continuous-time systems and finite as well as infinite-state systems.

We refer the reader to Section 1 of [18] for examples of how familiar systems fit Definition 4. In particular, on how from a *state-based* description of the system dynamics (e.g., through Ordinary Differential Equation (ODE) for continuous-time systems or through recurrence equations for discrete-time systems) we can compute the transition function φ. In this respect, we note that, typically, the transition function φ is computed by simulation while using numerical techniques (e.g., see [19]), since it is seldom available in a closed form (except, e.g., for linear, time-invariant systems).

2.3. Modelling the Specifications

A specification (Figure 1) defines the requirements that our SUV should satisfy for all operational scenarios. In our simulation-based setting, a specification is also defined through a system, as outlined below.

Notation 1 (Product of input spaces). *Let T be a time set and $\mathcal{U} \subseteq U^T$, $\mathcal{V} \subseteq V^T$ be input spaces. By abuse of language we denote with $\mathcal{U} \times \mathcal{V}$ the set $\{\theta \in (U \times V)^T \mid \exists u \in \mathcal{U}, v \in \mathcal{V} \text{ s.t. } \theta(t) = (u(t), v(t))\}$.*

A specification takes, as input, the SUV input (*operational scenario*) and state and returns a real value assessing the extent to which requirements are satisfied. This is formalized in Definition 5.

Definition 5 (CPS Specification). *Let $\mathcal{H}$ = (X, U, Y, T, $\mathcal{U}$, φ, ψ) be a system. A specification for $\mathcal{H}$ is a system $\mathcal{Q}$ = (Z, $U \times X$, $\mathbb{R}$, T, $\mathcal{U} \times X^T$, μ, θ), where:*

- *Z is the space of state values of the system Q;*
- *$U \times X$ is the space of input values of the system Q;*
- *$\mathbb{R}$ is the space of output values of the system Q;*
- *T is the* time set *of the system Q (and $\mathcal{H}$);*
- *$\mathcal{U} \times X^T$ is the space of input functions of Q;*
- *μ is the transition map of Q; and,*
- *θ is the observation function of Q.*

Example 1 (Example of CPS Specification). *Let $\mathcal{H}$ be the system defined by the ODE*

$$\dot{x} = -3x + u$$

with u constant. Subsequently, $\mathcal{H}$ = (X, U, Y, T, $\mathcal{U}$, φ, ψ) with:

- *$X = U = Y = T = \mathbb{R}$;*
- *$\mathcal{U}$ is the set of constant functions from T to U;*
- *$\varphi(t_0, t, x_0, u) = e^{-3(t-t_0)}(x_0 - \frac{u}{3}) + \frac{u}{3}$;*
- *$\eta(t, \bar{x}, u) = \bar{x}$.*

Of course, in general, the function φ is not available in a closed form and it can only be computed through simulation (e.g., using simulators, like Simulink, Dymola, or Open Modelica).

Suppose that we want to check that the output from $\mathcal{H}$ is always close enough *to $\frac{u}{3}$. This could be assessed by computing the* Root Mean Squared Error *(RMSE) of x with respect to $\frac{u}{3}$. This leads to the specification $\mathcal{Q}$ = (Z, $U \times X$, $\mathbb{R}$, T, $\mathcal{U} \times X^T$, μ, θ), where: $Z = U = X = T = \mathbb{R}$ and the space of input functions of Q consists of pairs of functions (u, x), where u is a constant function and x maps reals to reals.*

Finally: $\mu(t_0, t, z_0, u, x) = z_0 + \int_{t_0}^{t}(x(\tau) - \frac{u}{3})^2 d\tau$ (RMSE) and $\theta(t, \bar{z}, u, x) = \bar{z}$.

In order to formalize the relationship between a system and its specification, we need to define the notion of *monitored system* (Definition 6) [20–22] (in the following, as usual in modern programming languages (e.g., Java, Python), the lambda notation $\lambda t.f(t)$ denotes the function f that, applied to t returns $f(t)$).

Definition 6 (Monitored system)**.** *Let $\mathcal{H}$ be a system and $\mathcal{Q}$ be a specification for it. The ($\mathcal{H}$, $\mathcal{Q}$)* monitored system *is the system $\mathcal{M}$ = ($X \times Z$, U, $\mathbb{R}$, T, $\mathcal{U}$, Φ, Ψ) where:*

$$\Phi(t, t_0, (x_0, z_0), u) = (\varphi(t, t_0, x_0, u), \mu(t, t_0, z_0, (u, \lambda t.\varphi(t, t_0, x_0, u))))$$

and

$$\Psi(t, (x, z), v) = \theta(t, z, (v, x))$$

Example 2 (Example of monitored system)**.** *Using the systems $\mathcal{H}$ and $\mathcal{Q}$ from Example 1, we have that the monitored system ($\mathcal{H}$, $\mathcal{Q}$) is the system $\mathcal{M}$ = ($X \times Z$, U, $\mathbb{R}$, T, $\mathcal{U}$, Φ, Ψ) with:*

- *X, Z, U, T and $\mathcal{U}$ as in Example 1;*
- $\Phi(t, t_0, (x_0, z_0), u) = (e^{-3(t-t_0)}(x_0 - \frac{u}{3}) + \frac{u}{3}, z_0 + \int_{t_0}^{t}((e^{-3(\tau-t_0)}(x_0 - \frac{u}{3}) + \frac{u}{3}) - \frac{u}{3})^2 d\tau)$;
- $\Psi(t, (\bar{x}, \bar{z}), u) = \bar{z}$

Thus, the output $\Psi(t, \Phi(t, t_0, (x_0, z_0), u), u)$ of the monitored system at time t is $z_0 + \int_{t_0}^{t}((e^{-3(\tau-t_0)}(x_0 - \frac{u}{3}) + \frac{u}{3}) - \frac{u}{3})^2 d\tau$.

The real value that is returned by Ψ defines our KPI evaluating to which extent the system meets its requirements under the *operational scenario* that is defined by input function u. If a KPI is Boolean (i.e., it takes only values in $\{0, 1\}$, then we have a *crispy* classification (i.e., the system will *pass* or *fail* to meet the given specification), or else we have a *metric* classification measuring the extent requirements are violated. On such a basis, our verification problem can be formulated, as follows.

Definition 7 (Verification problem)**.** *Let $\mathcal{H}$ be a system, $\mathcal{Q}$ be a specification for $\mathcal{H}$ and $t_0 \in T$ a time instant. We say that $\mathcal{H}$* satisfies *its specification $\mathcal{Q}$ from t_0 if for all $(x_0, z_0) \in (X \times Z)$, for all $u \in \mathcal{U}$, for all $t > t_0$, we have that: $\Psi(t, \Phi(t, t_0, (x_0, z_0), u), u(t)) > 0$.*

The verification problem *($\mathcal{H}$, $\mathcal{Q}$, t_0) consists in checking if $\mathcal{H}$ satisfies its specification $\mathcal{Q}$ from t_0* (PASS) *or, otherwise* (FAIL) *in providing a counterexample, i.e., a state $(x_0, z_0) \in (X \times Z)$, an admissible input (operational scenario) $u \in \mathcal{U}$ and a time instant $t > t_0$, such that $\Psi(t, \Phi(t, t_0, (x_0, z_0), u), u(t)) \leq 0$.*

To limit complexity, an upper limit h (*horizon*) is usually given to the value of t in Definition 7. In this case, we have a *time-bounded verification* problem.

2.4. Statistical Model Checking

From Definition 7, we see that a verification problem can be cast as the problem of finding a state, an admissible input (operational scenario), and a time instant, such that the system KPI is negative (i.e., requirements are violated).

Of course, the above problem is computationally prohibitive in general. Thus, many techniques have been developed to compute an approximate solution to it with formal assurance about the verification outcomes. Basically, we have two main options: *first*, focus on simple models (e.g., finite state systems) to make the problem tractable; *second*, use statistical techniques in order to sample the environment (set of operational scenarios) in order to answer the verification problem (Definition 7) with some given statistical confidence.

We note that probabilistic model checking (e.g., PRISM [15]) is an exact approach, since it computes exactly the probability of reaching certain states and it suffers of the state explosion problem, much as the deterministic model checking. In order to overcome this, statistical techniques are used. Furthermore, non exhaustive approaches, for example *falsification* [23], do not provide a formal guarantee about the verification outcome. Thus, both probabilistic model checking (see [24]) and non exhaustive verification are out of our scope, because: probabilistic model checking returns an exact result; and, non exhaustive verification does not provide a statistical confidence level.

SMC follows the *second* approach and, indeed, offers a set of methods and software tools to solve the above problem in many different settings.

In the following, we will survey the available tools to carry out formal verification via SMC and categorize them on the basis of the environment model (i.e., space of input functions in Definition 2), SUV model (i.e., system in Definition 4), and specification model (i.e., SUV specification in Definition 5) they can support. This will help users in deciding which tool to use for a given verification task, trading off between the completeness of the environment model (all operational scenarios represented), faithfulness of the SUV model (all possible behaviors represented), and expressiveness of the specification language (all requirements are adequately formalized).

Finally, from a computational perspective, we will categorize tools with respect to the SMC algorithm used. This will enable users to evaluate which kind of parallelism they can expect to easily implement on the basis of the sampling mechanism used.

3. Background

By defining a probability measure on our set of operational scenarios $\mathcal{U}$ (space of input functions—see Definition 2), we can regard the deterministic system (SUV) in Definition 4 as a stochastic system. In such a context, the verification problem in Definition 7 will aim at estimating the probability that the system specification is satisfied. Such a probability can be computed while using numerical as well as statistical techniques.

Numerical Model Checking (NMC) techniques (e.g., [25,26]) need an explicit finite state model (*white box*) for the SUV. They compute accurate results, but they do not scale very well to large systems, because they suffer from the *state space explosion problem*.

SMC methods avoid an explicit representation of the SUV state space. They use a simulation-based approach and statistical inference over observations in order to establish whether a property, specified through a stochastic temporal logic (which is then transformed into a specification for the SUV as in Definition 5), is true within statistical confidence bounds. When compared to numerical techniques, SMC overcomes the state explosion problem and scales better to big systems, but only guarantees correctness in a statistical sense. Nevertheless, it is sometimes the only feasible approach to the problem.

SMC takes as input a model defining the dynamics of a system $\mathcal{S}$ and a property φ, defined as a logical formula. The goal of SMC is to decide whether $\mathcal{S}$ satisfies φ with a probability that is greater than or equal to a certain threshold α. In formal terms, we denote the SMC problem as $\mathcal{S} \models P_{\geq \alpha}(\varphi)$.

In the next sections, we describe the models that can be used to represent the system and the kind of temporal logics used to express the properties. Because all of the approaches are simulation-based, we have that all SUV models and languages to define properties discussed in the following can be transformed, respectively, into the general definitions in Definitions 4 and 5.

3.1. System Models

Quite often, rather than considering deterministic systems with stochastic inputs as in Definition 4, SMC considers systems without inputs with nondeterministic or stochastic transitions. This can be accomodated in our setting, as discussed at the beginning of Section 2. Accordingly, in our context, we will consider these two modelling approaches to be equivalent.

SMC input systems can be modelled through different kind of structures, such as *DTMC* or *CTMC* [26,27], and by *Generalized Semi Markov Process (GSMP)* [28].

A DTMC is defined as a tuple $M = (S, s_0, P, L)$, where: S is a set of states (*state space*), $s_0 \in S$ is the initial state, $P : S \times S \to [0, 1]$ is the transition function, and L is a function labeling states with atomic propositions. A CTMC is a *Markov Chain*, where time runs continuously. GSMP [29] is a stochastic process where the transition from one state to another depends on a set of several possible events that are associated with the current state.

Furthermore, SMC can be applied to *Discrete Event Stochastic Processes (DESPs)*, consisting of a set of states, which can be infinite, and whose dynamic depends on a set of discrete events. DESPs can be modelled as *Generalized Stochastic Petri Net (GSPN)* [30], which is a bi-partite graph that consists of two classes of nodes: *places*, representing the state of the modelled system; *transitions*, encoding the model dynamics.

SMC algorithms for *Probabilistic Timed Automaton (PTA)* are also available [31]. Using PTA, we can represent systems with probabilistic and real-time features.

More recently, SMC has been applied also to timed and hybrid systems with a stochastic behavior, like *Stochastic Hybrid Automaton (SHA)* and *Stochastic Timed Automaton (STA)* (see [32]). Some SMC algorithms also accept, as input models, *Markov Decision Processes (MDPs)* [33], which are particular sequential decision models [34] that are characterized by a set of states, a set of actions, probabilistic transitions, and rewards, or costs, which are associated to the actions, such that each decision only depends on the current state and not on the past. MDPs are often used to model concurrent process optimization problems.

SMC tools, like VeSTa or MultiVeSTa, which will be discussed in the following Sections 5.1 and 5.2, take, as input, models whose behavior is described through *Probabilistic rewrite theories* [35], which is a general high-level formalism to specify probabilistic systems.

The SMC tool SAM (in Section 5.16) works on systems that are modelled through StoKLAIM [36], which is a Markovian extension of KLAIM (*Kernel Language for Agents Interaction and Mobility*) [37], a language that is used to model mobile and distributed systems.

3.2. System Properties

The properties to be evaluated represent a set of behaviors of the system under verification and can be qualitative or quantitative. Verifying a *qualitative* property means deciding between two mutually exclusive hypotheses, namely if the probability to satisfy the property is above or below a certain threshold. *Quantitative* properties concern computing the estimation of a stochastic measure, i.e., the expected value of the probability that a given property is satisfied.

Properties are expressed through a temporal logic and, in literature, many different kinds of logics exist, as outlined below.

Linear Temporal Logic (LTL) [38] is the top family of temporal logics that reason along linear traces through time, where each instant is followed by only one future step. *Bounded Linear Temporal Logic (BLTL)* extends *LTL* by adding upper time bounds (bounded time of steps when the time domain is discrete) to temporal operators. *Probabilistic Bounded Linear Temporal Logic (PBLTL)* adds probabilistic operators to BLTL. *PBLTLc* corresponds to PBLTL with numeric constraints. Metric Temporal Logic (MTL) [39] and Metric Interval Temporal Logic (MITL) [40] both extend LTL by introducing an explicit representation of time.

Another class of stochastic logics considers time evolving in a branching fashion, rather than on a line. This class includes the *Computational Tree Logic (CTL)* [41], which is used to express qualitative properties over (non-stochastic) transition systems. *Probabilistic Computational Tree Logic (PCTL)* [42] extends CTL by including operators and time bounds to express the quantitative properties of a DTMC, say M. If s is a state of M and φ is a PCTL formula, then $s \models P_{\geq\alpha}(\varphi)$ means that "the probability that φ is satisfied in the next state of outgoing paths from state s is greater than or equal to α".

Several extensions of PCTL exist, like *Continuous Stochastic Logic (CSL)* [43], which can be used to express quantitative properties of *CTMCs*; *Quantitative Temporal Expressions (QuaTEx)* [35], a query language that is used to investigate quantitative aspects of probabilistic rewrite systems; *PCTL** [44], used to specify quantitative properties of *DTMCs* or *MDPs*. *PCTL** formulas can be interpreted over the states of a fully or concurrent probabilistic system. CSL^{TA} [45] is a superset of CSL for CTMC that allows for specifying path formulas through a one-clock Deterministic Timed Automaton (DTA).

Hybrid Automata Stochastic Language (HASL) is a temporal logic formalism that is used to define properties over DESPs (Section 3.1). A HASL formula consists of an automaton and an expression. The *automaton* is a *Linear Hybrid Automaton (LHA)*, i.e., a hybrid automaton with data variables whose dynamic depends on the modelled system states. The *expression* refers to the moments of path random variables associated with path executions of the system.

Finally, *Mobile Stochastic Logic (MoSL)* is a temporal stochastic logic that was introduced in [36]. MoSL is based on CSL and allows us to refer to the spatial structure of a mobile network. MoSL can be used to specify both qualitative and quantitative properties.

4. Statistical Inference Approaches

When using SMC algorithms, the system is simulated many times by executing Monte Carlo experiments. The property φ to be verified is checked at each simulation (sample). Let us associate a Bernoulli random variable Z to one simulation. We assume that Z is 1 if φ is satisfied, 0 otherwise, and that $Pr[Z = 1] = p$ and $Pr[Z = 0] = 1 - p$.

HT, as described in Section 4.1, *Estimation*, Section 4.2, and *Bayesian analysis*, Section 4.3, can be used as engines for the SMC algorithms, as outlined below.

4.1. Hypothesis Testing

The hypothesis testing approach consists of the evaluation of two mutually exclusive hypotheses, namely the *null hypothesis* $H_0 : p \geq \theta$, and the *alternative hypothesis* $H_1 : p < \theta$, where θ is a given threshold value. If we assert that H_0 is false while it is true, we make a *Type I* error (or *false positive*); if we decide to reject H_1, while it is true, we make a *Type II* error (or *false negative*). Let us denote, with α, the probability of Type I errors and with β the probability of Type II errors.

HT can be used to analyze both *qualitative* and *quantitative* properties. It can be performed on a set of samples that can be generated in advance or on demand. In the former case, the sample size is fixed. In the latter case, called *sequential testing*, the samples are collected incrementally and their number is not predetermined. In a sequential testing approach, the decision of whether or not to continue sampling is made based on the samples that were generated up to that point. Mixtures of the two types of sampling generation are possible. This is discussed in [5], where several approaches to HT are described, differing from each other in the guarantees they give about the correctness of the result. The algorithms implementing these approaches are the following: the *Chernoff C.I.* method, using a fixed sample size test, resting on a confidence interval based on the Chernoff–Hoeffding bound [46]; the *Gauss C.I.*, that is a similar procedure based on the Gaussian approximation to determine the confidence interval; the *Chow–Robbins* test, a sequential test method that continues sampling until the width of the confidence interval has reached a given value; the *Sequential Probability Ratio Test (SPRT)* technique of Wald [47] and its fixed sample size variant, the *Gauss–SSP* test, where SSP stands for *Single Sampling Plan* (see [48] for details).

When compared to simple sequential testing, SPRT is optimal in the sense that it minimizes the average sample size before a decision is made. The last method is the *Chernoff–SSP* test, which is a variant of the Gauss-SSP test using the Chernoff–Hoeffding bound instead of the Gaussian approximation.

When properties to be verified are rare, that is the probability that they hold is low, the number N of simulations required can explode. Fortunately, two techniques, such as *Importance Sampling (ISA)* and *Importance Splitting (ISS)*, can be applied to reduce N. ISA works by sampling through a weighted distribution, such that a rare property is more likely to be observed. Subsequently, the results are compensated by the weights, in order to estimate the probability under the original distribution. ISS works by decomposing the sampling from the rare probability distribution into the sampling from the product of less rare conditioned probabilities. See [49–51] for an in-depth discussion about the two mentioned rare properties techniques.

4.2. Estimation

Estimation is the kind of inference approach that is typically used to verify quantitative properties. Existing SMC algorithms rely on classical Monte Carlo based techniques in order to estimate a property with a corresponding *confidence interval*. The system model is repeatedly simulated and, if φ is the property to be evaluated and p is the probability to satisfy φ, an (ϵ, δ) approximation $\hat{p}$ of p is computed, with the assurance that the probability of error is $Pr(|\hat{p} - p| \geq \epsilon) \leq \delta$, where ϵ and δ are, respectively, the precision and confidence.

The confidence interval can be computed using different approaches, each one affecting the number of simulations N to perform. For instance, according to the Chernoff–Hoeffding bound [46] N is fixed a priori by setting $N = (\ln 2 - \ln \delta)/(2\epsilon^2)$. The *Confidence Interval (CI)* method, taken as input two parameters α and δ, samples until the size of the $(1 - \alpha) \times 100\%$ confidence interval, computed while using the *Student's t-test*, is bounded by δ.

In [52,53], Grosu and Smolka show how to compute an (ϵ, δ) approximation $\hat{p}$ of p satisfying the property:

$$Pr[p(1 - \epsilon) \leq \hat{p} \leq p(1 + \epsilon)] \geq (1 - \delta) \tag{1}$$

This can be done with an optimal number of simulation traces N, through the Optimal Approximation Algorithm $\mathcal{OAA}$ that was introduced by Dagum et al. in [54]. $\mathcal{OAA}$ uses the minimum number N of traces to compute an (ϵ, δ) approximation, within a constant factor, of a random variable distributed in $[0, 1]$. $\mathcal{OAA}$ algorithm improves the classic Monte Carlo algorithm, which is based on the hypothesis testing approach, since N is computed at runtime. This result is obtained by applying the sequential testing approach of Wald [47] to decide when to stop simulating. As well as for HT, as discussed in Section 4.1, when quantitative properties to be verified are rare, ISA or ISS can be applied (see [49–51]), together with the algorithm that was selected to solve the estimation problem, in order to reduce the number of simulations to perform.

4.3. Bayesian analysis

The Bayesian approach to SMC is based on the usage of *Bayes's theorem* and *sequential sampling*, and can be associated with both HT and Estimation. Additionally, in this case, as for Hypothesis Testing (Section 4.1) and Estimation (Section 4.2) algorithms, the system model is repeatedly simulated. The difference is that Bayes's theorem requires prior knowledge of the model to be verified, which is the *prior* density of a given random variable. Sequential sampling, as discussed in Section 4.1, means that the number of simulations to perform is decided at run-time. The properties to be verified are expressed as BLTL formulas and input systems are modelled as Discrete-Time Hybrid System (DTHS) (see Section 2).

The algorithm to perform SMC by Bayesian HT, introduced in [55], at each trace of the system model, generated by simulation, checks if a certain property is satisfied or not. Subsequently, it uses the *Bayes factor* $\mathcal{B}$ to decide whether the *null hypothesis* H_0 or the *alternative hypothesis* H_1 has to be accepted.

Zuliani et al. in [7] propose a Bayesian statistical Estimation algorithm. The main goal of this approach is estimating the (unknown) probability p that a random execution trace of the system model satisfies a fixed property ϕ. The estimate corresponds to a *confidence interval*, i.e., an interval that contains p with a high probability. Since p is unknown, it must be assumed that p is given by a random variable whose density is known *a priori*. The Bayesian Estimation algorithm iteratively generates traces of the system model and for each trace executes the following steps: checks whether ϕ is satisfied; computes the posterior mean, whhich is the Bayes estimator, of p; computes an interval estimate; produces a posteriori probability of p.

Finally, Bortolussi et al. in [56] present a novel approach, to the statistical model checking, based on the Bayes analysis and it is called *Smoothed Model Checking*. Their goal is computing a statistical estimation (and a confidence interval) of the satisfaction of a MITL property against an *Uncertain* CTMC, which is a CTMC indexed by a parameter vector.

4.4. Summary of the Algorithms Used for HT and Estimation

In Table 1, we summarize the main algorithms, as explained in the previous sections, which can be used to carry out Hypothesis testing or Estimation. For each algorithm, we specify whether the number N of simulation runs to execute is predetermined or not.

Table 1. For each considered algorithm we show with a • if it supports *Hypothesis Testing* (HT) and/or *Estimation* (E). In the last column *yes* means that the algorithm pre-computes the number of samples (#Samples Fixed a Priori), whereas *no* means that the number of samples is computed at runtime.

Algorithm	HT	E	#Samples Fixed a Priori
Gauss-SSP	•		*yes*
C.I.		•	*yes*
Chernoff C.I.	•	•	*yes*
Chernoff SSP.	•	•	*yes*
Chow-Robbins	•	•	*no*
SPRT	•		*no*
Bayesian HT	•		*no*
Bayesian E		•	*no*
$\mathcal{OAA}$		•	*no*

5. Statistical Model Checking Tools

In this section, we describe some of the existing Monte Carlo based SMC tools, focusing on those that are directly available to the academic community. There exist tools using simulation-based techniques, but whose output is not a statistical confidence interval, which are not included in our review, like: SyLVaaS [57], a Web-based software-as-a-service tool for System Level Formal Verification of CPSs; S-TaLiRo [58], which is a tool performing temporal logic falsification of properties over non-linear hybrid systems; Ariadne [59], which is a tool for the formal verification of CPSs, using reachability analysis for nonlinear hybrid automata; SpaceEx, a tool platform for reachability and safety verification of continuous and hybrid systems [60]; Symbolic PathFinder [61], a SMC tool implementing approximate algorithms for MDP.

For each surveyed tool, we give a brief description and some details about: the kind of models supported; the languages used to specify the properties to be verified; and, the statistical inference approach used (HT and/or Estimation and/or Bayesian analysis). The information about the environment model, that stem from the kind of model, are summarized in Table 2.

Tools performing statistical inference by algorithms not discussed in Section 4 (e.g., SAM, GreatSPN) will not be included in the taxonomy that is given in Table 2.

5.1. (P)VeStA

VeStA is a Java-based tool for statistical analysis of probabilistic systems. It implements the statistical methods from [27,43], based on Monte-Carlo simulation and simple statistical hypothesis testing [62].

Model and Property. This tool can verify properties expressed as CSL/PCTL formula (Section 3.2), against probabilistic systems that are specified as CTMCs or DTMCs (Section 3.1). VeStA is able also to statistically evaluate, in a Monte Carlo based way, the expected value of properties expressed in *QuaTEx* (Section 3.2), over the observations performed on *probabilistic rewrite theories* models. In this last case, models are described through PMAUDE, which is an executable algebraic specification language [35].

Statistical Inference approach. VeStA performs SMC by using classic statistical hypothesis testing (Section 4.1), rather than sequential hypothesis testing, according to the algorithm described in [63]. In particular, VeStA implements the *Gauss-SSP* hypothesis testing (Section 4.1), which is a fixed sample size version of the SPRT algorithm of Wald (Section 4.1). Consequently, it is easily parallelizable. PVeStA [64] is the tool extending and parallelizing the SMC algorithms implemented in VeStA.

5.2. MultiVeStA

MultiVeStA [65] extends VeStA (and PVeStA) with an important feature that is the integration with several existing discrete event simulators, in addition to the already supported ones.

Model and property. Properties to be verified are expressed in *Multi Quantitative Temporal Expressions (MultiQuaTEx)* query language, which extends *QuaTEx* (Section 3.2) and allows for querying more variables at a time through multiple observations on the same simulation. This represents an improvement of the performance obtained when evaluating several expressions. The supported SUV models are Discrete Event Systems (DESs).

Statistical Inference approach. MultiVeStA performs an estimation of the expected value of *MultiQuaTEx* properties by HT the Chow–Robbins method (Section 4.1).

5.3. Simulation-Based SMC for Hybrid Systems

There are many tools supporting SMC for hybrid systems, for example: ProbReach [66], Probably Approximately Correct (PAC) for hybrid systems verification [67,68], and Plasma [69]. Here, we focus on tool that can interface in a black-box fashion with any available simulator for hybrid systems. One of those is Plasma [70,71] which is a SMC platform that may be invoked from the command line or embedded in other software as a library. This tool uses external simulators (e.g., Simulink, SystemC) in order to define the SUV as well as the property to be verified.

Model and Properties. Plasma Lab accepts properties described as BLTL extended with customized temporal operators, against stochastic models such as CTMCs and MDPs (Section 3.1).

Statistical Inference approach. Plasma Lab can verify both qualitative and quantitative properties. In fact, the tool implements, among others, the following algorithms: the Monte Carlo probability estimation based on the Chernoff–Hoeffding bound [46], to decide *a priori* the number of simulations to execute; the SPRT algorithm for hypothesis testing; *ISA* when the properties are "rare" (Section 4.1).

5.4. APMC

The *Approximate Probabilistic Model Checker* (APMC) is a model checker implementing an efficient Monte Carlo-based method in order to approximate the probability that a monotone property is satisfied, with high confidence, by a fully probabilistic system. This algorithm is described in [72]. In [73] the optimal approximation algorithm $\mathcal{OAA}$ (Section 4.2) is used to obtain a randomized approximation scheme with multiplicative error parameter.

Model and property. APMC is used to verify quantitative properties over fully probabilistic transitions systems or DTMCs (Section 3.1). In 2006, APMC has been extended to manage also CTMCs (see [74]). Properties to be checked are expressed as LTL (Section 3.2) formulas.

Inference approach. This tool performs estimation through a Monte-Carlo sampling technique, based on the Chernoff–Hoeffding bound (Section 4.1), which is naturally parallelizable.

5.5. PRISM

PRISM [15] is a Probabilistic Symbolic Model Checker that from the version 4.0 offers support for SMC of several types of probabilistic models. Furthermore, in [75], a parallelized version of PRISM has been extended to handle MDPs.

Model and property. DTMCs, CTMCs, MDPs models (Section 3.1) are described through the PRISM modelling language, while the properties to be verified are defined by several probabilistic temporal logics, incorporating PCTL, PCTL*, CSL. and LTL (Section 3.2).

Statistical Inference approach. The tool uses the SPRT (Section 4.1) in order to verify qualitative properties and the following algorithms to verify the quantitative properties (Section 4.2): *CI* method, *Asymptotic Confidence Interval (ACI)* method, and *Approximate Probabilistic Model Checking (APMC)* method. All of these algorithms precompute the number of samples to be generated. See [76] for an updated detailed description.

5.6. Ymer

Ymer, as illustrated in [77], is a command-line tool written in C and C++, which implements the SMC techniques that are based on discrete event simulation and acceptance sampling. Furthermore, in the Ymer tool, two sampling-based algorithms for the statistical verification of time-unbounded properties of DTMCs are implemented (see in [78]).

Model and property. This tool can verify CSL properties against CTMCs and PCTL properties against DTMCs (Sections 3.1 and 3.2).

Statistical Inference approach. Ymer implements both sampling with a fixed number of observations and sequential acceptance sampling, performed through the SPRT method (Section 4.1). Ymer includes support for distributed acceptance sampling, i.e., the use of multiple machines to generate observations, which can result in significant speedup as each observation can be generated independently. The work in [78] also implements, in Ymer, estimation through two different approaches, the first based on Chernoff C.I and the second based on the Chow–Robbins sequential method.

5.7. UPPAAL-SMC

UPPAAL-SMC [79] is a stochastic and statistical model checking extension of UPPAAL [80], which is a toolbox for the verification of real-time systems, jointly developed by Uppsala University and Aalborg University.

Model and Properties. UPPAAL-SMC implements techniques in order to verify both quantitative and qualitative properties of timed and hybrid systems with a stochastic behavior, whose dynamic can be specified by *SHA*, effectively defining ODEs, and by STA [32]. Properties are expressed through a weighted extension of the temporal logic MITL (Section 3.2).

Statistical Inference approach. This tool carries out quantitative properties verification through a Monte Carlo based estimation algorithm using the Chernoff–Hoeffding bound (Section 4.1), where the number of samples to be generated is predetermined. Qualitative properties are verified through the SPRT algorithm (Section 4.1).

5.8. COSMOS

COSMOS [30] is a statistical verification tool that is implemented in C++.

Model and property. This tool analyzes DESPs, a class of stochastic models, including CTMCs, represented in the form of a GSPN (Section 3.1). Properties to be verified are expressed as HASL formulae (Section 3.2).

Statistical Inference approach. COSMOS relies on Confidence Interval based methods to estimate the probability that the property under verification holds, by implementing two possible approaches: the *static sample size estimation*, based on the Chernoff–Hoeffding bound (Section 4.1), where the sample size is fixed *a priori*; the *dynamic sample size estimation*, where the sample size depends on a stopping condition, such as that provided by Chow and Robbins (Section 4.1). COSMOS also provides the SPRT method.

5.9. GreatSPN

GreatSPN [81] is a tool that supports the design and the qualitative and quantitative analysis of GSPN. GreatSPN contains, in particular, two modules: a CTL model checker and a CSL^{TA} model checker.

Model and Properties. GreatSPN can verify: CTL properties against models represented as GSPN or its colored extension, defined as *Stochastic Symmetric Nets* (SSN); CSL^{TA} properties (Section 3.2) against CTMCs.

Statistical Inference approach. The CTL model checker of GreatSPN verifies CTL properties by numeric symbolic algorithms (see Section 3). The CSL^{TA} module is a probabilistic model checker for estimation of properties that can a.so interact with external tools, like PRISM (Section 5.5) and MRMC (Section 5.10).

5.10. MRMC

MRMC (Markov Reward Model Checker) [82] is a command-line tool, written in *C* language, which implements SMC techniques. This tool is a *Probabilistic Model Checker*, rather than a SMC. However, as MRMC performs statistical inference through Monte Carlo-based techniques, it is included in our review, for sake of completeness.

Model and property. The tool can verify CSL and PCTL properties (Section 3.2) against CTMCs and DTMCs (Section 3.1).

Statistical Inference approach. MRMC performs probability estimation by the *Confidence interval* method that is based on the Chow–Robbins test (Section 4.1), with a *dynamic sample size*. The only problem is that since MRMC always loads Markov chain representation completely in memory, it can lose the benefits deriving from simulating instead of using numerical techniques.

5.11. SBIP

SBIP [83] is a statistical model checker for the BIP (Behavior, Interaction, Priority) general framework for the design of component-based systems that were developed at Verimag (see [84]).

Model and property. It supports DTMC, CTMC, and GSMP (Section 3.1) as the input models. The properties to be verified can be expressed as PBLTL and MTL formula (Section 3.2).

Statistical Inference approach. The tool implements several statistical testing algorithms for stochastic systems verification of both qualitative and quantitative properties. The qualitative properties are checked through one of the following algorithms: *Single Sampling Plan (SSP)* [3], where the number of samples is predetermined, and SPRT, where the number of samples is generated at runtime (Section 4.1). The quantitative properties are verified through a *Probability Estimation* procedure, based on the Chernoff–Hoeffding bound (Section 4.1).

5.12. MARCIE

MARCIE [85] is a tool that is written in C++ and made up of several analysis engines, in particular a Stochastic Simulation Engine that is based on statistical estimation.

Model and property. This tool can verify both quantitative and qualitative properties over systems that are modelled as GSPN, including CTMC (Section 3.1). Properties can be defined by CSL, Continuous Stochastic Reward Logic (CSRL) or PLTLc. CSRL includes CSL and adds reward intervals to the temporal operators (Section 3.2).

Statistical Inference approach. The component of MARCIE that is dedicated to estimation implements an algorithm performing several simulation runs depending on the variance of the system stochastic behavior and determined through a *Confidence interval* method that is based on the Chernoff–Hoeffding bound (Section 4.1).

5.13. Modest Toolset Discrete Event Simulator: `Modes`

Modest [86] is a high-level compositional modelling language for STA. It is completely supported by the Modest Toolset, available at [87], which includes tools for analyzing different subsets of the STA, e.g., PTA, MDP, DTMC, CTMC (Section 3.1). In particular, the `modes` tool, a discrete event simulator, which is based on the Modest language, is available. From version 1.4 it offers SMC functionalities.

Model and property. `modes` supports the analysis of SHA, STA, PTA, and MDP. THe properties to be verified are expressed in Modest language.

Statistical Inference approach. `modes` verifies quantitative properties through a *confidence interval* based algorithm, which allows for deciding at runtime how many simulations to do; qualitative properties through the SPRT method (Section 4.1).

5.14. APD Analyser

Aggregated Power Demand (APD) Analyser ([88,89]) is a SMC based *domain-specific* tool to compute the APD probability distribution in a smart grid scenario in the presence of highly dynamic individualised electricity price policies [90,91].

Model and property. The input model consists of a probabilistic model of end-user deviations with respect to their expected behaviors. The tool computes a single *domain-specific* property: the APD probability distribution in Electric Distribution Networks. Further post-processing of this distribution allows for the Distribution System Operators (DSO) to compute the safety properties of interest.

Statistical Inference approach. As an exact computation of the required APD probability distribution is computationally prohibitive, because of its exponential dependence on the number of users, APD-Analyser computes Monte Carlo based (ϵ, δ)-approximation, through an efficient *High-Performance Computing (HPC)*-based implementation of the $\mathcal{OAA}$ algorithm, discussed in Section 4.2.

5.15. ViP Generator

The works in [92–94] present a tool, called *ViP* (Virtual Patient generator). This tool uses a SMC based approach for computing complete populations of virtual patients for *in silico* clinical trials and the model-based synthesis of optimal personalised pharmaceutical treatments [95,96].

Model and property. The input model is a system of ODEs and the boolean property to be satisfied is the completeness of the virtual patient set generated.

Statistical Inference approach. HT (Section 4.1) is used to check, with high statistical confidence, the completeness of the virtual patient set S generated so far. After defining a probability threshold ϵ and a confidence threshold δ, the SMC algorithm in [92] randomly extracts, from a parameter space Λ, a sample $\{\lambda\}$ that, if admissible, is added to S. On the basis of [52,53], the algorithm ends when S remains unchanged after $N = \ln \delta / \ln(1 - \epsilon)$ attempts.

5.16. SAM

SAM (Stochastic Analyzer for Mobility) [36] is a tool supporting the analysis of mobile and distributed systems. SAM uses a statistical model checking algorithm in order to verify whether a mobile system satisfies a certain property.

Model and Properties. Systems are specified in StoKLAIM (Section 3.1). The properties to be verified are expressed through MoSL (Section 3.2).

Statistical Inference approach. SAM performs the estimation of quantitative properties.

5.17. Bayesian Tool

The Bayesian analysis consists of the introduction of Bayesian statistics to the SMC approach, as discussed in Section 4.3.

Model and property. The models to be analyzed are DTHS (see Section 2) defined as Stateflow/Simulink models, while properties are expressed as BLTL formula.

Statistical Inference approach. The Bayesian analysis includes: (i) an algorithm to perform SMC using *Bayesian hypothesis testing* in order to verify BLTL properties against Stateflow/Simulink models with hybrid features; (ii) a *Bayesian estimation* algorithm, to compute an interval estimate of the probability that a BLTL formula is satisfied in a stochastic hybrid system model.

5.18. Tool Comparison

In Table 2, we show a taxonomy of the surveyed tools, which is based on their main properties, namely: the environment model, characterized by the time (discrete or continue), the set of events values (that can be finite/infinite); the SUV model, consisting of the kind of models or language used to represent the system and the set of states of the model (that can be finite/infinite); the property specification, which is the stochastic logic used to specify the properties to be verified and the search horizon (that can be bounded/unbounded); the verification technique, consisting of the statistical inference procedure used (HT and/or Estimation and/or Bayesian analysis), together with the algorithm implemented to perform the inference. The algorithm is useful to know whether the number of simulation runs is fixed beforehand or not (as shown in Table 1).

In addition to the tools extensively that were reviewed in [4,5], we have included the Bayesian analysis and two other SMC domain-specific software tools, the *APD Analyser* and the *ViP* (Virtual Patient) *generator*.

Table 2. Monte Carlo simulation-based SMC tools comparison table. In the *Time* column *D* stands for discrete, *C* for continue. In the column *Model*, the model is specified as its representation structure or as the language describing the model. In the columns *Event values* and *Set of states*, fin means *finite* and inf means *infinite*. In the *Search horizon* column, *bnd* stands for *bounded*, *ubnd* for *unbounded*. In the *Inference* column, *HT* stands for *Hypothesis Testing*; *E* stands for *Estimation* and *NS* stands for *Numeric-symbolic* methods (see Section 3). In the same column, for each inference approach, the name of the algorithm used is specified. For further details about the algorithms, see Table 1. Depending on the algorithm, the number of samples can be computed a priori or at runtime.

	ENVIRONMENT MODEL		**SUV MODEL**		**SPECIFICATION**		**VERIFICATION TECHNIQUE**	
TOOL	**Time**	**Event Values**	**Model**	**Set of States**	**Property Language**	**Search Horizon**	**Inference**	**#samples Computing**
(P)VeStA [63]	C	fin	CTMC, DTMC/ PMaude	fin/inf	CSL, PCTL, QuaTEx	ubnd	HT: Gauss-SSP; E: C.I.	HT and E: a priori
MultiVeStA [65]	D/C	fin	DES	inf	MultiQuaTEx	ubnd	E: Chow-Robbins	E: at runtime
Plasma [69]	C	fin/inf	CTMC, MDP	inf	BLTL	bnd	HT: SPRT; E: Chernoff C.I.	HT: at runtime; E: a priori
APMC [73,97]	D	inf	DTMC, CTMC	fin	LTL	ubnd on monotone LTL	E: Chernoff-SSP/$\mathcal{OAA}$	E: a priori/at runtime
PRISM [15,75]	D/C	fin	DTMC, CTMC, MDP	fin	BLTL	ubnd	HT: SPRT; E: Chernoff C.I.; NS	HT: at runtime; E: a priori
Ymer [77,78]	D/C	inf	DTMC, CTMC	fin	PCTL, CSL	ubnd	HT: SPRT/Gauss-SSP; E: Chow-Robbins/ Chernoff C.I.	HT: at runtime/a priori; E: at runtime/a priori; NS
UPPAAL-SMC [79]	C	inf	SHA	inf	MITL	bnd	HT: SPRT; E: Chernoff C.I.	HT: at runtime; E: a priori
COSMOS [30]	C	fin	GSPN	fin	HASL	ubnd	HT: SPRT; E: Chernoff C.I. / Chow-Robbins	HT: at runtime; E: a priori/at runtime
MRMC [82]	D/C	fin	DTMC, CTMC	fin	PCTL, CSL	ubnd	E: Chow-Robbins; NS	E: at runtime
SBIP [83]	D/C	inf	DTMC, CTMC, GSMP	inf	PBLTL	bnd	HT: SPRT; E: Chernoff C.I.	HT: at runtime; E: a priori
MARCIE [85]	C	fin	GSPN	fin	CSL, CSRL, PCTL	ubnd	E: Chernoff C.I.; NS	E: at runtime
`modes` [86]	C	fin	SHA, STA, PTA, MDP	fin	MODEST	ubnd	HT: SPRT; E: Chernoff C.I.	HT: at runtime; E: a priori
APD Analyser [89]	D	fin	Custom model	inf	Safety properties	bnd	E: $\mathcal{OAA}$	E: at runtime
ViP generator [92,93]	D	fin	ODEs	inf	Boolean properties	bnd	HT: SPRT	HT: at runtime
Bayesian tools [7,56]	D	fin	DTHS, Uncertain CTMC	inf	BLTL, MTL,MITL	bnd	HT: Bayesian HT; E: Bayesian E	HT and E: at runtime

6. Discussion

The taxonomy shown in Table 2 highlights the following trade-offs.

The *Environment model* is: *complete* if it contains all possible operational scenarios; *sound* if it contains only the operational scenarios that can occur in a real situation. Ideally, we would like a complete and sound environment model. However, in practice, this is not easy to define. Depending on priorities, correctness, or efficiency, we may select a complete (possibly unsound) model or a sound (possibly incomplete) model.

A *SUV model* has to capture the dynamics of the system. If the properties to be verified are of the *safety* kind, tools taking as input an *over-approximated* model (i.e., containing more behaviors than the real system) have to be preferred. If *liveness* properties have to be verified, tools accepting *under-approximated* models can be used.

The property language has to be selected with respect to its capability to define the property to be verified. The search horizon should be *large enough*, but not *too large*, in order to avoid making simulations overly time consuming. In this case, the trade-off is between the expressiveness and the computational complexity of the verification activity.

The verification technique selection depends on the goal (namely, estimation or HT) and on the need to know beforehand, or not, the number of samples to be generated. Accordingly, all of the tools implement the SPRT to perform HT on qualitative statements, except: (P)VeStA that uses the Gauss–SSP test, through which the sample size is predetermined; the Bayesian analysis, which bounds the sample size while using a test based on the Bayes factor. SBIP, besides SPRT, also implements also SSP.

The estimation of quantitative properties is performed through different techniques. (P)VeSta uses the Confidence Interval method to evaluate QuaTEx formulae; MultiVeStA, COSMOS (from version 1.0) and MRMC (from version 1.5) implement the Chow-Robbins test; Plasma Lab, UPPAAL-SMC, PRISM (from version 4.0), SBIP, and MARCIE use the Chernoff–C.I. method; Ymer uses the Gauss-SSP test; APMC (from v3.0) implements the so-called Chernoff–SSP test; the APD Analyser implements the $\mathcal{OAA}$ algorithm (see Table 1). The Bayesian Estimation procedure is different from all of the others, because it is based on the Bayes factor test, but, like with the Chow Robbins test, the sample size is not fixed *a priori*.

If we have to choose a Monte Carlo based SMC tool minimizing the number of simulations to do, in the case of qualitative statements, all of the tools implementing the SPRT to solve HT are suitable. On the other hand, an a priori fixed sample size technique is easily parallelizable, then sometimes this kind of approach could be preferable. In the case of quantitative statements, the tools implementing the Chow–Robbins test or $\mathcal{OAA}$ are the fastest. The Bayesian analysis also allows for us to decide at runtime how many simulations to do, but it needs to know the probability distribution of the variables to be estimated in advance.

As a last result, Table 2 highlights an open research challenge, which is the lack of tools to perform an unbounded verification of hybrid systems whose environment has continuous- or discrete-time.

7. Conclusions

We have reviewed most of the SMC tools currently available for research purposes, focusing on the complexity, in terms of sample size, of the Monte Carlo-based techniques that are used to evaluate quantitative and qualitative properties through *HT*, *Estimation*, and *Bayesian analysis*.

For each tool, we have highlighted: the environment model; the SUV model; the stochastic logic that is used to define the properties to be verified, together with the search horizon type; the statistical inference procedure; and, the corresponding algorithm used, by explicitly indicating whether the number of samples is generated at runtime or not. After an in-depth comparison, we produced the taxonomy presented in Table 2.

First, we observe that most of the tools assume that the environment is *sound*, thus events are taken from a finite set. In this scenario, the SUV model set of states is typically finite and the search horizon for the properties to be checked is *unbounded*. Otherwise, if the SUV model set of states is infinite, then the search horizon is always *bounded*.

Second, 90% of the tools performing *HT* use algorithms that decide, at runtime, the number of samples to be generated; approximately 70% of the tools implementing *Estimation* employ algorithms computing the number of samples beforehand. Thus, according to the problem at hand, we suggest using SMC tools computing *a priori* the number of samples if there is a need to parallelize and, instead, using tools generating one sample at each iteration if it is more important to reduce the number of simulations.

Finally, our taxonomy points out the challenging task of deploying tools to perform unbounded verification of discrete- or continuous-time hybrid systems.

Author Contributions: Conceptualization, A.P., A.M. and E.T.; formal analysis, A.P. and E.T.; writing—original draft preparation, A.P.; writing—review and editing, A.P., A.M. and E.T.; supervision, E.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the following research projects/grants: Italian Ministry of University & Research (MIUR) grant *Dipartimenti di eccellenza 2018–2022* (Dept. Computer Science, Sapienza Univ. of Rome); EC FP7 project PAEON (Model Driven Computation of Treatments for Infertility Related Endocrinological Diseases, 600773); INdAM "GNCS Project 2019"; POR FESR 2014–2020 project SCAPR (Sistema per il Contrasto di Aeromobili a Pilotaggio Remoto).

Acknowledgments: We thank Alberto Lluch Lafuente for his very useful remarks on a preliminary version of this paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Alur, R. *Principles of Cyber-Physical Systems*; MIT Press: Cambridge, MA, USA, 2015.
2. Clarke, E.; Wing, J.M. Formal Methods: State of the Art and Future Directions. *Comput. Surv. (CSUR)* **1996**, *28*, 626–643. [CrossRef]
3. Legay, A.; Delahaye, B.; Bensalem, S. Statistical Model Checking: An Overview. In *Runtime Verification, First International Conference, RV 2010, St. Julians, Malta, November 2010. Proceedings*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6418, pp. 122–135. [CrossRef]
4. Agha, G.; Palmskog, K. A Survey of Statistical Model Checking. *ACM Trans. Model. Comput. Simul.* **2018**, *28*, 6:1–6:39. [CrossRef]
5. Reijsbergen, D.; de Boer, P.T.; Scheinhardt, W.; Haverkort, B. On hypothesis testing for statistical model checking. *Int. J. Softw. Tools Technol. Transf.* **2015**, *17*, 377–395. [CrossRef]
6. Bakir, M.; Gheorghe, M.; Konur, S.; Stannett, M. Comparative Analysis of Statistical Model Checking Tools. In Proceedings of the Membrane Computing: 17th International Conference (CMC 2016), Milan, Italy, 25–29 July 2016. [CrossRef]
7. Zuliani, P.; Platzer, A.; Clarke, E. Bayesian Statistical Model Checking with Application to Stateflow/Simulink Verification. *Form. Methods Syst. Des.* **2013**, *43*, 338–367. [CrossRef]
8. Devroye, L. *Non-Uniform Random Variate Generation*; Springer: Berlin/Heidelberg, Germany, 1986.
9. Simulink. Available online: http://www.mathworks.com (accessed on 18 December 2020).
10. Dymola. Available online: http://www.claytex.com/products/dymola/ (accessed on 18 December 2020).
11. SimulationX. Available online: http://www.simulationx.com (accessed on 18 December 2020).
12. Wolfram Research, Inc. SystemModeler. Available online: http://www.wolfram.com/system-modeler (accessed on 18 December 2020).
13. Zhou, F.; Chen, L.; Wu, Y.; Ding, J.; Zhao, J.; Zhang, Y. MWorks : A Modern IDE for Modeling and Simulation of Multi-domain Physical Systems Based on Modelica. In Proceedings of the 5th International Modelica Conference (Modelica 2006), Vienna, Austria, 4–5 September 2006.
14. OpenModelica. Available online: http://www.openmodelica.org (accessed on 18 December 2020).

15. Kwiatkowska, M.; Norman, G.; Parker, D. PRISM 4.0: Verification of Probabilistic Real-time Systems. In Proceedings of the 23rd International Conference on Computer Aided Verification (CAV 2011), Snowbird, UT, USA, 14–20 July 2011; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6806, pp. 585–591.
16. McMillan, K.; The SMV System. In *Symbolic Model Checking*; Springer: Berlin/Heidelberg, Germany, 1993; pp. 61–85.
17. Baier, C.; Katoen, J.P. *Principles of Model Checking (Representation and Mind Series)*; MIT Press: Cambridge, MA, USA, 2008.
18. Sontag, E. *Mathematical Control Theory: Deterministic Finite Dimensional Systems*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 1998.
19. Cellier, F.; Kofman, E. *Continuous System Simulation*; Springer: Berlin/Heidelberg, Germany, 2010.
20. Pinisetty, S.; Jéron, T.; Tripakis, S.; Falcone, Y.; Marchand, H.; Preoteasa, V. Predictive runtime verification of timed properties. *J. Syst. Softw.* **2017**, *132*, 353–365. [CrossRef]
21. Thati, P.; Roşu, G. Monitoring Algorithms for Metric Temporal Logic Specifications. In *Runtime Verification, Fourth Workshop on Runtime Verification 2004, RV 2004, Barcelona, Spain, April 2004. Proceedings*; Electronic Notes in Theoretical Computer Science; Elsevier: Amsterdam, The Netherlands, 2004; Volume 113, pp. 145–162. [CrossRef]
22. Bauer, A.; Leucker, M.; Schallhart, C. Runtime Verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.* **2011**, *20*. [CrossRef]
23. Abbas, H.; Fainekos, G.; Sankaranarayanan, S.; Ivančić, F.; Gupta, A. Probabilistic Temporal Logic Falsification of Cyber-Physical Systems. *ACM Trans. Embed. Comput. Syst.* **2013**, *12*, 95:1–95:30. [CrossRef]
24. Katoen, J. The Probabilistic Model Checking Landscape. In *31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2016)*; Association for Computing Machinery: New York, NY, USA, 2016; pp. 31–45. [CrossRef]
25. Younes, H.; Kwiatkowska, M.; Norman, G.; Parker, D. Numerical vs. Statistical Probabilistic Model Checking. *Int. J. Softw. Tools Technol. Transf.* **2006**, *8*, 216–228. [CrossRef]
26. Baier, C.; Haverkort, B.; Hermanns, H.; Katoen, J.P. Model-checking algorithms for continous-time markov chains. *IEEE Trans. Softw. Eng.* **2003**, *29*, 524–541. [CrossRef]
27. Younes, H.; Simmons, R. Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling. In Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002), Copenhagen, Denmark, 27–31 July 2002; Springer: Berlin/Heidelberg, Germany, 2002; Volume 2404, Lecture Notes in Computer Science; pp. 223–235. [CrossRef]
28. Sen, K.; Viswanathan, M.; Agha, G. Statistical model checking of black-box probabilistic systems. In Proceedings of the 16th International Conference on Computer Aided Verification (CAV 2004), Boston, MA, USA, 13–17 July 2004; Springer: Berlin/Heidelberg, Germany, 2004; Volume 3114, Lecture Notes in Computer Science; pp. 202–213.
29. Whitt, W. Continuity of Generalized Semi-Markov Processes. *Math. Oper. Res.* **1980**, *5*, 494–501. [CrossRef]
30. Ballarini, P.; Barbot, B.; Duflot, M.; Haddad, S.; Pekergin, N. HASL: A New Approach for Performance Evaluation and Model Checking from Concepts to Experimentation. *Perform. Eval.* **2015**, *90*, 53–77. [CrossRef]
31. Norman, G.; Parker, D.; Sproston, J. Model checking for probabilistic timed automata. *Form. Methods Syst. Des.* **2013**, *43*, 164–190. [CrossRef]
32. David, A.; Du, D.; Larsen, K.; Legay, A.; Mikučionis, M.; Poulsen, D.; Sedwards, S. Statistical Model Checking for Stochastic Hybrid Systems. *Electron. Proc. Theor. Comput. Sci.* **2012**, *92*, 122–136. [CrossRef]
33. Legay, A.; Sedwards, S.; Traonouez, L. Scalable Verification of Markov Decision Processes. In *Software Engineering and Formal Methods*; Springer: Berlin/Heidelberg, Germany, 2015; pp. 350–362. [CrossRef]
34. Puterman, M. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*; John Wiley & Sons: Hoboken, NJ, USA, 2005.
35. Agha, G.; Meseguer, J.; Sen, K. PMaude: Rewrite-based Specification Language for Probabilistic Object Systems. In *Proceedings of the 3rd Workshop on Quantitative Aspects of Programming Languages (QAPL 2005)*; Elsevier: Amsterdam, The Netherlands, 2005.
36. De Nicola, R.; Katoen, J.; Latella, D.; Loreti, M.; Massink, M. Model checking mobile stochastic logic. *Theor. Comput. Sci.* **2007**, *382*, 42–70. [CrossRef]

37. De Nicola, R.; Ferrari, G.L.; Pugliese, R. KLAIM: A kernel language for agents interaction and mobility. *IEEE Trans. Softw. Eng.* **1998**, *24*, 315–330. [CrossRef]
38. Rozier, K. Linear Temporal Logic Symbolic Model Checking. *Comput. Sci. Rev.* **2011**, *5*, 163–203. [CrossRef]
39. Mediouni, B.; Nouri, A.; Bozga, M.; Dellabani, M.; Legay, A.; Bensalem, S. SBIP 2.0: Statistical Model Checking Stochastic Real-Time Systems. In Proceedings of the 16th International Symposium on Automated Technology for Verification and Analysis (ATVA 2018), Los Angeles, CA, USA, 7–10 October 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 536–542.
40. Alur, R.; Feder, T.; Henzinger, T. The Benefits of Relaxing Punctuality. *J. ACM* **1996**, *43*, 116–146. [CrossRef]
41. Clarke, E.; Henzinger, T.; Veith, H. *Handbook of Model Checking*; Springer: Berlin/Heidelberg, Germany, 2016.
42. Hansson, H.; Jonsson, B. A logic for reasoning about time and reliability. *Form. Asp. Comput.* **1994**, *6*, 512–535. [CrossRef]
43. Sen, K.; Viswanathan, M.; Agha, G. On Statistical Model Checking of Stochastic Systems. In Proceedings of the 17th International Conference on Computer Aided Verification (CAV 2005), Edinburgh, UK, 6–10 July 2005; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3576, pp. 266–280.
44. Baier, C. On Algorithmic Verification Methods for Probabilistic Systems. Ph.D. Thesis, University of Mannheim, Mannheim, Germany, 1998.
45. Donatelli, S.; Haddad, S.; Sproston, J. Model Checking Timed and Stochastic Properties with CSL^{TA}. *IEEE Trans. Softw. Eng.* **2009**, *35*, 224–240. [CrossRef]
46. Hoeffding, W. Probability Inequalities for Sums of Bounded Random Variables. *J. Am. Stat. Assoc.* **1963**, 13–30. [CrossRef]
47. Wald, A. Sequential tests of statistical hypotheses. *Ann. Math. Stat.* **1945**, *16*, 117–186. [CrossRef]
48. Younes, H. Verification and Planning for Stochastic Processes with Asynchronous Events. Ph.D. Thesis, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA, USA, 2005.
49. Jegourel, C.; Legay, A.; Sedwards, S. Command-based importance sampling for statistical model checking. *Theor. Comput. Sci.* **2016**, *649*, 1–24. [CrossRef]
50. Jegourel, C.; Legay, A.; Sedwards, S. Importance Splitting for Statistical Model Checking Rare Properties. In Proceedings of the 25th International Conference on Computer Aided Verification (CAV 2013), Saint Petersburg, Russia, 13–19 July 2013; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2013; Volume 8044, pp. 576–591.
51. Legay, A.; Lukina, A.; Traonouez, L.; Yang, J.; Smolka, S.; Grosu, R. Statistical Model Checking. In *Computing and Software Science: State of the Art and Perspectives*; Springer Nature: Berlin/Heidelberg, Germany, 2019; pp. 478–504. [CrossRef]
52. Grosu, R.; Smolka, S. Quantitative Model checking. In Proceedings of the 1st International Symposium on Leveraging Applications of Formal Method (ISoLA 2004), Paphos, Cyprus, 30 October–2 November 2004; pp. 165–174.
53. Grosu, R.; Smolka, S. Monte Carlo Model Checking. In Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2005), Edinburgh, UK, 4–8 April 2005; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3440, Lecture Notes in Computer Science; pp. 271–286. [CrossRef]
54. Dagum, P.; Karp, R.; Luby, M.; Ross, S.M. An Optimal Algorithm for Monte Carlo Estimation. *SIAM J. Comput.* **2000**, *29*, 1484–1496. [CrossRef]
55. Jha, S.; Clarke, E.; Langmead, C.; Legay, A.; Platzer, A.; Zuliani, P. A Bayesian Approach to Model Checking Biological Systems. In Proceedings of the 7th International Conference on Computational Methods in Systems Biology (CMSB 2009), Bologna, Italy, 31 August–1 September 2009; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5688, pp. 218–234. [CrossRef]
56. Bortolussi, L.; Milios, D.; Sanguinetti, G. Smoothed model checking for uncertain Continuous-Time Markov Chains. *Inf. Comput.* **2016**, *247*, 235–253. [CrossRef]
57. Mancini, T.; Mari, F.; Massini, A.; Melatti, I.; Tronci, E. SyLVaaS: System Level Formal Verification as a Service. In Proceedings of the 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2015), Turku, Finland, 4–6 March 2015; pp. 476–483.

58. Annpureddy, Y.; Liu, C.; Fainekos, G.E.; Sankaranarayanan, S. S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. In Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2011), Saarbrücken, Germany, 26 March–3 April 2011; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6605, pp. 254–257. [CrossRef]
59. Bresolin, D.; Collins, P.; Geretti, L.; Segala, R.; Villa, T.; Gonzalez, S. A Computable and Compositional Semantics for Hybrid Automata. In Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control (HSCC 2020), Sydney, Australia, 21–24 April 2020; ACM: New York, NY, USA, 2020. [CrossRef]
60. Frehse, G.; Le Guernic, C.; Donzé, A.; Cotton, S.; Ray, R.; Lebeltel, O.; Ripado, R.; Girard, A.; Dang, T.; Maler, O. SpaceEx: Scalable Verification of Hybrid Systems. In Proceedings of the 23rd International Conference on Computer Aided Verification (CAV 2011), Snowbird, UT, USA, 14–20 July 2011; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6806, pp. 379–395.
61. Luckow, K.; Păsăreanu, C.; Dwyer, M.; Filieri, A.; Visser, W. Exact and Approximate Probabilistic Symbolic Execution for Nondeterministic Programs. In Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering (ASE 2014), Vsters, Sweden, 15–19 September 2014; ACM: New York, NY, USA, 2014.
62. Hogg, R.; McKean, J.W.; Craig, A.T. *Introduction to Mathematical Statistics*, 8th ed.; Pearson Education: Upper Saddle River, NJ, USA, 2018.
63. Sen, K.; Viswanathan, M.; Agha, G. VeStA: A statistical model-checker and analyzer for probabilistic systems. In Proceedings of the QEST 2005—Proceedings Second International Conference on the Quantitative Evaluation of SysTems, Torino, Italy, 19–22 September 2005; Volume 2005, pp. 251–252. [CrossRef]
64. AlTurki, M.; Meseguer, J. PVeStA: A Parallel Statistical Model Checking and Quantitative Analysis Tool. In Proceedings of the 4th International Conference on Algebra and Coalgebra in Computer Science (CALCO 2011), Winchester, UK, 30 August–2 September 2011; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6859, pp. 386–392.
65. Sebastio, S.; Vandin, A. MultiVeStA: Statistical Model Checking for Discrete Event Simulators. In Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools (ValueTools 2013), Torino, Italy, 10–12 December 2013; ICST/ACM: New York, NY, USA, 2013, pp. 310–315.
66. Shmarov, F.; Zuliani, P. Probabilistic Hybrid Systems Verification via SMT and Monte Carlo Techniques. In Proceedings of the Hardware and Software: Verification and Testing, 12nd International Haifa Verification Conference (HVC 2016), Haifa, Israel, 14–17 November 2016; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2016; Volume 10028. [CrossRef]
67. Xue, B.; Fränzle, M.; Zhao, H.; Zhan, N.; Easwaran, A. Probably Approximate Safety Verification of Hybrid Dynamical Systems. In Proceedings of the Formal Methods and Software Engineering—21st International Conference on Formal Engineering Methods (ICFEM 2019), Shenzhen, China, 5–9 November 2019; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11852. [CrossRef]
68. Xue, B.; Liu, Y.; Ma, L.; Zhang, X.; Sun, M.; Xie, X. Safe Inputs Approximation for Black-Box Systems. In Proceedings of the 24th International Conference on Engineering of Complex Computer Systems (ICECCS 2019), Guangzhou, China, 10–13 November 2019; pp. 180–189. [CrossRef]
69. Plasma Lab. Available online: https://project.inria.fr/plasma-lab/ (accessed on 18 December 2020).
70. Jegourel, C.; Legay, A.; Sedwards, S. A Platform for High Performance Statistical Model Checking–PLASMA. In Proceedings of the 18th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2012), Tallinn, Estonia, 24 March–1 April 2012; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7214, pp. 498–503.
71. Boyer, B.; Corre, K.; Legay, A.; Sedwards, S. PLASMA-lab: A Flexible, Distributable Statistical Model Checking Library. In Proceedings of the 10th International Conference on Quantitative Evaluation of Systems (QEST 2013), Buenos Aires, Argentina, 27–30 August 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 160–164.
72. Hérault, T.; Lassaigne, R.; Magniette, F.; Peyronnet, S. Approximate Probabilistic Model Checking. In Proceedings of the 5th International Workshop on Verification, Model Checking, and Abstract Interpretation (VMCAI 2004), Venice, Italy, 11–13 January 2004; Springer: Berlin/Heidelberg, Germany, 2004; pp. 73–84. [CrossRef]

73. Lassaigne, R.; Peyronnet, S. Probabilistic verification and approximation. *Ann. Pure Appl. Log.* **2008**, *152*, 122–131. [CrossRef]
74. Peyronnet, S.; Lassaigne, R.; Herault, T. APMC 3.0: Approximate Verification of Discrete and Continuous Time Markov Chains. In Proceedings of the QEST 2006—Proceedings Third International Conference on the Quantitative Evaluation of SysTems, Riverside, CA, USA, 11–14 September 2006; pp. 129–130. [CrossRef]
75. Henriques, D.; Martins, J.; Zuliani, P.; Platzer, A.; Clarke, E. Statistical Model Checking for Markov Decision Processes. In Proceedings of the 2012 Ninth International Conference on Quantitative Evaluation of Systems, London, UK, 17–20 September 2012; pp. 84–93. [CrossRef]
76. Parker, D.; Norman, G.; Kwiatkowska, M. PRISM 2017. Statistical Model Checker. Available online: https://www.prismmodelchecker.org/manual/RunningPRISM/StatisticalModelChecking (accessed on 18 December 2020).
77. Younes, H. Ymer: A Statistical Model Checker. In Proceedings of the 17th International Conference on Computer Aided Verification (CAV 2005), Edinburgh, UK, 6–10 July 2005; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2005; Volume 3576, pp. 429–433. [CrossRef]
78. Younes, H.; Clarke, E.; Zuliani, P. Statistical Verification of Probabilistic Properties with Unbounded Until. In Proceedings of the 13th Brazilian Symposium on Formal Methods (SBMF 2010), Natal, Brazil, 8–11 November 2010; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2010; Volume 6527. [CrossRef]
79. David, A.; Larsen, K.; Legay, A.; Mikučionis, M.; Poulsen, D. Uppaal SMC tutorial. *Int. J. Softw. Tools Technol. Transf.* **2015**, *17*, 397–415. [CrossRef]
80. Bengtsson, J.; Larsen, K.; Larsson, F.; Pettersson, P.; Yi, W. UPPAAL—A Tool Suite for Automatic Verification of Real-Time Systems. In *Hybrid Systems III: Verification and Control*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1996; Volume 1066, pp. 232–243. [CrossRef]
81. Amparore, E.G.; Beccuti, M.; Donatelli, S. (Stochastic) Model Checking in GreatSPN. In Proceedings of the Applications and Theory of Petri Nets and Concurrency (PETRI NETS 2014), Tunis, Tunisia, 23–27 June 2014; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2014; Volume 8489, pp. 354–363. [CrossRef]
82. Katoen, J.P.; Zapreev, I.S.; Hahn, E.M.; Hermanns, H.; Jansen, D.N. The ins and outs of the probabilistic model checker MRMC. *Perform. Eval.* **2011**, *68*, 90–104. [CrossRef]
83. Nouri, A.; Mediouni, B.; Bozga, M.; Combaz, J.; Bensalem, S.; Legay, A. Performance Evaluation of Stochastic Real-Time Systems with the SBIP Framework. *Int. J. Crit. Comput. Based Syst.* **2018**, 1–33. [CrossRef]
84. Verimag. BIP Component Framework. Available online: http://www-verimag.imag.fr/Rigorous-Design-of-Component-Based.html (accessed on 18 December 2020).
85. Heiner, M.; Rohr, C.; Schwarick, M. MARCIE—Model Checking and Reachability Analysis Done Efficiently. In Proceedings of the Applications and Theory of Petri Nets and Concurrency (PETRI NETS 2013), Milan, Italy, 24–28 June 2013; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2013; Volume 7927, pp. 389–399. [CrossRef]
86. Bogdoll, J.; Hartmanns, A.; Hermanns, H. Simulation and Statistical Model Checking for Modestly Nondeterministic Models. In Proceedings of the Measurement Modelling and Evaluation of Computing Systems and Dependability and Fault Tolerance (MMB&DFT 2012), Kaiserslautern, Germany, 19–21 March 2012; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2012; pp. 249–252. [CrossRef]
87. MODEST. Available online: http://www.modestchecker.net (accessed on 18 December 2020).
88. Mancini, T.; Mari, F.; Melatti, I.; Salvo, I.; Tronci, E.; Gruber, J.; Hayes, B.; Prodanovic, M.; Elmegaard, L. Demand-Aware Price Policy Synthesis and Verification Services for Smart Grids. In Proceedings of the 2014 IEEE International Conference on Smart Grid Communications (SmartGridComm 2014), Venice, Italy, 3–6 November 2014; pp. 794–799. [CrossRef]
89. Mancini, T.; Mari, F.; Melatti, I.; Salvo, I.; Tronci, E.; Gruber, J.; Hayes, B.; Elmegaard, L. Parallel Statistical Model Checking for Safety Verification in Smart Grids. In Proceedings of the 2018 IEEE International Conference on Smart Grid Communications (SmartGridComm 2018), Aalborg, Denmark, 29–31 October 2018. [CrossRef]

90. Mancini, T.; Mari, F.; Melatti, I.; Salvo, I.; Tronci, E.; Gruber, J.; Hayes, B.; Prodanovic, M.; Elmegaard, L. User Flexibility Aware Price Policy Synthesis for Smart Grids. In Proceedings of the 18th Euromicro Conference on Digital System Design (DSD 2015), Funchal, Portugal, 26–28 August 2015; pp. 478–485. [CrossRef]
91. Hayes, B.; Melatti, I.; Mancini, T.; Prodanovic, M.; Tronci, E. Residential Demand Management using Individualised Demand Aware Price Policies. *IEEE Trans. Smart Grid* **2017**, *8*. [CrossRef]
92. Tronci, E.; Mancini, T.; Salvo, I.; Sinisi, S.; Mari, F.; Melatti, I.; Massini, A.; Davi', F.; Dierkes, T.; Ehrig, R.; Röblitz, S.; Leeners, B.; Krüger, T.; Egli, M.; Ille, F. Patient-Specific Models from Inter-Patient Biological Models and Clinical Records. In Proceedings of the 14th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2014), Lausanne, Switzerland, 21–24 October 2014; pp. 207–214. [CrossRef]
93. Mancini, T.; Tronci, E.; Salvo, I.; Mari, F.; Massini, A.; Melatti, I. Computing Biological Model Parameters by Parallel Statistical Model Checking. In Proceedings of the 3rd International Conference on Bioinformatics and Biomedical Engineering (IWBBIO 2015), Granada, Spain, 15–17 April 2015; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9044, pp. 542–554. [CrossRef]
94. Sinisi, S.; Alimguzhin, V.; Mancini, T.; Tronci, E.; Leeners, B. Complete populations of virtual patients for in silico clinical trials. *Bioinformatics* **2020**, to appear. [CrossRef]
95. Mancini, T.; Mari, F.; Massini, A.; Melatti, I.; Salvo, I.; Sinisi, S.; Tronci, E.; Ehrig, R.; Röblitz, S.; Leeners, B. Computing Personalised Treatments through In Silico Clinical Trials. A Case Study on Downregulation in Assisted Reproduction. In Proceedings of the 25th RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA 2018), Oxford, UK, 13 July 2018.
96. Sinisi, S.; Alimguzhin, V.; Mancini, T.; Tronci, E.; Mari, F.; Leeners, B. Optimal Personalised Treatment Computation through In Silico Clinical Trials on Patient Digital Twins. *Fundam. Inform.* **2020**, *174*, 283–310. [CrossRef]
97. Guirado, G.; Hérault, T.; Lassaigne, R.; Peyronnet, S. Distribution, Approximation and Probabilistic Model Checking. In Proceedings of the 4th International Workshop on Parallel and Distributed Methods in Verification (PDMC 2005), Lisboa, Portugal, 10 July 2005; Elsevier: Amsterdam, The Netherlands, 2006; Volume 135, pp. 19–30. [CrossRef]

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.