Any-Horizon Uniform Random Sampling and Enumeration of Constrained Scenarios for Simulation-Based Formal Verification

Toni Mancini, Igor Melatti, and Enrico Tronci

Abstract—*Model-based* approaches to the verification of non-terminating Cyber-Physical Systems (CPSs) usually rely on *numerical simulation* of the System Under Verification (SUV) model under input scenarios of possibly varying duration, chosen among those satisfying given *constraints*. Such constraints typically stem from *requirements* (or *assumptions*) on the SUV inputs and its *operational environment* as well as from the enforcement of *additional conditions* aiming at, e.g., *prioritising* the (often extremely long) verification activity, by, e.g., focusing on scenarios explicitly exercising *selected* requirements, or avoiding *vacuity* in their satisfaction. In this setting, the possibility to *efficiently sample at random* (with a known distribution, e.g., uniformly) within, or to efficiently *enumerate* (possibly in a uniformly random order) scenarios among those satisfying all the given constraints is a key enabler for the practical viability of the verification process, e.g., via simulation-based statistical model checking. Unfortunately, in case of non-trivial combinations of constraints, iterative approaches like Markovian random walks in the space of sequences of inputs in general *fail* in extracting scenarios according to a given distribution (e.g., uniformly), and can be *very inefficient* to produce at all scenarios that are both legal (with respect to SUV assumptions) and of interest (with respect to the additional constraints). For example, in our case studies, up to 91% of the scenarios generated using such iterative approaches would need to be neglected. In this article, we show how, given a set of constraints on the input scenarios succinctly defined by multiple *finite memory monitors*, a data structure (*scenario generator*) can be synthesised, from which *any-horizon scenarios* satisfying the input constraints can be *efficiently* extracted by (possibly uniform) random sampling or (randomised) enumeration. Our approach enables *seamless support to virtually all simulation-based approaches to CPS verification*, ranging from simple random testing to statistical model checking and formal (i.e., exhaustive) verification, when a suitable bound on the horizon or an iterative horizon enlargement strategy is defined, as in the spirit of bounded model checking.

Index Terms—Simulation-based verification, cyber-physical systems, scenario generation

1 Introduction

Cyber-Physical Systems (CPSs) are typically *non-terminating* systems encompassing *software* which senses and controls (in an endless loop) one or more *physical plants* (e.g., motors, electrical circuits, etc.) Such systems are ubiquitous in a wide spectrum of application domains, e.g., automotive, space, avionics, smart grids, systems biology, healthcare, just to mention a few.

The intrinsic complexity of industry-relevant CPSs makes their verification and certification very challenging. To this end, *model-based* approaches are widely exploited to enable *system verification* starting from the early design phases, well before an actual implementation is built (see, e.g., [8]). Indeed, model-based approaches work on a *model* describing the CPS behaviour, rather than on an actual implementation.

While for, e.g., digital circuits, model-based verification is usually carried out using *model-checking* techniques (see, e.g., [19]), the case of CPSs is peculiar: In fact, although, from a formal point of view, they can be modelled as *hybrid systems*, model checkers for hybrid systems can only handle verification of moderately sized CPSs. Thus, *numerical simulation* is currently the main workhorse for the model-based verification of large CPSs, and is widely supported by available design tools (see, e.g., Simulink, VisSim, Dymola, ESA Satellite Simulation Infrastructure SIMULUS). Such tools take as input a model of the behaviour of the CPS (for example, via systems of algebraic-differential equations plus algorithmic snippets for, e.g., handling of events) along with an *operational scenario* (defining the actual system inputs and the conditions on the environment in which the CPS is operating), and provide as output the associated time course of the quantities of interest (*system trajectory*) up to a user-requested time-horizon (possibly dynamically revised during the verification process).

1.1 Motivation

The case of CPS models only available as *black boxes* (e.g., *simulators*, our focus here) is typical for many industry-relevant systems and has far-reaching implications. In

• *The authors are with the Computer Science Department, Sapienza University of Rome, 00185 Roma, Italy. E-mail: {tmancini, melatti, tronci}@di.uniroma1.it.*

Manuscript received 12 February 2021; revised 14 July 2021; accepted 30 August 2021. Date of publication 2 September 2021; date of current version 17 October 2022.
This work was partially supported by the Italian Ministry of University & Research under Grant Dipartimenti di eccellenza 2018–2022 of the Department of Computer Science, Sapienza University of Rome; INdAM GNCS Project 2020; Sapienza U. Projects RG11816436BD4F21, RG11916B892E54DB, RP11916B8665242F; Lazio POR FESR Projects E84G20000150006, F83G17000830007.
(Corresponding author: Toni Mancini.)
Recommended for acceptance by C. Wang.
Digital Object Identifier no. 10.1109/TSE.2021.3109842

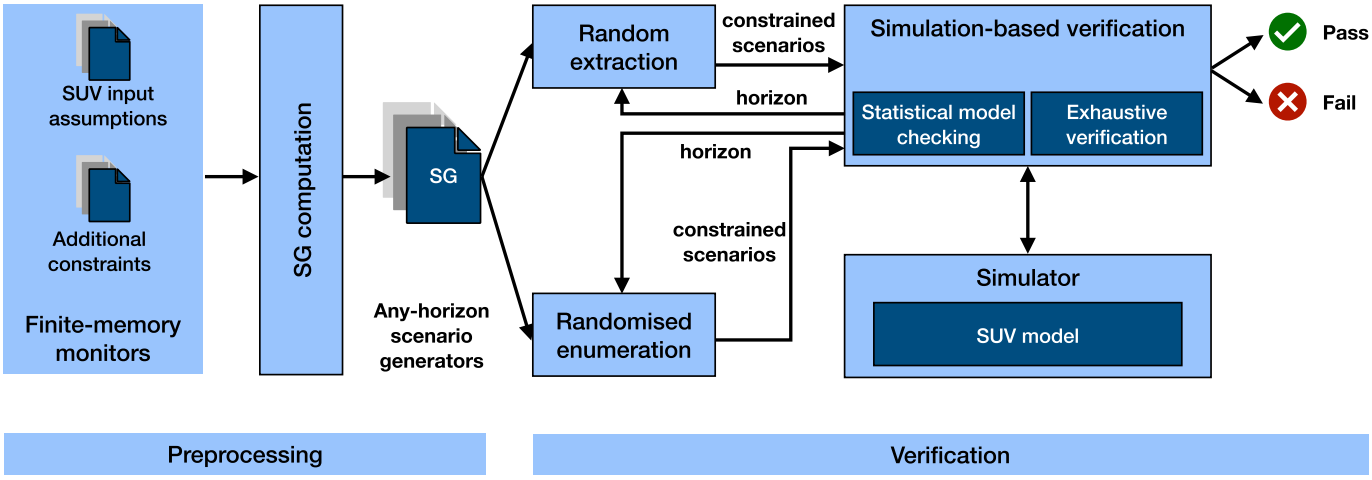

Fig. 1. High-level overview of our approach.

particular, (bounded-horizon) *numerical simulation* of the System Under Verification (SUV) model becomes the *only viable means* to obtain the system trajectory under any given input scenario.

In this setting, carrying out a verification process of the SUV upon an *explicit, precise, and implementation-independent definition* of the set of its (non-terminating) *legal input scenarios* (which stem from requirements deriving from the SUV *assumptions* and constraints about its operational *environment*) is crucial for the system *certification* (for, e.g., *regulatory* purposes). Also, the possibility to effectively *sample* (possibly *uniformly at random*) from this set or (*randomly*) *enumerate* it (when finite [43], [47]), is important to derive *statistical guarantees* about its correctness (e.g., via Statistical Model Checking, SMC), and to spot *hard-to-find* errors (i.e., errors in the system attained only under a *small* number of input scenarios).

Furthermore, *effective means to dynamically focus*, among such legal scenarios, on those that satisfy certain *additional constraints*, and to efficiently enumerate or sample at random within these subsets, enables *prioritisation* of the (often prohibitively long) verification activity, and allows one to focus on scenarios that drive the system towards specific portions of the space of behaviours, e.g., to *explicitly exercise selected requirements* or to avoid *vacuity* in their satisfaction (see, e.g., [12], [39], [69]).

Unfortunately, it is well known (see, e.g., [17] and citations thereof) that, in case of non-trivial combinations of constraints, iterative approaches like Markovian random walks in the space of sequences of inputs in general *fail* in extracting scenarios according to a given distribution (e.g., uniformly). Also, such approaches can be *very inefficient* to produce at all scenarios that are both legal (with respect to SUV assumptions and requirements on its environment) and of interest (with respect to the additional constraints). This is because the intricacies of and the inter-dependencies among requirements and additional constraints easily make such iterative approaches reach *deadlocks*. As an example, in our case studies up to 91% of random walks in the space of sequences of *legal* inputs would *fail* to produce scenarios satisfying all the provided constraints. This would translate into a *major source of inefficiency* for the verification (e.g., SMC) algorithm, which would be forced to *trash out* such (possibly huge) ratio of generated random walks.

1.2 Contributions

In this article we rely on Finite State Machines (FSMs) as a succinct, flexible, and practical means to compositionally define (as *monitors*) requirements about legal scenarios on which the SUV shall be exercised, as well as additional constraints aiming at restricting the focus of the verification process (e.g., for prioritisation needs, or to exercise some requirements avoiding vacuity in their satisfaction). Furthermore, we show how, by exploiting and combining together results from supervisory control theory and combinatorics, we can synthesise a data structure (Scenario Generator) from which any-horizon scenarios entailed by a monitor, or a combination thereof, can be efficiently extracted by (possibly uniform) random sampling or (randomised) enumeration.

Our approach enables *seamless support to virtually all simulation-based approaches to CPS verification*, ranging from simple random testing to SMC and formal (i.e., exhaustive) verification, when a suitable bound on the horizon or an iterative/dynamic horizon enlargement strategy is defined, as in the spirit of bounded model checking (see, e.g., [47]). Fig. 1 gives a high-level overview of our approach.

1.3 Paper Outline

This article is organised as follows. Section 2 introduces relevant background and states our formal setting. Section 3 presents our approach to the (possibly uniform) random sampling and (possibly randomised) enumeration of any-horizon constrained scenarios. Section 4 describes our three case studies and our experimental results. Section 5 is devoted to related work and Section 6 draws conclusions.

2 Background and Formal Setting

In this section we introduce the relevant background notions and give the formal setting on which we build our approach.

Throughout the paper, we denote with $\mathbb{R}$, $\mathbb{R}_{0+}$ and $\mathbb{R}_{+}$ the sets of, respectively, all real, non-negative real, strictly positive real numbers, and with $\mathbb{N}_{+}$ and $\mathbb{N}$ the sets of,

respectively, strictly positive and non-negative integer numbers. We also denote with $\mathrm{Bool} = \{\mathrm{false}, \mathrm{true}\}$ the set of Boolean values. Furthermore, given set A, we denote with A^* the set of infinite sequences of elements of A.

2.1 Systems and System Requirements

In this article, we focus on verifying *non-terminating* systems, such as CPSs, available as black boxes via simulators. These systems are typically defined as dynamical systems [63].

Contracts have been widely advocated to formalise the *requirements* of a system in an implementation-independent way, in terms of *assumptions* of the inputs and *guarantees* on the outputs (see, e.g., [13]). Verifying whether SUV $\mathcal{H}$ is correct with respect to its contract means to check that, whenever $\mathcal{H}$ is fed with inputs satisfying the *constraints* dictated by the contract assumptions, then it produces outputs compliant to the contract guarantees.

We assume that inputs to our SUV $\mathcal{H}$ will be given as unbounded *input time functions* in the form of time courses of *assignments* to n variables $\mathbb{V} = \{v_1, v_2, \ldots, v_n\}$ (the *input variables* of $\mathcal{H}$), where each v_i takes values within domain $\mathbb{U}_{v_i}$. Thus, each constraint defining the set of input scenarios to consider (e.g., as dictated by the contract assumptions or by any additional constraint we might want to enforce) will be defined over (a subset of) these variables. Given any $V = \{v_{j_1}, v_{j_2}, \ldots, v_{j_k}\} \subseteq \mathbb{V}$, we denote by $\mathbb{U}_V = \mathbb{U}_{v_{j_1}} \times \cdots \times \mathbb{U}_{v_{j_k}}$ the space of assignments of variables in V to values of their respective domains, and given an assignment $u \in \mathbb{U}_V$ (on some set of variables $V \subseteq \mathbb{V}$) and a subset V' of V, we denote by $u_{|V'} \in \mathbb{U}_{V'}$ the *projection* of u onto V', i.e., the assignment which coincides with u for all variables in V' and is undefined elsewhere.

2.2 Monitors

Since our SUVs of interest are non-terminating, both assumptions and guarantees of their contracts are *time unbounded*. Hence, we need practical means to *finitely* and *conveniently* define them. A general and flexible means to define sets of unbounded time functions is via the use of *monitors*, i.e., (possibly black-box) systems that scan an input time function and reject it as soon as they conclude it violates some requirement.

Given our focus on verification tasks where numerical simulation is the only means to get the trajectory of the SUV when fed with an input scenario, we assume that the SUV input space $\mathbb{U}_{\mathbb{V}}$ is *finite* (and, without loss of generality, *ordered*) and contract assumptions can be defined by a monitor which uses only *finite memory*.

Our assumption is in line with an engineering (rather than purely mathematical) point of view, where man-made CPSs need to satisfy the properties under verification with some degree of *robustness* with respect to the actual input time functions (see, e.g., [1], [23] and references thereof).

A direct consequence of our setting is that it allows us to seamlessly support (see Section 3) the *full continuum* from contract random testing to statistical model checking up to formal (i.e., exhaustive) simulation-based (robust) verification of black-box systems.

Our approach naturally applies to systems whose input values denote *events* such as faults (which can typically be finitely enumerated under a given order). However, thanks to the SUV robustness hypothesis, inputs assuming *continuous values* can be tackled by means of a suitable *discretisation* of their (ordered) domains, whilst truly *continuous-time inputs* (e.g., additive noise signals) can be managed as long as they can be cast into (or suitably approximated by) *finitely parametrisable* functions, in which case the input space actually defines such a (discrete or discretised) parameter space. Practical examples of such finite parameterisations of the SUV input space (or projections thereof) are those defining limited, quantised Taylor expansions of continuous-time finite inputs, or those defining quantised values for the first coefficients (those carrying out the most information) of the Fourier series of a finite-bandwidth noise, see, e.g., [1], [49]. Clearly, the most suitable approach to finitely define the SUV input space (perhaps a safe approximation thereof, consistent with the verification needs) depends on the case at hand, and should be carefully designed by the verification engineer. Our case studies in Section 4 contain uses of several of such features, and show that our setting can be easily met in practice.

Definition 1 introduces the notion of (finite-memory) monitor on top of the standard notion of FSM.

Definition 1 (Monitor). *A monitor $\mathcal{M}$ is defined in terms of a FSM, namely a tuple (V, X, x_0, f) where:*

- *$V \subseteq \mathbb{V}$ is the finite set of* input variables*, which defines the monitor* input space *$\mathbb{U}_V$;*
- *X is the finite set of the monitor* states*;*
- *$x_0 \in X$ is the monitor* initial state*;*
- *$f : X \times \mathbb{U}_V \to X$, the monitor* transition function*, is a (possibly partial) function defining the* legal transitions *of $\mathcal{M}$.*

We denote by $Traces(\mathcal{M}) \subseteq \mathbb{U}_V^*$ the set of traces *of $\mathcal{M}$, that is the set of* infinite *sequences $(u_0, u_1, u_2, \ldots)$ of inputs (assignments to the input variables) associated to the infinite computation paths of $\mathcal{M}$ that start from x_0.*

Given $h \in \mathbb{N}_+$, we denote by $Traces(\mathcal{M})_{|h}$ the set of prefixes *of length h of the traces of $\mathcal{M}$.*

Note that a monitor transition function f can (and usually will) be partial. This allows us to easily encode within f constraints on the infinite sequences of inputs entailed by the monitor (i.e., the set of monitor traces).

Monitor traces can be easily interpreted as piecewise-constant discrete- or continuous-time functions. Indeed, given a time unit $\tau \in \mathbb{T} - \{0\}$ (with $\mathbb{T}$, the time-set, being $\mathbb{N}$ or $\mathbb{R}_{0+}$ for discrete- and continuous-time contracts and systems, respectively), a trace $(u_0, u_1, u_2, \ldots)$ of a monitor $\mathcal{M}$ is interpreted as time function $\mathbf{u}$ defined as $\mathbf{u}(t) = u_{\lfloor \frac{t}{\tau} \rfloor}$ for all $t \in \mathbb{T}$. This allows us, in particular, to regard traces of a monitor defining the assumptions of a contract for a system model $\mathcal{H}$ as (suitable approximations of) actual input time functions with which to feed $\mathcal{H}$ during simulation-based verification.

Monitors can be conveniently defined using a wide variety of specification languages, e.g., temporal logics (see, e.g., [31]), data-flow languages (see, e.g., [30]), but also conventional programming and simulation languages (e.g., the

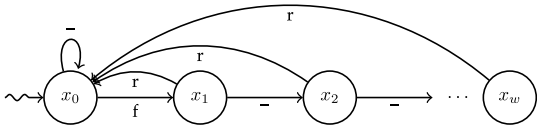

Fig. 2. An explicit view of the FSM for monitor $\mathcal{A}_{j_i}$ in Example 1 ($i \in [1,q]$). Edge labels represent assignments to the single input variable v_{j_i} ('–': no-op, 'f': fault, 'r': repair).

same language of the SUV simulator itself). Indeed, it will be enough, for our approach to work, that monitors are given as *black boxes*, in that we only need to repeatedly invoke their transition function.

Definition 2 shows that monitors can be *conjoined* by simply combining their (possibly black-box) transition functions. This will allow us to define *systems of constraints* for, e.g., the whole set of assumptions of a contract starting from monitors for subspaces thereof, e.g., for subsets of the requirements, plus monitors enforcing additional constraints for the scenarios of interest.

Definition 2 (Conjoint monitor). *Let $\mathcal{M}_a = (V_a, X_a, x_{a,0}, f_a)$ and $\mathcal{M}_b = (V_b, X_b, x_{b,0}, f_b)$ be two monitors over (possibly overlapping) sets of variables.*

The conjoint monitor $\mathcal{M} = \mathcal{M}_a \bowtie \mathcal{M}_b$ *between $\mathcal{M}_a$ and $\mathcal{M}_b$ is $(V_a \cup V_b, X_a \times X_b, (x_{a,0}, x_{b,0}), f_a \bowtie f_b)$, where for all $x_a \in X_a$, $x_b \in X_b$, $u \in \mathbb{U}_{V_a \cup V_b}$:*

$$(f_a \bowtie f_b)((x_a, x_b), u) = (f_a(x_a, u_{|V_a}), f_b(x_b, u_{|V_b})),$$

if both $f_a(x_a, u_{|V_a})$ and $f_b(x_b, u_{|V_b})$ are defined and is undefined otherwise.

The set of traces of $\mathcal{M}$, $Traces(\mathcal{M})$, is the set of sequences $(u_0, u_1, u_2, \ldots) \in \mathbb{U}^*_{V_a \cup V_b}$ such that $(u_{0|V_a}, u_{1|V_a}, u_{2|V_a}, \ldots) \in Traces(\mathcal{M}_a)$ and $(u_{0|V_b}, u_{1|V_b}, u_{2|V_b}, \ldots) \in Traces(\mathcal{M}_b)$.

Example 1 shows a contract whose assumptions are defined via a finite-memory monitor, obtained by composing monitors defining the assumptions from single sub-systems (assumption subspaces).

Example 1. Consider the following contract to be implemented by the digital control system of an autopilot for a space vehicle (this is inspired from our Apollo Lunar Module Digital Autopilot, ALMA, case study in Section 4.1.3). The system receives, as an input stream from the mission-control system, requests to change the attitude of (i.e., to rotate, along one or more axes) the vehicle. Such requests are defined as assignments over input variables V_θ. At each input, the new attitude is computed by the system itself on the basis of the current attitude and the new request. Such a stream of inputs is guaranteed to satisfy some additional constraints (such that, e.g., the requests can be actually obeyed by the vehicle, given its technical capabilities). All this yields monitor $\mathcal{A}_\theta$ defining legal input functions over input variables V_θ. When all rotation commands and attitude values are multiple of a minimum angle and occur at time points multiple of a given time unit, monitor $\mathcal{A}_\theta$ is finite-memory.

The autopilot control system receives feedback about the current attitude of the vehicle from a given set of sensors, and commands a set of q on-board actuators (directional jets). Jets may be subject to temporary unavailabilities (aka *faults*) which are always recovered within a given number w of time units (t.u.) By denoting with v_{j_i} the (single) input variable devoted at representing fault and repair events on the i-th jet ($i \in [1,q]$), the above yields finite-memory monitor $\mathcal{A}_{j_i}$ over the input variable v_{j_i} (see Fig. 2).

The additional requirement that the control system must be resilient to the concurrent unavailability of at most $k \leq q$ actuators yields finite-memory monitor $\mathcal{A}_{\leq k}$ over input variables $\left\{v_{j_1}, \ldots, v_{j_q}\right\}$.

The overall set of input variables of the contract that the autopilot must implement is thus $V = V_\theta \cup \left\{v_{j_1}, \ldots, v_{j_q}\right\}$, and the overall monitor $\mathcal{A}$ for the contract assumptions is obtained by conjoining the above submonitors as shown in Definition 2. Additional constraints (defined via additional monitors) on the set of scenarios of interest can be enforced similarly.

3 Any-Horizon Uniform Random Sampling and Enumeration of Constrained Scenarios From Monitors

When performing simulation-based verification of a SUV model $\mathcal{H}$, we need to simulate $\mathcal{H}$ under input scenarios satisfying all the considered requirements, e.g., assumptions on the SUV inputs and any additional constraints we chose to enforce, in order to focus or prioritise the verification activity. In practice, as such scenarios are time unbounded, we need to consider suitable *finite prefixes* of them.

In this section we show how, given a monitor $\mathcal{M}$ over input variables V, for example obtained by conjoining the monitor defining contract assumptions and monitors defining additional constraints, we can use a combination of techniques inspired from supervisory control theory and combinatorics, in order to define two functions:

(1) $nb\_traces : \mathbb{N} \to \mathbb{N}$
(2) $trace : \mathbb{N} \times \mathbb{N} \to \mathbb{U}^*_V$

Given a horizon $h \in \mathbb{N}$, $nb\_traces(h)$ evaluates to the number of *distinct* prefixes of length h of traces in $Traces(\mathcal{M})$; whilst, given $h \in \mathbb{N}$ and $i \in [0, nb\_traces(h) - 1]$, $trace(i,h)$ evaluates to the i-th (in lexicographic order) such prefix. Building on these two functions, it will be straightforward to implement any trace *random sampling policy* (possibly *uniform*), just by choosing an horizon or a range thereof (possibly dynamically during the verification activity) and then sampling over the range of indices of traces of the chosen horizon(s). *Random enumeration* of traces of a given horizon can similarly be achieved by randomly sampling a permutation of the range of indices (see, e.g., [47]), and any horizon-enlargement verification policies (as in the spirit of bounded model checking [20]) can be seamlessly applied.

Functions nb_traces and $trace$ can be implemented as to be very efficient, as it will be shown in Section 3.2. Namely, after a preliminary $\mathcal{O}(|\mathbb{U}_V||X|^2)$ step, functions nb_traces and $trace$ can be implemented to run in (amortised) time $\mathcal{O}(1)$ and $\mathcal{O}(h \log |\mathbb{U}_V|)$, respectively.

3.1 From Monitors to Scenario Generators

When monitor $\mathcal{M}$ is obtained by conjoining multiple sub-monitors, *inter-dependency* constraints may arise on the legal sequences of input values/events that make some *finite* computation paths of $\mathcal{M}$ not extensible to the infinity, and thus not defining legal prefixes of $Traces(\mathcal{M})$. Technically, we say that $\mathcal{M}$ might be *blocking* and, hence, may lead to *deadlocks*.

Example 2. Consider again Example 1. From knowledge stemming from the root causes of the temporary unavailability of each actuator (e.g., the need to charge internal capacitors or batteries after intensive use), we know that, if the attitude change requests follow certain patterns, then certain actuators (those intensively used to obey those requests) are more likely to become temporarily unavailable in the near future.

We might then decide to focus the verification of the autopilot system by prioritising those legal scenarios that *actually show* such actuator unavailabilities. This yields *monitor* $\mathcal{F}$.

Thus, our verification activity will first focus on input functions belonging to the monitor $\mathcal{M} = \mathcal{A} \bowtie \mathcal{F}$. Monitor $\mathcal{M}$ resulting from such combination of requirements would be blocking. This is a consequence of the fact that infinite computation paths of $\mathcal{M}$ need to satisfy conflicting requirements: 'no more than k actuators are simultaneously unavailable' (from $\mathcal{A}$) and 'whenever attitude change requests follow certain patterns, some given actuators will become unavailable within a certain time'. Paths not extensible to the infinity (e.g., those envisioning attitude change request patterns yielding, at some point in time, unavailabilities of more than k actuators) represent scenarios that should *not be considered* in the current verification stage.

Thus, our first (and one-time preliminary) step is to *minimally restrict* monitor $\mathcal{M}$ into a new monitor $Gen(\mathcal{M})$ (Scenario Generator, SG) such that: (i) all finite paths of $Gen(\mathcal{M})$ can be extended to the infinity ($Gen(\mathcal{M})$ is *non-blocking*), and (ii) all the infinite paths of $\mathcal{M}$ are retained ($Gen(\mathcal{M})$ is *complete*).

The key concept we need for this purpose (and borrowed from supervisory control theory) is the set of *safe* states of $\mathcal{M}$ (Definition 3), i.e., those states that are actually traversed by infinite paths of $\mathcal{M}$.

Definition 3 (Safe states of a monitor). *Let $\mathcal{M} = (V, X, x_0, f)$ be a finite-state monitor. State $x \in X$ is* safe *for $\mathcal{M}$ if $\Phi_f(x)$ is true, where $\Phi_f : X \to \text{Bool}$ is the greatest fixed-point of the following equation:*

$$\forall x \in X \quad \Phi_f(x) \equiv \exists u, x' \quad x' = f(x, u) \wedge \Phi_f(x'). \tag{1}$$

The Scenario Generator (SG) stemming from monitor $\mathcal{M}$ (Definition 4) is obtained by restricting the transition function of $\mathcal{M}$ as to reach only safe states.

Definition 4 (Scenario Generator). *Let $\mathcal{M} = (V, X, x_0, f)$ be a monitor and Φ_f be the function defined in Definition 3 for $\mathcal{M}$, with $\Phi_f(x_0) = \text{true}$.*

The Scenario Generator (SG) for $\mathcal{M}$ is monitor $Gen(\mathcal{M}) = (V, X, x_0, f_{gen})$, where, for any state x and input u, $f_{gen}(x, u) = f(x, u)$ if $f(x, u)$ is defined and $\Phi_f(f(x, u)) = \text{true}$, and is undefined otherwise.

Note that Definition 4 requires that the initial state x_0 of $Gen(\mathcal{M})$ is a safe state for $\mathcal{M}$. Indeed, if not, then $\mathcal{M}$ entails no trace, and no SG for it exists.

A few observations are in order. The definition of Φ_f in Definition 3 is well-posed, since formula (1) is formally positive, hence, by the Knaster-Tarski Theorem [27], its greatest fixed-point exists. This implies that the SG of a monitor is unique, when it exists.

Function Φ_f for monitor $\mathcal{M} = (V, X, x_0, f)$ can be computed in time $\mathcal{O}(|\mathbb{U}_V||X|^2)$, by starting with $\Phi_f \equiv \text{true}$ and by iteratively setting $\Phi_f(x)$ to false for those states x violating (1), until the fixed-point is reached. Note that, to perform such a computation, it is enough to have $\mathcal{M}$ available as a black box, as we only need to repeatedly invoke its transition function f.

Finally, we point out that function Φ_f implicitly defines the (unique) most liberal *supervisory controller* for $\mathcal{M}$, i.e., function $K : X \times \mathbb{U}_V \to \text{Bool}$, such that $K(x, u) = (\exists x' \; x' = f(x, u) \wedge \Phi_f(x'))$ [71]. (Note that what are called *uncontrollable inputs* in [71] can be modelled in the starting monitor with intermediate states having only one outgoing transition.)

Proposition 1 shows that SGs satisfy our requirements (proofs of all results are available in the supplementary material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2021.3109842).

Proposition 1. *Let $\mathcal{M} = (V, X, x_0, f)$ be a monitor and $Gen(\mathcal{M}) = (V, X, x_0, f_{gen})$ be its associated SG. Then:*

(1) Each finite path of $Gen(\mathcal{M})$ is extensible to an infinite path (non-blocking);
(2) $Gen(\mathcal{M})$ and $\mathcal{M}$ have the same set of traces: $Traces(Gen(\mathcal{M})) = Traces(\mathcal{M})$ (completeness).

Being based on FSMs, SGs have several desirable properties. Remark 1 states some of them.

Remark 1. For every pair of monitors $\mathcal{M}_a$ and $\mathcal{M}_b$:

(1) $Gen(Gen(\mathcal{M}_a)) = Gen(\mathcal{M}_a)$;
(2) $Gen(\mathcal{M}_a \bowtie \mathcal{M}_b) = Gen(Gen(\mathcal{M}_a) \bowtie Gen(\mathcal{M}_b))$;
(3) If $\mathcal{M}_a$ and $\mathcal{M}_b$ share no input variables (i.e., they are *independent* monitors), then $Gen(\mathcal{M}_a \bowtie \mathcal{M}_b) = Gen(\mathcal{M}_a) \bowtie Gen(\mathcal{M}_b)$.

Although straightforward, properties listed in Remark 1 are very useful in practice. Namely, 2) enables the *incremental* computation of an SG defined by further constraining (e.g., by imposing additional requirements on) an already computed SG (e.g., an assumptions monitor). Property 3) holds when conjoining *independent* sub-monitors. This is a frequent situation occurring when defining the assumptions of a contract for a system obtained by composing monitors (on assumptions subspaces) on its components, as in Example 1. We will see in Remark 3 that in such cases, it is enough to compute the SGs of the different (typically much smaller) sub-monitors in order to extract traces of the (never computed) conjoint monitor.

3.2 Index-Based Trace Extraction

Definition 5 defines two helper functions that are at the basis of the definition of functions nb_traces and $trace$.

Definition 5 (Functions ext and ξ). *Let $Gen(\mathcal{M}) = (V, X, x_0, f_{gen})$ be the SG of $\mathcal{M}$. We define the following two functions:*

- $ext : X \times \mathbb{N} \to \mathbb{N}$, *where, for any $x \in X$ and $k \in \mathbb{N}$:*
  - $ext(x, k) = 1$ *if* $k = 0$;
  - $ext(x, k) = \sum_{\hat{u} \in \mathbb{U}_V} ext(f_{gen}(x, \hat{u}), k-1)$ *otherwise.*
- $\xi : X \times \mathbb{U}_V \times \mathbb{N} \to \mathbb{N}$, *where, for any $x \in X$ and $k \in \mathbb{N}$: $\xi(x, u, k) = \sum_{\hat{u} < u} ext(f_{gen}(x, \hat{u}), k)$.*

Remark 2 (Adapted from [70]). Let $Gen(\mathcal{M}) = (V, X, x_0, f_{gen})$ be a SG. For every $x \in X$ and $k \in \mathbb{N}$, function $ext(x, k)$ (Definition 5) evaluates to the number of all-distinct computation paths of length k of $Gen(\mathcal{M})$ starting from x.

Since $Gen(\mathcal{M})$ is a *deterministic* automaton, each computation path is uniquely associated to a distinct sequence of inputs. This yields our definition of function $nb\_traces : \mathbb{N} \to \mathbb{N}$ which, given a horizon h returns the number $ext(x_0, h)$ of distinct prefixes of length h of the traces of $Gen(\mathcal{M})$, i.e., the cardinality of $Traces(Gen(\mathcal{M}))_{|h}$. Conversely, function ξ, by offering a way to efficiently compute any trace prefix of length h from its unique index, is at the basis of the definition of function $trace : \mathbb{N} \times \mathbb{N} \to \mathbb{U}_V^*$.

Algorithm 1. Index-Based Trace Extraction From a SG

```
1   global
2     Gen(M) = (V, X, x_0, f_gen);
3     h_max ∈ ℕ ∪ {undef}, initially undef;
4     ext, a map of the form X × ℕ → ℕ, initially empty;
5     ξ, a map of the form X × 𝕌_V × ℕ → ℕ, init. empty;
      //Invariant: ext(x,h) & ξ(x,u,h) defined iff h ≤ h_max
6   function nb_traces(h)
      Input: h ∈ ℕ
7     if h_max = undef or h > h_max then
8       incrementally compute ext and ξ up to h;
9       h_max ← h;
10    return ext(x_0, h);
11  function trace(i, h)
      Input: i ∈ ℕ, h ∈ ℕ
      Ouput: (u_0, u_1, u_2, ..., u_{h-1}), i-th trace of len. h
12    if i ≥ nb_traces(h) then error index out of bounds;
13    x ← x_0; k ← h; m ← i;
14    for j from 0 to h − 1 do
15      u_j ← max{u | ξ(x, u, k − 1) ≤ m};
16      m ← m − ξ(x, u_j, k − 1);
17      x ← f_gen(x, u_j);
18      k ← k − 1;
19    return (u_0, u_1, u_2, ..., u_{h-1});
```

Algorithm 1 shows an implementation of functions nb_traces and $trace$ which relies on an incremental computation of maps implementing functions ext and ξ. Provided that constant-time lookup tables (i.e., *maps*) for functions ext and ξ are already available for all inputs up to horizon h, the time complexity of functions nb_traces and $trace$ is, respectively, $\mathcal{O}(1)$ and $\mathcal{O}(h \log |\mathbb{U}_V|)$. Note that, in the typical case where the verification activity asks trace prefixes of the same length h, computation of maps ext and ξ is a *one-time task*, and its cost (although in practice negligible with respect to simulation time, see Section 4) is *amortised* by the multiple (efficient) executions of function $trace$.

Proposition 2 (Correctness of Algorithm 1, adapted from [70]). *Functions in Algorithm 1 are such that:*

(1) For any given $h \in \mathbb{N}$, $nb\_traces(h)$ returns the cardinality of $Traces(Gen(\mathcal{M}))_{|h}$, i.e., the number of all-distinct prefixes of length h of the traces of $Gen(\mathcal{M})$.

(2) For any given $h \in \mathbb{N}$ and $i \in [0, nb\_traces(h) - 1]$, $trace(i, h)$ returns the i-th (in lexicographic order) element of $Traces(Gen(\mathcal{M}))_{|h}$.

Remark 3 clarifies that when a monitor $\mathcal{M}$ is defined as the conjoining of two sub-monitors whose input spaces share no input variables (*independent* sub-monitors), then functions nb_traces and $trace$ for $Gen(\mathcal{M})$ can be equivalently stated in terms of the respective functions for $Gen(\mathcal{M}_a)$ and $Gen(\mathcal{M}_b)$. This allows one to avoid the computation of $\mathcal{M}$ and of $Gen(\mathcal{M})$ altogether, and to perform the index-based extraction of traces of $\mathcal{M}$ by pairing together traces extracted from (the typically much smaller) $\mathcal{M}_a$ and $\mathcal{M}_b$ (by computing only $Gen(\mathcal{M}_a)$ and $Gen(\mathcal{M}_b)$, thanks to property (3) of Remark 1).

Remark 3. Let $\mathcal{M}_{ab}$ be a monitor defined as $\mathcal{M}_a \bowtie \mathcal{M}_b$, with $\mathcal{M}_a$ and $\mathcal{M}_b$ sharing no input variables.

Let nb_traces_γ and $trace_\gamma$ be the functions of Algorithm 1 for SG $Gen(\mathcal{M}_\gamma)$, with $\gamma \in \{a, b, ab\}$.

The following holds for all $h \in \mathbb{N}_+$:

(1) $nb\_traces_{ab}(h) = nb\_traces_a(h) * nb\_traces_b(h)$ (where '$*$' is integer multiplication);

(2) For all $i \in [0, nb\_traces_{ab}(h) - 1]$, $trace_{ab}(i, h) = trace_a(sel(i, a), h) \cdot trace_b(sel(i, b), h)$, where:
- $sel(i, a) = \left\lfloor \frac{i}{nb\_traces_b(h)} \right\rfloor$
- $sel(i, b) = i \bmod nb\_traces_b(h)$,

with $\bmod$ being the modulo operator, and '$\cdot$' the *pairing* of two traces: $(u_{a,0}, u_{a,1}, u_{a,2}, \ldots) \cdot (u_{b,0}, u_{b,1}, u_{b,2}, \ldots) = ((u_{a,0}, u_{b,0}), (u_{a,1}, u_{b,1}), (u_{a,2}, u_{b,2}), \ldots)$.

4 Experimental Results

In this section we define our case studies (Section 4.1) and present experimental results assessing the practical viability and scalability of our approach, focusing first on SG computation (Section 4.2) and then on index-based trace extraction from SGs (Section 4.3).

4.1 Case Studies

We experiment with 3 case studies consisting of contracts for the following 3 system models: Fuel Control System (FCS), Buck DC/DC Converter (BDC), Apollo Lunar Module Digital Autopilot (ALMA).

Sections 4.1.1, 4.1.2, and 4.1.3 give more details on these systems and their associated contracts.

4.1.1 Fuel Control System

The Fuel Control System (FCS) is a Simulink/Stateflow model (distributed with MathWorks Simulink) of a

TABLE 1
Fuel Control System: Filtering Criteria

constraint monitor	description
1	Each sensor will fail every 15–20 t.u.
2	Whenever a fault on the throttle sensor occurs, a fault on the speed sensor will occur within 9–11 t.u.
3	Whenever a fault on the throttle sensor occurs, a fault on the speed sensor will occur within 13–15 t.u.
4	Whenever a fault on the throttle sensor occurs, a fault on the speed sensor will occur within 18 or 19 t.u.
5	Whenever a fault on the EGO sensor occurs, a fault on the MAP sensor will occur within 16 or 17 t.u.
6	Whenever a fault on the EGO sensor occurs, a fault on the MAP sensor will occur within 20 or 21 t.u.

controller for a fault tolerant gasoline engine, which has also been used as a case study in [18], [37], [38], [42], [47], [48], [74].

The FCS has four sensors: Throttle angle, speed, EGO (measuring the residual oxygen present in the exhaust gas) and MAP (manifold absolute pressure). The goal of the control system is to maintain the air-fuel ratio (the ratio between the air mass flow rate pumped from the intake manifold and the fuel mass flow rate injected at the valves) close to the stoichiometric ratio of 14.6, which represents a good compromise between power, fuel economy, and emissions. From the measurements coming from its 4 sensors, the FCS estimates the mixture ratio and provides feedback to the closed-loop control, yielding an increase or a decrease of the fuel rate.

The FCS sensors are subject to temporary faults, and the whole control system is expected to tolerate single sensor faults. In particular, if a sensor fault is detected, the FCS changes its control law and operates the engine with a higher fuel rate to compensate. In case two or more sensors fail, the FCS shuts down the engine, as the air-fuel ratio cannot be controlled.

The assumptions monitor $\mathcal{A}_{\mathrm{FCS}}$ for the FCS we define for our experiments specifies that only one sensor can be faulty at any given time, and that each faulty sensor recovers within the following time bounds (in t.u.): 3–5 (throttle), 5–7 (speed), 10–15 (EGO), 13–17 (MAP). Accordingly, the *input space* of the SUV and its contract comprises 6 values: '–' (no-op), 'fault on s' (one value for each of the four sensors s) and 'repair' (which triggers repair of the single faulty sensor).

We define the additional monitors listed in Table 1 to further constrain the scenarios to focus on in various ways. In our experiments we will combine such constraints together.

4.1.2 Buck DC/DC Converter

The Buck DC/DC Converter (BDC) is a mixed-mode analog circuit converting the DC input voltage (denoted as V_i) to a desired DC output voltage (V_o). Buck converters are often used off-chip to scale down the typical laptop battery voltage (12–24 V) to the just few volts needed by, e.g., a laptop processor (the *load*) as well as on-chip to support dynamic voltage and frequency scaling in multicore processors (see, e.g., [54]). A BDC converter is *self-regulating*, i.e., it is able to maintain the desired output voltage V_o notwithstanding variations in the input voltage V_i or in the load R.

TABLE 2
Buck DC/DC Converter: Filtering Criteria

constraint monitor	description
1	V_i changes at least every 6 t.u.
2	V_i changes at least every 7 t.u.
3	R changes at least every 5 t.u.
4	R changes at least every 6 t.u.
5	V_i and R do not change simultaneously
6	Whenever V_i changes, R will change after 8 or 9 t.u.
7	Whenever V_i changes, R will change after 2 t.u.

A typical control system for the converter is based on software (implemented with a micro-controller) controlling a switch u, implemented with a MOSFET. Our case study involves the BDC software controller based on fuzzy logic originally introduced in [62].

The assumptions monitor $\mathcal{A}_{\mathrm{BDC}}$ for the BDC we define for our experiments specifies that the input voltage V_i and the load R may vary during time up to at most $\pm 30\%$ of their nominal values, in steps of $\pm 5\%$ and $\pm 10\%$ of their initial values. Also, the assumptions monitor specifies that values for V_i and R are stable for at least 6 and 5 t.u., respectively. Finally, to have a proper set-up, V_i and R are assumed stable to their nominal values for the first 2 t.u. The above assumptions monitor is easily definable as $\mathcal{A}_i \bowtie \mathcal{A}_R$, where sub-monitors $\mathcal{A}_i$ and $\mathcal{A}_R$ focus on the input voltage and the load separately.

We define the additional monitors listed in Table 2 to further constrain the scenarios to focus on in various ways. In our experiments we will combine such constraints together.

4.1.3 Apollo Lunar Module Digital Autopilot

The Apollo Lunar Module Digital Autopilot (ALMA) is a Simulink/Stateflow model (distributed with MathWorks Simulink) defining the logic that implements the phase-plane control algorithm of the autopilot of the lunar module used in the Apollo 11 mission.

The Module is equipped with sensors (for yaw, roll and pitch) and actuators (16 reaction jets to rotate the Module along the three axes). The controller takes as input a request to change the Module attitude (i.e., to perform a rotation along the three axes) as well as its current orientation as read from the sensors, and computes which reaction jets to fire to obey the request.

The assumptions monitor $\mathcal{A}_{\mathrm{ALMA}}$ for the ALMA we define for our experiments can be defined as $\mathcal{A}_{\mathrm{s}} \bowtie \mathcal{A}_{\mathrm{rj}}$, where sub-monitor $\mathcal{A}_{\mathrm{s}}$ deals with sensors noise signals, and $\mathcal{A}_{\mathrm{rj}}$ with rotation commands and consequent actuation of reaction jets. Namely:

- $\mathcal{A}_{\mathrm{s}}$ (assumptions monitor on sensors) specifies that sensors can be subject to additive noise in the form of one out of 6 predefined continuous-time signals

TABLE 3
Apollo Lunar Module Digital Autopilot: Filtering Criteria

constraint monitor	description
1	Only jets number 15 and 16 may be temporarily unavailable
2	Whenever a jet is actuated for 2 consecutive t.u., it will certainly become unavailable within 3 or 4 t.u.
3	At most 1 jet is unavailable at any time
4	Rotation requests regard at most 1 axis each
5	Rotation requests regard at most 2 axes each
6	Noise signal changes for at most 1 sensor at any time
7	Noise signal for each sensor remains stable for at least 5 and at most 10 t.u. and changes by ± 1 position in the given order

(which, to ease application of forthcoming additional contraints, are assumed to be sorted by amplitude).

- $\mathcal{A}_{\text{rj}}$ (assumptions monitor on rotation commands and reaction jet actuations) specifies that attitude change requests do not ask the autopilot to immediately undo the rotation requested along any axis in the preceding t.u., and that the reaction jets may be temporary unavailable (unavailabilities are always recovered within 2–3 t.u.) Although $\mathcal{A}_{\text{rj}}$ can be further decomposed, for simplicity we consider it as a whole.

We define the additional monitors listed in Table 3 to further constrain the scenarios to focus on in various ways. In our experiments we will combine such constraints together.

TABLE 4
Computation Times of Scenario Generators (SGs)

SUV	SG nb.	$\mathcal{M}$ assumptions monitor	$\mathcal{M}$ constraint monitors	$\mathcal{M}$ size of input space	$Gen(\mathcal{M})$ time [s]
FCS	1	$\mathcal{A}_{\text{FCS}}$	–	6	0.100
	2	$\mathcal{A}_{\text{FCS}}$	1	6	7.990
	3	$\mathcal{A}_{\text{FCS}}$	1, 3	6	4.920
	4	$\mathcal{A}_{\text{FCS}}$	1, 2	6	4.610
	5	$\mathcal{A}_{\text{FCS}}$	1, 4	6	6.340
	6	$\mathcal{A}_{\text{FCS}}$	1, 4, 5	6	5.920
	7	$\mathcal{A}_{\text{FCS}}$	1, 4, 6	6	6.550
BDC	1	$\mathcal{A}_i$	–	5	0.190
	2	$\mathcal{A}_R$	–	5	0.170
	3	$\mathcal{A}_i \bowtie \mathcal{A}_R$	–	25	0.360
	4	$\mathcal{A}_i$	1	5	0.120
	5	$\mathcal{A}_i$	2	5	0.170
	6	$\mathcal{A}_R$	3	5	0.110
	7	$\mathcal{A}_R$	4	5	0.160
	8	$\mathcal{A}_i \bowtie \mathcal{A}_R$	5	25	37.340
	9	$\mathcal{A}_i \bowtie \mathcal{A}_R$	2, 4, 5	25	29.680
	10	$\mathcal{A}_i \bowtie \mathcal{A}_R$	2, 4, 5, 6	25	1.940
	11	$\mathcal{A}_i \bowtie \mathcal{A}_R$	1, 3, 5, 7	25	2.160
ALMA	1	$\mathcal{A}_{\text{rj}}$	–	1769472	0.440
	2	$\mathcal{A}_{\text{rj}}$	1	108	0.440
	3	$\mathcal{A}_{\text{rj}}$	1, 2	108	448.880
	4	$\mathcal{A}_{\text{rj}}$	1, 2, 3	108	247.270
	5	$\mathcal{A}_{\text{rj}}$	1, 2, 3, 4	108	55.190
	6	$\mathcal{A}_{\text{rj}}$	1, 2, 3, 5	108	188.300
	7	$\mathcal{A}_{\text{s}}$	–	27	2.940
	8	$\mathcal{A}_{\text{s}}$	6	27	1.330
	9	$\mathcal{A}_{\text{s}}$	6, 7	27	782.200
	10	$\mathcal{A}_{\text{ALMA}}$	1, 2, 3, 4, 6, 7	2916	837.390

4.2 Computation of Scenario Generators

In this section we show experimental results about generation of SGs associated to our case studies. Our Python/C hybrid implementation allows users to define monitors in different convenient ways: Either using concise object-oriented Python code (one of the best known and simplest to use general-purpose programming languages), or via standard Functional Mock-up Unit (FMU) objects. The latter are opaque binary objects defining dynamical systems according to the Functional Mock-up Interface (FMI) open standard for model exchange. As such, FMUs can be *automatically* generated from 100+ different simulation platforms, including Modelica simulators (also open source implementations via, e.g., [60]), Mathworks Stateflow/Simulink, and SBML (via, e.g., the tool in [41]).

Indeed, Our SG computing software expects a monitor object (either a Python object or an FMU, with other languages/formats that could be similarly supported) implementing a few API functions (mainly, a function returning the input values admissible in the current monitor state and one performing a transition from the current state given an admissible input value). Such functions can also be easily provided by the user as to define a conjoined monitor of other monitors. Our implementation computes SGs by performing a Depth-First Search (DFS) on the input monitor treated as a black box. This means it needs to access only the monitor initial state and input space, and to repeatedly invoke the monitor transition function (and get the resulting states, even if as opaque objects). Saving and restoring monitor states during search is implemented either within our software (for Python-defined monitors) or by exploiting the FMI API (for FMU-defined monitors, for which we used the implementation in [60]).

In the following experiments, we defined our monitors in Python. All computations were run on single cores of Intel (R) i7-4930K computers @ 3.40 GHz with 64 GB RAM.

Table 4 shows, for each SUV and each monitor $\mathcal{M}$ (defining contract assumptions satisfying the given filter conditions), the number of inputs of $\mathcal{M}$ (column "size of input space") as well as the time (in seconds) needed to compute $Gen(\mathcal{M})$ (column "time").

The table shows that computation of the SGs is very efficient. This is also because the decomposition properties of monitors can be often exploited (see Remarks 1 and 3). For example, the ALMA SG number 1 has been computed as a tuple of 19 sub-SGs, $(Gen_{r_1}, \ldots, Gen_{r_3}, Gen_{j_1}, \ldots, Gen_{j_{16}})$: $Gen_{r_1}, \ldots, Gen_{r_3}$ are three identical SGs associated to the assumption subspace of the rotation commands along each axis, and $Gen_{j_1}, \ldots, Gen_{j_{16}}$ are 16 identical SGs, each one associated to the assumption subspace of a single reaction jet. No combination among such 19 (independent) SGs needs to be actually computed in order to extract the associated traces (Remark 3), hence the computation time of whole SG is the overall time to compute one SG of each kind.

Clearly, when sub-monitors are defined which span multiple assumption subspaces, the SG computation may be more expensive. For example, to compute ALMA SGs number 2–6, we need to actually conjoin the 16 above single-jet SGs; and, to compute SGs number 8–9 we need to conjoin 3 identical SGs, each one defining the assumption subspace of the noise signal for each sensor, before conjoining the monitor defining constraint 6. On the other hand, the ALMA SG number 10 is never computed as a whole, but is defined as the *pair* of SGs number 5 and number 9 (hence, again its computation time is the sum of the computation times of two sub-SGs).

Overall, Table 4 shows that, even when we need to conjoin multiple sub-SGs because of the presence of constraint monitors spanning several assumption subspaces, *the overall computation times are negligible* when compared with the time needed to perform any kind of simulation-based verification of the SUV. As an example, the time to compute the most expensive SG in Table 4 (ALMA, number 10) equals the time to simulate just a few input traces of the Simulink ALMA SUV model (e.g., less than 50 traces for 200 t.u. each, since simulating each of them takes around 20 seconds).

4.3 Index-Based Trace Extraction From Scenario Generators

The decomposed representation of an SG as a tuple of sub-SGs is also exploited when extracting traces, by relying on the equivalences of Remark 3 when applicable. Fig. 3 shows the efficiency and scalability of our monitor-based approach to scenario generation for simulation-based verification of contracts for our CPSs. Namely, for each SG $Gen(\mathcal{M})$ reported in Table 4 and for different values for the time horizon h, the following statistics are plotted.

Number of Traces. This is the overall number of traces of length h entailed by $Gen(\mathcal{M})$, as returned by function nb_traces of Algorithm 1. Unsurprisingly, especially for SGs defined as tuples of many sub-SGs, such numbers can easy jump to tremendously high values.

Trace Extraction Time. This is the average time (in seconds) for the extraction of a single trace from its unique index, i.e., the average execution time of function $trace$ of Algorithm 1. Average trace extraction times have been computed by extracting 1000 same-horizon traces *uniformly at random*, and *include* the (amortised) time to compute maps ext and ξ (whose share essentially represents $\sim 100\%$ of the overall time). In case of SGs defined as tuples of sub-SGs, the amortised time to extract a single trace of horizon h is the sum of the times to extract a single trace of horizon h from each sub-SG. In all cases, the average trace extraction times are *negligible* (typically a tiny fraction of a second).

Selectivity of Constraint Monitors. This is the ratio between the number of traces of length h entailed by $Gen(\mathcal{M})$ and the number of traces (of the same horizon) entailed by the SG computed by *ignoring* any constraint monitor. This measures the selectivity of the (conjoined) applied constraint monitors, i.e., their power to narrow down the set of traces of interest for the current verification stage. The *extremely small ratios* show the crucial importance of constraint monitors to restrict the set of scenarios of interest to those satisfying additional requirements, and to obtain more reasonable numbers of traces on which to temporarily focus the verification activity. By defining a *priority list* of such additional requirements, the verification activity itself can be guided in a well-controlled way towards the scenarios of interest.

SG Selectivity (Comparison Against Baseline). This statistic directly measures the benefits of computing the SG from the starting monitor $\mathcal{M}$ (or collection of monitors) *before* extracting traces. Namely, SG selectivity is the ratio between the number of traces of length h entailed by $Gen(\mathcal{M})$ and the number of traces (of the same horizon) entailed by the (possibly blocking) monitor $\mathcal{M}$. Such ratio can be *very small* (as small as 9% in our experiments), especially when combinations of constraints are involved. Indeed, by skipping the construction of the SG, the verification process (e.g., a statistical model checker sampling Markovian random walks directly on $\mathcal{M}$, which can be thought of as a baseline) could possibly need, on our examples, to trash out as many as 91% of the sampled traces (for the FCS case study), once discovering that they do not represent bounded-horizon prefixes of legal infinite traces. Clearly, such ratio can be made arbitrarily small by using more intricate constraints.

5 Related Work

Several approaches, both explicit (see, e.g., [25], [70]) and symbolic (see, e.g., [2], [17]) have been proposed to sample uniformly at random finite paths from a finite-state combinatorial structure (automaton, combinatorial circuit, or SAT instance). Differently from such methods, since we focus on *finite prefixes of infinite paths*, we first need to restrict the input monitor (a FSM or a collection thereof) to its non-blocking portion, via the concept of Scenario Generator, by essentially computing the most liberal supervisory controller of it.

Automatic generation of admissible operational scenarios to support simulation-based verification of CPSs has been investigated in [42], [44], [45], [46], [48] as for fully general Simulink models, and in [67] as for ESA SIMSAT models. We note however that the above approaches need to generate *upfront* all scenarios of a given length satisfying the assumptions of the SUV contract in order to be able to select a subset thereof at random. When the number of the admissible scenarios is very large (our case) this can be impossible. Also, in case of blocking combinations of assumptions and filtering conditions, each generated scenario should be checked after being generated to avoid to waste time to simulate the SUV model on not-truly-admissible scenarios. As shown in Section 4, this may cause major loss of efficiency of the verification process. The approach proposed in this article solves this problem, by delivering a scenario generator that can efficiently extract scenarios satisfying the SUV contract assumptions (plus, possibly, additional constraints) of *any* length (possibly dynamically revised during verification) in *any given* order (e.g., uniformly at random, or as a randomised enumeration). This, in turn, enables an

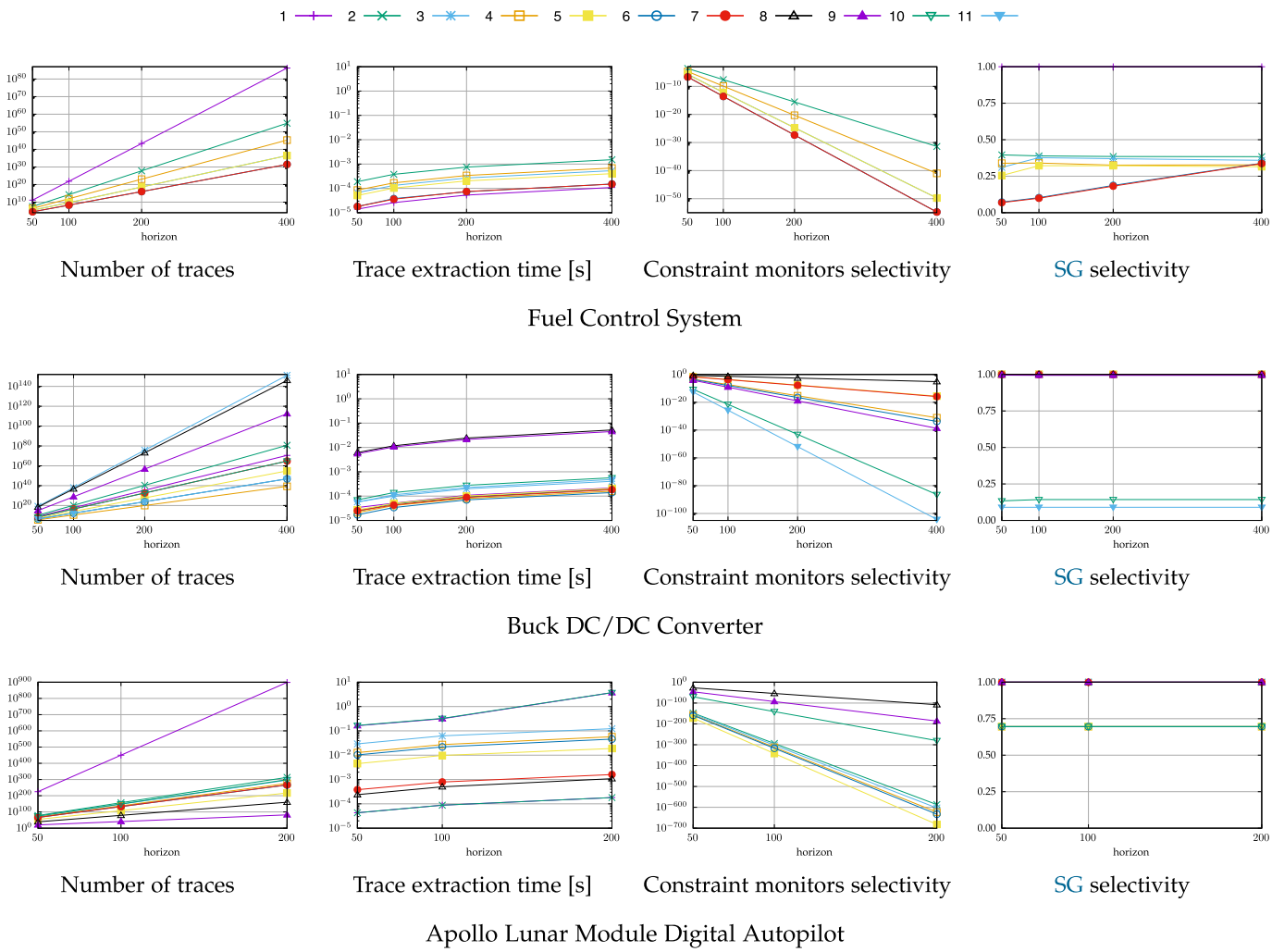

Fig. 3. Number of entailed traces, amortised trace extraction time (over 1000 traces), constraint monitor as well as SG selectivity (showing savings with respect to a Markovian random walk generator in the input monitors, our baseline) for different horizons (one curve per SG of Table 4).

embarrassingly parallel approach to SUV contract verification.

System-Level Formal Verification (SLFV) of CPSs via simulation-based bounded model checking has been studied in many contexts. Here are a few examples. Simulation-based reachability analysis for large linear continuous-time dynamical systems has been investigated in [9]. A simulation based data-driven approach to verification of hybrid control systems described by a combination of a black-box simulator for trajectories and a white-box transition graph specifying mode switches has been investigated in [24]. Formal verification of discrete time Simulink models (e.g., Stateflow or models restricted to discrete time operators) with small domain variables has been investigated in, e.g., [15], [55], [64], [68]. We note, however, that none of the above approaches supports simulation-based bounded model checking of arbitrary simulation models, starting from contracts defined via monitors.

Simulation-based approaches to SMC have been also widely investigated, e.g., [11], [14], [16], [26], [28], [29], [35], [36], [72], [73]. Simulink models for CPSs have been studied in [21], [74], mixed-analog circuits have been analysed in [18]. Smart grid control policies have been considered in [32], [50], [51], [52], biological models have been studied in [53], [56], [61], [66]. Our scenario generators are independent of the SMC technique employed and can be seamlessly applied to any of them. Also, thanks to the possibility to sample the set of scenarios *without* replacement, our algorithm (differently than Monte Carlo–based SMC) is able to answer a contract verification problem with *certainty* in those cases where a not-too-large number of bounded-horizon scenarios is enough to consider, and enough time is allocated to simulate all of them (see, e.g., [43], [47]).

Parallel approaches to SMC have been investigated, e.g., in [5] in the context of probabilistic properties. Our approach enables *embarrassingly parallel* verification. This is because, once a SG has been generated and a verification horizon has been selected, the range of indices of admissible traces can be split upfront into subintervals, and independent verification processes can be run in parallel on each of them.

Simulation-based *falsification* of CPS properties has been extensively investigated for Simulink models. Examples are in [1], [3], [6], [33], [59]. In particular, [1] falsifies a metric temporal logic property through Monte Carlo optimisation guided by a robustness metric defined by the property. Although [1] does not provide a quantification of the level of assurance achieved when no counterexample is found (as instead SMC does), we note that the search strategy in [1] can

falsify properties that would take too long to falsify using uniform sampling. Developing methods that can provide the benefits of both approaches is, to the best of our knowledge, an open problem and an important direction for future work.

The supervisory control problem has been introduced in [57], [58] in a language-theoretic setting. The work in [10] (respectively, [65]) presents a symbolic (OBDD-based) algorithm for the synthesis of (optimal) supervisory controllers for finite state systems. Since then, supervisory control theory has been used in many settings, e.g., analysis of database transaction execution [40], concurrent programs synthesis [34], confidentiality-enforcing in security [22], model-based system engineering [7], synthesis of control software [4], [54]. An up to date overview is in [71]. In this article, we use supervisory control theory through an *explicit* (since we face with possibly black-box monitors) implementation of the symbolic algorithm in [10].

6 Conclusion

In this article we focused on the (possibly uniform) random sampling and (randomised) enumeration of *constrained* input scenarios of *any horizon* for the simulation-based verification of non-terminating Cyber-Physical Systems (CPSs).

By relying on finite state machines as a succinct, flexible, and practical means to define *constraints* to be satisfied by the System Under Verification (SUV) input scenarios (where such constraints stem from *requirements* on the SUV inputs, e.g., assumptions and a definition of its *operational environment*, as well as from *additional conditions* aiming at dynamically *restricting* the focus of the verification process, e.g., to exercise some requirements or to counteract *vacuity* in their satisfaction), we showed how to exploit and combine together results from supervisory control theory and combinatorics in order to synthesise a data structure (Scenario Generator, SG) from which *any-horizon* scenarios can be efficiently *sampled* (possibly *uniformly* at random) or (randomly) *enumerated*.

Our methods can be seamlessly exploited in *virtually all simulation-based approaches to CPS verification*, ranging from simple random testing to statistical model checking and formal (i.e., exhaustive) verification (e.g., bounded model checking).

Acknowledgments

Authors are grateful to: The anonymous reviewers for their comments; Michele Laurenti, who contributed to initial investigations and developed a first prototype of our FMU-based SG computing software as part of his B.Sc. thesis; Vadim Alimguzhin, for discussions and technical assistance on our implementation.

References

[1] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivančić, and A. Gupta, "Probabilistic temporal logic falsification of cyber-physical systems," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 2s, pp. 95:1–95:30, 2013.

[2] D. Achlioptas, Z. S. Hammoudeh, and P. Theodoropoulos, "Fast sampling of perfectly uniform satisfying assignments, "in *Proc. 21st Int. Conf. Theory Appl. Satisfiability Testing*, 2018, pp. 135–147.

[3] A. Adimoolam, T. Dang, A. Donzé, J. Kapinski, and X. Jin, "Cassification and coverage-based falsification for embedded control systems," in *Proc. 29th Int. Conf. Comput. Aided Verification*, 2017, pp. 483–503.

[4] V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, and E. Tronci, "Linearizing discrete-time hybrid systems," *IEEE Trans. Autom. Control*, vol. 62, no. 10, pp. 5357–5364, Oct. 2017.

[5] M. AlTurki and J. Meseguer, "PVeStA: A parallel statistical model checking and quantitative analysis tool," in *Proc. 4th Int. Conf. Algebra Coalgebra Comput. Sci.*, 2011, pp. 386–392.

[6] Y. S. R. Annpureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan, "S-TaLiRo: A tool for temporal logic falsification for hybrid systems," in *Proc. 17th Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2011, pp. 254–257.

[7] J. C. M. Baeten, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Integration of supervisory control synthesis in model-based systems engineering," in, *Complex Systems*. Berlin, Germany: Springer, 2016, pp. 39–58.

[8] C. Baier and J. -P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. Cambridge, MA, USA: MIT Press, 2008.

[9] S. Bak and P. S. Duggirala, "Simulation-equivalent reachability of large linear systems with inputs," in *Proc. 29th Int. Conf. Comput. Aided Verification*, 2017, pp. 401–420.

[10] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin, "Supervisory control of a rapid thermal multiprocessor," *IEEE Trans. Autom. Control*, vol. 38, no. 7, pp. 1040–1059, Jul. 1993.

[11] A. Basu, S. Bensalem, M. Bozga, B. Caillaud, B. Delahaye, and A. Legay, "Statistical abstraction and model-checking of large heterogeneous systems," in *Proc. Int. Conf. Formal Techn. Distrib. Syst.*, 2010, pp. 32–46.

[12] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh, "Efficient detection of vacuity in temporal model checking," *Formal Methods Syst. Des.*, vol. 18, no. 2, pp. 141–163, 2001.

[13] A. Benveniste *et al.*, "Contracts for system design," *Found. Trends Electron. Des. Autom.*, vol. 12, no. 2/3, pp. 124–400, 2018.

[14] J. Bogdoll, L. M. F. Fioriti, A. Hartmanns, and H. Hermanns, "Partial order methods for statistical model checking and simulation," in *Proc. Int. Conf. Formal Techn. Distrib. Syst.*, 2011, pp. 59–74.

[15] P. Boström and J. Wiik, "Contract-based verification of discrete-time multi-rate Simulink models," *Softw. Syst. Model.*, vol. 15, no. 4, pp. 1141–1161, 2016.

[16] B. Boyer, K. Corre, A. Legay, and S. Sedwards, "PLASMA-lab: A flexible, distributable statistical model checking library," in *Proc. 10th Int. Conf. Quantitative Eval. Syst.*, 2013, pp. 160–164.

[17] S. Chakraborty, A. A. Shrotri, and M. Y. Vardi, "On uniformly sampling traces of a transition system," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2020, pp. 1–9.

[18] E. M. Clarke, A. Donzé, and A. Legay, "On simulation-based probabilistic model checking of mixed-analog circuits," *Formal Methods Syst. Des.*, vol. 36, no. 2, pp. 97–113, 2010.

[19] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.

[20] E. M. Clarke, T. A. Henzinger, and H. Veith, *Handbook of Model Checking*. Berlin, Germany: Springer, 2016.

[21] E. M. Clarke and P. Zuliani, "Statistical model checking for cyber-physical systems," in *Proc. 9th Int. Symp. Autom. Technol. Verification Anal.*, 2011, pp. 1–12.

[22] J. Dubreil, P. Darondeau, and H. Marchand, "Supervisory control for opacity," *IEEE Trans. Autom. Control*, vol. 55, no. 5, pp. 1089–1100, May 2010.

[23] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theor. Comput. Sci.*, vol. 410, no. 42, pp. 4262–4291, 2009.

[24] C. Fan, B. Qi, S. Mitra, and M. Viswanathan, "DryVR: Data-driven verification and compositional reasoning for automotive systems," in *Proc. 29th Int. Conf. Comput. Aided Verif.*, 2017, pp. 441–461.

[25] M. C. Gaudel, "Counting for random testing," in *Proc. 23rd IFIP Int. Conf. Testing Softw. Syst.* 2011, pp. 1–8.

[26] T. Gonschorek, B. Rabeler, F. Ortmeier, and D. Schomburg, "On improving rare event simulation for probabilistic safety analysis," in *Proc. 15th ACM-IEEE Int. Conf. Formal Methods Models Syst. Des.*, 2017, pp. 15–24.

[27] E. Grädel *et al.*, *Finite Model Theory and its Applications*. Berlin, Germany: Springer, 2007.

[28] R. Grosu and S. A. Smolka, "Quantitative model checking," in *Proc. 1st Int. Symp. Leveraging Appl. Formal Method*, 2004, pp. 165–174.

[29] R. Grosu and S. A. Smolka, "Monte Carlo model checking," in *Proc. 11th Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2005, pp. 271–286.

[30] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous data flow programming language LUSTRE," *Proc. IEEE*, vol. 79, no. 9, pp. 1305–1320, Sep. 1991.
[31] K. Havelund and G. Roşu, "Synthesizing monitors for safety properties," in *Proc. 8th Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2002, pp. 342–356.
[32] B. Hayes, I. Melatti, T. Mancini, M. Prodanovic, and E. Tronci, "Residential demand management using individualised demand aware price policies," *IEEE Trans. Smart Grid*, vol. 8, no. 3, pp. 1284–1294, May 2017.
[33] B. Hoxha, A. Dokhanchi, and G. Fainekos, "Mining parametric temporal logic properties in model based design for cyber-physical systems," *Int. J. Softw. Tools Technol. Transfer*, vol. 20, no. 1, pp. 79–93, 2018.
[34] M. V. Iordache and P. J. Antsaklis, "Concurrent program synthesis based on supervisory control," in *Proc. IEEE Amer. Control Conf.*, 2010, pp. 3378–3383.
[35] C. Jegourel, A. Legay, and S. Sedwards, "A platform for high performance statistical model checking–plasma," in *Proc. 18th Int. Conf. Tools Algorithms Construction Anal. Syst.*, 2012, pp. 498–503.
[36] C. Jegourel, A. Legay, and S. Sedwards, "Importance splitting for statistical model checking rare properties," in *Proc. 25th Int. Conf. Comput. Aided Verification*, 2013, pp. 576–591.
[37] Y. J. Kim, O. Choi, M. Kim, J. Baik, and T. -H. Kim, "Validating software reliability early through statistical model checking," *IEEE Softw.*, vol. 30, no. 3, pp. 35–41, May/Jun. 2013.
[38] Y.J. Kim and M. Kim, "Hybrid statistical model checking technique for reliable safety critical systems," in *Proc. 23rd IEEE Int. Symp. Softw. Rel. Eng.*, 2012, pp. 51–60.
[39] O. Kupferman and M.Y. Vardi, "Vacuity detection in temporal model checking," *Int. J. Softw. Tools Technol. Transfer*, vol. 4, no. 2, pp. 224–233, 2003.
[40] S. Lafortune, "Modeling and analysis of transaction execution in database systems," *IEEE Trans. Autom. Control*, vol. 33, no. 5, pp. 439–447, May 1988.
[41] F. Maggioli, T. Mancini, and E. Tronci, "SBML2Modelica: Integrating biochemical models within open-standard simulation ecosystems," *Bioinformatics*, vol. 36, no. 7, pp. 2165–2172, 2020.
[42] T. Mancini, F. Mari, A. Massini, I. Melatti, F. Merli, and E. Tronci, "System level formal verification via model checking driven simulation, in *Proc. 25th Int. Conf. Comput. Aided Verification*, 2013, pp. 296–312.
[43] T. Mancini, F. Mari, A. Massini, I. Melatti, I. Salvo, and E. Tronci, "On minimising the maximum expected verification time," *Inf. Process. Lett.*, vol. 122, pp. 8–16, 2017.
[44] T. Mancini, F. Mari, A. Massini, I. Melatti, and E. Tronci, "Anytime system level verification via random exhaustive hardware in the loop simulation," in *Proc. 17th Euromicro Conf. Digit. Syst. Des.*, 2014, pp. 236–245.
[45] T. Mancini, F. Mari, A. Massini, I. Melatti, and E. Tronci, "System level formal verification via distributed multi-core hardware in the loop simulation," in *Proc. 22nd Euromicro Int. Conf. Parallel Distrib. Netw.-Based Process.*, 2014, pp. 734–742.
[46] T. Mancini, F. Mari, A. Massini, I. Melatti, and E. Tronci, "SyLVaaS: System level formal verification as a service," in *Proc. 23rd Euromicro Int. Conf. Parallel Distrib. Netw.-Based Process.*, 2015, pp. 476–483.
[47] T. Mancini, F. Mari, A. Massini, I. Melatti, and E. Tronci, "Anytime system level verification via parallel random exhaustive hardware in the loop simulation," *Microprocessors Microsyst.*, vol. 41, pp. 12–28, 2016.
[48] T. Mancini, F. Mari, A. Massini, I. Melatti, and E. Tronci, "SyLVaaS: System level formal verification as a service, *Fundamenta Informaticae*, vol. 149, no. 1/2, pp. 101–132, 2016.
[49] T. Mancini, F. Mari, A. Massini, I. Melatti, and E. Tronci, "On checking equivalence of simulation scripts," *J. Log. Algebr. Methods Program.*, vol. 120, 2021, Art. no. 100640.
[50] T. Mancini *et al.*, "Demand-aware price policy synthesis and verification services for smart grids," in *Proc. IEEE Int. Conf. Smart Grid Commun.*, 2014, pp. 794–799.
[51] T. Mancini *et al.*, "Parallel statistical model checking for safety verification in smart grids," in *Proc. IEEE Int. Conf. Smart Grid Commun.*, 2018, pp. 1–6.
[52] T. Mancini *et al.*, "User flexibility aware price policy synthesis for smart grids," in *Proc. IEEE 18th Euromicro Conf. Digit. Syst. Des.*, 2015, pp. 478–485.
[53] T. Mancini, E. Tronci, I. Salvo, F. Mari, A. Massini, and I. Melatti, "Computing biological model parameters by parallel statistical model checking," in *Proc. 3rd Int. Conf. Bioinf. Biomed. Eng.*, 2015, pp. 542–554.
[54] F. Mari, I. Melatti, I. Salvo, and E. Tronci, "Model based synthesis of control software from system level formal specifications," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 1, pp. 1–42, 2014.
[55] B. Meenakshi, A. Bhatnagar, and S. Roy, "Tool for translating Simulink models into input language of a model checker," in *Proc. 8th Int. Conf. Formal Eng. Methods.*, 2006, pp. 606–620.
[56] N. Miskov-Zivanov, P. Zuliani, E. M. Clarke, and J. R. Faeder, "Studies of biological networks with statistical model checking: Application to immune system cells," in *Proc. ACM Conf. Bioinf. Comput. Biol. Biomed. Inform.*, 2013, pp. 728–729.
[57] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, 1987.
[58] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.
[59] S. Sankaranarayanan, S. A. Kumar, F. Cameron, B. W. Bequette, G. Fainekos, and D. M. Maahs, "Model-based falsification of an artificial pancreas control system," *ACM SIGBED Rev.*, vol. 14, no. 2, pp. 24–33, 2017.
[60] S. Sinisi, V. Alimguzhin, T. Mancini, and E. Tronci, "Reconciling interoperability with efficient verification and validation within open source simulation environments," *Simul. Modelling Pract. Theory*, vol. 109, 2021, Art. no. 102277.
[61] S. Sinisi, V. Alimguzhin, T. Mancini, E. Tronci, and B. Leeners, "Complete populations of virtual patients for in silico clinical trials," *Bioinformatics*, vol. 36, no. 22/23, pp. 5465–5472, 2020.
[62] W.-C. So, C. K. Tse, and Y.-S. Lee, "Development of a fuzzy logic controller for DC/DC converters: Design, computer simulation, and experimental evaluation," *IEEE Trans. Power Electron.*, vol. 11, no. 1, pp. 24–32, Jan. 1996.
[63] E. D. Sontag, *Mathematical Control Theory: Deterministic Finite Dimensional Systems*, 2nd ed. Berlin, Germany: Springer, 1998.
[64] S. Tripakis, C. Sofronis, P. Caspi, and A. Curic, "Translating discrete-time Simulink to Lustre," *ACM Trans. Embedded Comput. Syst.*, vol, 4, no. 4, pp. 779–818, 2005.
[65] E. Tronci, "Optimal finite state supervisory control," in *Proc. 35th IEEE Conf. Decis. Control.*, 1996, pp. 2237–2242.
[66] E. Tronci *et al.*, "Patient-specific models from inter-patient biological models and clinical records", in *Proc. IEEE 14th Int. Conf. Formal Methods Comput.-Aided Des.*, 2014, pp. 207–214.
[67] G. Verzino *et al.*, "Model checking driven simulation of sat procedures," in *Proc. 12th Int. Conf. Space Operations*, 2012, Art. no. 1275611.
[68] M. W. Whalen, D. D. Cofer, S. P. Miller, B. H. Krogh, and W. Storm, "Integration of formal analysis into a model-based software development process," in *Proc. 12th Int. Workshop Formal Methods Ind. Crit. Syst.*, 2007, pp 68–84.
[69] M. W. Whalen, A. Rajan, M. P. E. Heimdahl, and S. P. Miller, "Coverage metrics for requirements-based testing," in *Proc. Int. Symp. Softw. Testing Anal.*, 2006, pp. 25–36.
[70] H. S. Wilf, "A unified setting for sequencing, ranking, and selection algorithms for combinatorial objects," *Advances Math.*, vol. 24, no. 3, pp. 281–291, 1977.
[71] W. M. Wonham, *Supervisory Control of Discrete-Event Systems*. Berlin, Germany: Springer, 2015, pp. 1396–1404.
[72] H. L. S. Younes, M. Z. Kwiatkowska, G. Norman, and D. Parker, "Numerical vs. statistical probabilistic model checking," *Int. J. Softw. Tools Technol. Transfer*, vol. 8, no. 3, pp. 216–228, 2006.
[73] H. L. S. Younes and R. G. Simmons, "Probabilistic verification of discrete event systems using acceptance sampling," in *Proc. 14th Int. Conf. Comput. Aided Verification.*, 2002, pp. 223–235.
[74] P. Zuliani, A. Platzer, and E. M. Clarke, "Bayesian statistical model checking with application to Stateflow/SIMULINK verification," *Formal Methods Syst. Des.*, vol. 43, no. 2, pp. 338–367, 2013.