Information Processing Letters 122 (2017) 8–16

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

On minimising the maximum expected verification time

Toni Mancini, Federico Mari, Annalisa Massini*, Igor Melatti, Ivano Salvo, Enrico Tronci*

Computer Science Department, Sapienza University of Rome, Italy

A R T I C L E I N F O

Article history:
Received 18 February 2016
Received in revised form 2 November 2016
Accepted 1 February 2017
Available online 4 February 2017
Communicated by Krishnendu Chatterjee

Keywords:
Formal verification
Explicit model checking
System-level formal verification
Formal methods
Software engineering

A B S T R A C T

Cyber Physical Systems (CPSs) consist of hardware and software components. To verify that the *whole* (i.e., software + hardware) system meets the given specifications, *exhaustive* simulation-based approaches (Hardware In the Loop Simulation, HILS) can be effectively used by first generating *all* relevant simulation scenarios (i.e., sequences of *disturbances*) and then actually simulating all of them (*verification phase*). When considering the whole verification activity, we see that the above mentioned verification phase is repeated until no error is found. Accordingly, in order to minimise the time taken by the whole verification activity, in each verification phase we should, ideally, start by simulating scenarios witnessing errors (*counterexamples*). Of course, to know beforehand the set of such scenarios is not feasible. In this paper we show how to select scenarios so as to minimise the Worst Case Expected Verification Time.

1. Introduction

A CPS consists of hardware (e.g., engines, electrical circuits, etc.) and software components. Thus, in order to verify a CPS design, we need methods and tools that can model and effectively support analysis of hardware as well as software components.

From a formal point of view, CPS can be modelled as hybrid systems (see, e.g., [8,32,31] and citations thereof). Many *Model-Based Design* software tools offer support for modelling and simulation of CPSs. Well known examples are Simulink, VisSim, Open Modelica, JModelica, Dymola. All such tools take as input a (mathematical) model of the behaviour of the CPS along with a simulation scenario and provide as output the time evolution (*trace* or *simulation run*) of the system.

System Level Verification of CPSs aims at verifying that the *whole* (i.e., software + hardware) system meets the given specifications. *System Level Formal Verification (SLFV)* has the goal of *exhaustively* verifying that the above holds for *all* possible operational scenarios.

For digital circuits, formal verification is usually carried out using symbolic model checking techniques (see, e.g., [13,12]). Unfortunately, model checkers for hybrid systems cannot handle SLFV of real world CPSs because of state explosion. Thus, HILS is currently the main workhorse for system-level verification of CPSs, and is supported by model-based design tools.

In HILS, the *control software* (see, e.g., [30,4,5]) reads/sends values from/to mathematical models (*simulation*) of the physical systems (e.g., mechanical or electrical systems) it will be interacting with. Simulation can be very time consuming. Accordingly, in order to reduce system design time, there are tools providing modelling and simulation software along with FPGA-based hardware to support real-time simulation. Examples are Opal-RT and dSpace.

Finally, model-based design of CPSs often refers to the activity of synthesising control software from system re-

* Corresponding authors.
E-mail address: massini@di.uniroma1.it (A. Massini).

http://dx.doi.org/10.1016/j.ipl.2017.02.001
0020-0190/© 2017 Elsevier B.V. All rights reserved.

quirements (see, e.g., [7,6] and citations thereof). Here, instead, we assume that a model for the whole system (software + hardware) is given, and we are only interested in CPS SLFV.

1.1. Motivations

Simulation-based approaches to the analysis of hybrid systems have been proven very effective in application domains as diverse as CPSs (see, e.g., [24,28,17,11,41,1,42]), smart grids (see, e.g., [40,29,20]) and biological systems (see, e.g., [22,38]). The goal of all such approaches is to show that, notwithstanding the possible presence of *disturbances* (i.e., uncontrollable events such as faults, variations in system parameters, etc.) from the environment, the system meets its requirements. This is done by using HILS to show that for all *simulation scenarios* (i.e., time sequences of disturbances) in a given set, the system meets its requirements. HILS, in turn, is carried out using a simulator (e.g., Simulink, Open Modelica, JModelica, Dymola) able to model and simulate both hardware (e.g., mechanical or electrical systems) as well as software components. Simulation-based verification can be carried out using two approaches: *online* and *offline*. The *online* approach typically selects the next disturbance to be simulated using a *Monte Carlo* strategy. The verification activity then consists of a sequence of disturbance generation and simulation steps. The *offline* approach *first* generates the whole (ordered) set of scenarios to be simulated (*scenario generation phase*) and *then* simulates all of them (*verification phase*).

The verification activity simulates the SUV until either a scenario (*counterexample*) whose simulation returns *FAIL* is found, or all scenarios have been simulated and simulation returns *PASS*. If the verification activity returns *FAIL*, then the SUV design is revised, by exploiting the counterexample, and a new verification activity is performed. We note the following points.

First, with an *offline* approach to CPS verification, more than 99% of the overall verification time is spent in the verification phase (see, e.g., [24]). Namely, for CPSs, simulating a single scenario may take from several seconds to several minutes (see, e.g., [24,26,28]) depending on the complexity of the system model (since typically a system of ordinary differential equations has to be solved in order to simulate the SUV dynamics). For example, for the SUV considered in [24], we see that generating a simulation scenario takes on average 0.45 ms (thus generating 4 million simulation scenarios takes about 30 minutes), whereas the Simulink simulation of a single scenario takes on average about 16.8 seconds (and the sequential simulation of all scenarios would take more than 700 days!). This is in contrast with, e.g., digital hardware simulation, where the time needed to generate a scenario and to simulate it are comparable. Accordingly, within an *offline* framework, we can afford to increase (e.g., doubling) the time spent in the generation phase if that can decrease (even slightly) the expected time for the verification phase. Note however that the *offline* approach makes sense only for CPSs, whereas the *online* approach can always be used and is indeed the approach always used in digital hardware as well as in software verification.

Second, whenever an error is found (and the SUV revised accordingly), the verification activity needs to simulate again *all* scenarios, including those already been simulated in previous verification activities (since revising the SUV design may have introduced new errors).

Third, in the *offline* approach, the scenario generation phase is performed only once, at the beginning of the verification activity. This is possible because the scenario generation phase depends only on the environment the SUV will be interacting with, and not on the SUV model itself. Thus, revising the SUV design, after an error has been found, does not change the set of simulation scenarios to be considered in the verification phase.

From the above points, it follows that simulating scenarios preceding a counterexample is indeed a waste of time, since those scenarios will have to be simulated again anyway. In order to minimise such a waste of time, one would like to order the simulation scenarios in such a way that those witnessing errors (counterexamples) are simulated at the very beginning in each verification phase. Of course, to know beforehand the set of counterexamples is not feasible (it is indeed the purpose of the verification activity). Furthermore, while reordering of the set of scenarios to be simulated can be effectively done within an *offline* framework (and has been done indeed in [25,28]), this is not possible within an *online* framework where SUV simulation starts before the whole set of scenarios is known. Indeed, from [10] we see that no strategy to select the next disturbance in an *online* strategy can be optimal for all SUVs.

The above considerations motivate investigation on effective algorithms that can order the set of simulation scenarios so as to minimise the *worst case expected time* for the verification activity within an *offline* CPS verification approach.

Of course our techniques could be applied to simulation-based verification of any system (be it software or hardware) with a finite set of scenarios. However, from a practical point of view, it only makes sense when scenario generation takes much less than scenario simulation (see discussion above). Presently, to the best of our knowledge, this is only the case for CPSs and this is why we focus on them.

1.2. Main contributions

From the previous discussion we see that the *computation time* (defined as the number of scenarios to be simulated before hitting a *counterexample*, if any) of a verification phase (*off-line* approach) depends on the order in which scenarios are simulated and on *where* in such an order counterexamples are.

Accordingly, the generic verification phase (also simply called *verification* in the following) can be modelled as a two-player zero-sum game as follows. First, player 1 (*verifier*) chooses the (possibly probabilistic) ordering strategy in which scenarios will be simulated. Second, player 2 (*adversary*) chooses which scenarios will be counterexamples (that is, will witness an error). Finally, the verifier simulates the scenarios in the chosen order.

The goal for the verifier is to minimise the verification time, whilst the adversary aims at maximising it. In order to achieve such a goal, the adversary must place counterexamples in such a way that they will be the last scenarios to be simulated by the verifier. On the contrary, the verifier must order scenarios so that counterexamples are among the first scenarios to be simulated.

In our game-theoretical setting, this may be modelled by defining the payoff for the adversary as the verification time. This entails that the verifier aims at minimising verification time.

We take a non-deterministic model for the adversary and a probabilistic model for the verifier. Accordingly the Worst Case Expected Verification Time (WCEVT) is the maximum (among all possible choices of the adversary) of the expected number of scenarios to be simulated before hitting a counterexample. The objective for the verifier (respectively, adversary) is to minimise (respectively, maximise) the WCEVT. We note that the verification activity stops as soon as a counterexample is found. Thus, without loss of generality, we focus on the case in which the adversary places just one counterexample, since placing more than one would decrease the Expected Verification Time (EVT).

Our main contributions are as follows.

First, we show that the minimum WCEVT is $\frac{n+1}{2}$, where n is the number of simulation scenarios.

Second, we show that there is an infinite set (forming a bounded convex polytope) of optimal simulation strategies, i.e., strategies for which the verifier attains the $\frac{n+1}{2}$ optimal payoff.

Third, we show that ordering *simulation scenarios* in a uniformly random way yields an optimal simulation strategy.

Fourth, within an *online* (Monte Carlo–based) simulation setting, we show how to select probability distribution on *disturbances* so that the resulting simulation strategy is optimal.

1.3. Paper overview

This paper is organised as follows. Section 2 compares the paper contributions with the existing literature. Section 3 provides basic definitions on simulation scenarios. Section 4 characterises optimal *offline* simulation strategies. Section 5 characterises optimal *online* (Monte Carlo) simulation strategies. Section 6 gives proofs and, finally, Section 7 provides conclusions.

2. Related work

In this section we compare our contributions with related research work.

2.1. Monte Carlo–based simulation

Within an *online* setting (see Section 1.1), the performance of Monte Carlo–based exploration has been extensively studied. See, e.g., [10] for a survey evaluating many strategies to select the next action (*disturbance* in our setting) so as to minimise the expected time to attain the goal (*error state* in our context). The crucial difference of a Monte Carlo–based approach with respect to ours is that our strategy selects *scenarios* (i.e., sequences of disturbances) rather than single *disturbances*. The rationale behind [10] is that the set of scenarios is not known *beforehand*, but is rather discovered during the simulation process. This is indeed the typical case for digital hardware simulation and software simulation where the time to generate a simulation scenario is comparable to that of simulating it. However, in our setting (verification of CPSs) the time needed to generate a simulation scenario (e.g., from a finite state model of the environment as in [24]) is negligible (see Section 1.1) with respect to the time needed to simulate it. Accordingly, by exploiting full knowledge of the set of simulation scenarios, we can devise optimal strategies whereas, as shown in [10], this is not possible in the typical setting where such set is not known beforehand. In particular, [10] shows, with counterexamples, that selecting *disturbances* uniformly at random is *not always* an optimal strategy, whereas we show that selecting scenarios uniformly at random is *always* an optimal strategy (independently of the SUV).

The fact that test scenarios should be uniformly distributed is widely accepted to be valid (though not formally proved as it is in this paper) in many application domains, such as: generation of "well-distributed" random test programs for functional processors verification (see, e.g., [3]), generation of solutions to a constraint satisfaction problem uniformly at random (see, e.g., [14]). However, such uniform selection of scenarios cannot be achieved without full knowledge of the set of simulation scenarios, that is our setting. Furthermore, our formal approach goes beyond *conventional wisdom*, by showing that there is indeed an infinite convex set of optimal simulation strategies. This opens the door to methods that may be able to select an optimal test strategy (not necessarily a uniform one) even without full knowledge of the set of simulation scenarios.

2.2. Offline generation of simulation scenarios

Knowledge of the full set of scenarios to be simulated (*offline* approach, Section 1.1) can be exploited to speed up HILS based (formal) verification of CPSs. For example, [39,24,26,27] present *offline* strategies where simulation scenarios are ordered with respect to a *Depth-First Search (DFS)* on the finite state automaton describing the set of admissible disturbance sequences. We note however that none of the above papers aims at reducing the WCEVT. Random reordering of simulation scenarios is considered in [25,28] with the goal of supporting *graceful degradation* (of exhaustiveness), by estimating the *omission probability* (i.e., the probability that an error is present in a yet-to-be simulated scenario) during the verification activity. The present paper further explores the benefits of random ordering of simulation scenarios by showing how this can be used to minimise the WCEVT, an issue not addressed in any of the above papers.

2.3. CPS verification

Of course, *CPS verification* techniques have been widely investigated within the *online* verification setting (Section 1.1). For example [17,42] present *online* approaches using deterministic strategies to select the next disturbance to the SUV, whereas [11,41,1,42,16,33,2,23,18] present *online* approaches using probabilistic strategies (*Monte Carlo* simulation) to select the next disturbance to the SUV.

Monte Carlo Model Checking of finite state deterministic systems (see, e.g., [19,37,35]) is a formal (*online*) verification approach closely related to our setting (which however can also handle infinite state systems). Monte Carlo model checkers generate simulation scenarios using a Monte Carlo–based approach randomly selecting disturbances (rather than scenarios as in our *offline* setting).

Probabilistic model checking (see, e.g., [15,21] and citations thereof) consists of checking if a probabilistic property holds for a probabilistic system (modelled as a Markov Chain). We differ from such *online* (Section 1.1) approaches because here we consider deterministic systems and select scenarios rather than disturbances. Moreover, our approach is *black box*, that is we do not require a model for the system to be available, since we only require availability of a simulator. Thus the above mentioned approach cannot be used in our setting.

2.4. Summing up

From the above, we see that none of the above *online* approaches (i.e., Sections 2.1 and 2.3) addresses the problem of minimising the WCEVT, which is the focus of our *offline* approach. In fact, all such papers present (*online*) methods to search for requirement violations in the given system. On the contrary, our paper focuses on understanding what is the simulation strategy that minimises the (expected worst case) time to find an error. In this respect, our paper shows that an *offline* approach can provide *infinitely many* optimal simulation strategies (Section 4), whereas *online* selection strategies are, in general (see [10]), not optimal when considering the WCEVT (Section 5).

3. Background

In this section we give the definitions of simulation scenario and simulation campaign, as well as other preliminary notions.

In the following, unless otherwise stated, $\mathcal{D}$ denotes a non-empty finite set whose elements are called *uncontrollable inputs* or *disturbances*. Set $\mathcal{D}$ models the set of disturbances (e.g., faults, delays, etc.), including the null disturbance (i.e., nominal case), our SUV is supposed to withstand.

A simulation scenario is obtained by using disturbances in $\mathcal{D}$ according to the following definition.

Definition 1 *(Simulation Scenario).* A *simulation scenario* δ (or just *scenario*) is a finite sequence of elements of $\mathcal{D}$, that is $\delta = \langle d_1, d_2, \ldots, d_n \rangle$ with $d_i \in \mathcal{D}$ for all $i \in [1, n]$. Given a simulation scenario $\delta = \langle d_1, d_2, \ldots, d_n \rangle$, we write $\delta(i)$ for d_i, and we call n the *length* (or *time horizon*) of δ. Given two scenarios $\delta_1 = \langle d_{11}, d_{12}, \ldots, d_{1n} \rangle$ and $\delta_2 = \langle d_{21}, d_{22}, \ldots, d_{2m} \rangle$, their *concatenation* is defined by $\delta_1 \cdot \delta_2 = \delta_1\delta_2 = \langle d_{11}, d_{12}, \ldots, d_{1n}, d_{21}, d_{22}, \ldots, d_{2m} \rangle$. We denote $\mathcal{D}^+$ the set of all possible simulation scenarios on $\mathcal{D}$.

Example 1 *(Simulation Scenario).* Let us consider the Fuel Control System (FCS) model in the Simulink distribution, whose formal verification has been discussed in [24,26,28]. The model is equipped with four sensors: throttle angle, speed, Oxygen in Exhaust Gas (EGO) and Manifold Absolute Pressure (MAP). Let us assume that only sensors EGO and MAP can fail, giving rise to disturbances d_1 and d_2, respectively. Moreover, let us assume that the minimum time between faults is one second and all faults are transient and are repaired within one second. Hence disturbance d_1 models a fault on sensor EGO, followed by a repair within one second, and disturbance d_2 models a fault on sensor MAP, followed by a repair within one second too. We also consider the no-fault event, which we model with disturbance d_3. Then, the set of disturbances $\mathcal{D}$ is $\{d_1, d_2, d_3\}$. The following are examples of simulation scenarios: $\delta_1 = \langle d_1, d_3, d_2, d_3 \rangle$ (of length 4) and $\delta_2 = \langle d_2, d_3, d_2 \rangle$ (of length 3).

Remark 1 *(Scenario Simulation Time).* We assume that all scenarios take basically the same time to simulate regardless the disturbance sequence being simulated. This holds for many real world systems. For example, considering again the FCS in Example 1, we have that simulating (on an Intel(R) Xeon(R) @ 2.66 GHz Linux machine) a scenario of length 100 seconds takes on average (over about 75000 randomly selected scenarios) 16.80 seconds with a standard deviation of 2.99 seconds (i.e., 18% of the average time).

In the following, starting from the set of simulation scenarios, we define the notion of simulation campaign, which defines how each verification phase is actually performed. To this end, we first define the Admissible System Environment, which restricts the set of possible simulation scenarios to a more useful subset for our objective.

Definition 2 *(Admissible System Environment).* An Admissible System Environment (ASE) is a nonempty finite set of simulation scenarios $\mathcal{A} \subset \mathcal{D}^+$, such that no scenario in $\mathcal{A}$ is a prefix of another one. Formally, for each $\delta, \theta \in \mathcal{A}$, if $\delta \neq \theta$ then there exists no $\sigma \in \mathcal{D}^+$ such that: $\delta = \theta\sigma$.

Example 2 *(Admissible System Environment).* Let us consider the FCS model and the set of disturbances $\mathcal{D} = \{d_1, d_2, d_3\}$ of Example 1. Typically, one is interested in verifying the SUV when at most one fault can occur. Thus, if we only consider simulation scenarios of length 3, we obtain the simulation scenarios set $\mathcal{A} = \{\delta_1, \ldots, \delta_7\}$ consisting of $\delta_1 = \langle d_1, d_3, d_3 \rangle$, $\delta_2 = \langle d_2, d_3, d_3 \rangle$, $\delta_3 = \langle d_3, d_1, d_3 \rangle$, $\delta_4 = \langle d_3, d_2, d_3 \rangle$, $\delta_5 = \langle d_3, d_3, d_1 \rangle$, $\delta_6 = \langle d_3, d_3, d_2 \rangle$, $\delta_7 = \langle d_3, d_3, d_3 \rangle$. $\mathcal{A}$ is an ASE, in fact no scenario in $\mathcal{A}$ is the prefix of another one.

Given an ASE $\mathcal{A}$, we may now define the notion of simulation campaign.

Definition 3 *(Simulation Campaign).* A *simulation campaign* σ for an ASE $\mathcal{A} = \{\delta_1, \dots, \delta_n\}$ is a permutation $\langle \delta_{i_1}, \dots, \delta_{i_n} \rangle$ of the elements of $\mathcal{A}$. We denote with $\text{Sim}(\mathcal{A})$ the set of all $n!$ simulation campaigns for $\mathcal{A}$.

The j-th scenario in a simulation campaign σ, i.e., δ_{i_j}, is denoted with $\sigma(j)$. The *position* of a simulation scenario α in the simulation campaign σ is $\chi(\sigma, \alpha)$, that is, $\chi(\sigma, \alpha) = j$ if and only if $\delta_{i_j} = \sigma(j) = \alpha$.

Example 3 *(Simulation Campaign).* Let us consider the ASE $\mathcal{A} = \{\delta_1, \delta_2, \delta_3\}$. Then the set of simulation campaigns consists of $3! = 6$ elements. Namely, $\text{Sim}(\mathcal{A}) = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \sigma_6\}$, where

$$\sigma_1 = \langle \delta_1, \delta_2, \delta_3 \rangle, \;\; \sigma_2 = \langle \delta_1, \delta_3, \delta_2 \rangle,$$

$$\sigma_3 = \langle \delta_2, \delta_1, \delta_3 \rangle, \;\; \sigma_4 = \langle \delta_2, \delta_3, \delta_1 \rangle,$$

$$\sigma_5 = \langle \delta_3, \delta_1, \delta_2 \rangle \text{ and } \sigma_6 = \langle \delta_3, \delta_2, \delta_1 \rangle.$$

The position where the simulation scenario δ_3 occurs in the simulation campaign σ_4 is $\chi(\sigma_4, \delta_3) = 2$, whereas the position of δ_3 in the simulation campaign σ_1 is $\chi(\sigma_1, \delta_3) = 3$.

4. Minimising verification time

In this section we show the main results of our paper. To this aim, we first define the notions of error injection strategy and simulation strategy. This allows us to model the verification activity as a two-player zero-sum game. Namely, the error injection strategy is the (probabilistic) strategy of the adversary player, whilst the simulation strategy is the strategy for the verifier player.

Definition 4 *(Error Injection Strategy and Simulation Strategy).* An *error injection strategy* x for an ASE $\mathcal{A}$ is a real-valued function $x : \mathcal{A} \to [0, 1]$ such that $\sum_{\alpha \in \mathcal{A}} x(\alpha) = 1$.

A *pure* error injection strategy, denoted x^*_k for $k = 1, \dots, |\mathcal{A}|$, is a strategy defined as: $x^*_k(\delta_j) = 1$ if $k = j$ and 0 otherwise.

A *simulation strategy* y for an ASE $\mathcal{A}$ is a real-valued function $y : \text{Sim}(\mathcal{A}) \to [0, 1]$ such that $\sum_{\sigma \in \text{Sim}(\mathcal{A})} y(\sigma) = 1$.

A *pure* simulation strategy, denoted y^*_k for $k = 1, \dots, |\text{Sim}(\mathcal{A})| = 1, \dots, |\mathcal{A}|!$, is defined as: $y^*_k(\sigma_j) = 1$ if $k = j$ and 0 otherwise.

We denote with X the set of all error injection strategies, with Y the set of all simulation strategies, with $X^* \subseteq X$ the set of pure error injection strategies, and with $Y^* \subseteq Y$ the set of pure simulation strategies.

Example 4 *(Error Injection Strategy and Simulation Strategy).* Let us consider the ASE $\mathcal{A} = \{\delta_1, \delta_2, \delta_3\}$. Examples of error injection strategies are the functions x_1, $x_2 \in X$ defined as: $\mathbf{x_1}$) $x_1(\delta_i) = \frac{1}{3}$, $i \in [1, 3]$; $\mathbf{x_2}$) $x_2(\delta_1) = x_2(\delta_3) = 0$, $x_2(\delta_2) = 1$. Informally, strategy x_2 consists in deterministically choosing δ_2 as the failing scenario, whilst x_1 consists in picking the failing scenario at random among the three in $\mathcal{A}$. Note that x_2 is a pure strategy, whilst x_1 is not.

The set of simulation campaigns for $\mathcal{A}$ is $\text{Sim}(\mathcal{A}) = \{\sigma_1, \dots, \sigma_6\}$, where each σ_i is defined as in Example 3. Examples of simulation strategies are the functions y_1, $y_2 \in Y$ defined as: $\mathbf{y_1}$) $y_1(\sigma_i) = \frac{1}{6}$, $i \in [1, 6]$; $\mathbf{y_2}$) $y_2(\sigma_2) = y_2(\sigma_4) = \frac{1}{2}$, $y_2(\sigma_i) = 0$, $i = 1, 3, 5, 6$. Informally, strategy y_1 chooses at random any of the six available simulation campaigns whereas strategy y_2 chooses at random between the simulation campaigns σ_2 and σ_4. Note that none of the above two strategies is pure.

We may now define the expected and the worst case expected verification times defining the payoff for our game.

Definition 5 *(Expected Verification Time).* Given an error injection strategy x for an ASE $\mathcal{A}$ and a simulation strategy y for the set of simulation campaigns $\text{Sim}(\mathcal{A})$, the *Expected Verification Time (EVT)* for the verification flow is defined as the expected number of simulation scenarios to be simulated before hitting the one that witnesses the error:

$$\text{EVT}(x, y) = \sum_{\delta \in \mathcal{A}} \sum_{\sigma \in \text{Sim}(A)} x(\delta)\chi(\sigma, \delta)y(\sigma)$$

The *Worst Case Expected Verification Time (WCEVT)* is the maximum EVT after any adversary choice:

$$\text{WCEVT}(y) = \max_{x \in X} \text{EVT}(x, y).$$

Example 5 *(Expected Verification Time).* Let us consider the ASE $\mathcal{A}$, the error injection strategy x_1 and the simulation strategy y_2 of Example 4. Then $\text{EVT}(x_1, y_2) = 2$.

The following Lemma 1 points out a property of the EVT useful in proving our results. In Lemma 1 we consider the simulation strategy associating uniform probability to simulation campaigns and we denote it as $\hat{y}$, namely $\hat{y}(\sigma) = \frac{1}{n!}$ for all $\sigma \in \text{Sim}(\mathcal{A})$. We refer to $\hat{y}$ as the *uniform* simulation strategy.

Lemma 1. *Let $x \in X$ be an error injection strategy, $x^* \in X^*$ a pure error injection strategy and $\hat{y} \in Y$ be the uniform simulation strategy. Then* $\max_{x \in X} \text{EVT}(x, \hat{y}) = \max_{x^* \in X^*} \text{EVT}(x^*, \hat{y})$.

Proof. From game theory (see, e.g., [36]), we have that, for all $x \in X$, $x^* \in X^*$ and for $\hat{y} \in Y$,

$$\sum_{\delta \in \mathcal{A}} \sum_{\sigma \in \text{Sim}(\mathcal{A})} x(\delta)\chi(\sigma, \delta)\hat{y}(\sigma)$$
$$\leq \sum_{\delta \in \mathcal{A}} \sum_{\sigma \in \text{Sim}(\mathcal{A})} x^*(\delta)\chi(\sigma, \delta)\hat{y}(\sigma),$$

that is $\text{EVT}(x, \hat{y}) \leq \text{EVT}(x^*, \hat{y})$. This implies that $\max_{x \in X} \text{EVT}(x, \hat{y}) \leq \max_{x^* \in X^*} \text{EVT}(x^*, \hat{y})$. Since $X^* \subseteq X$, we also have that $\max_{x \in X} \text{EVT}(x, \hat{y}) \geq \max_{x^* \in X^*} \text{EVT}(x^*, \hat{y})$. □

Lemma 1 states that, when we consider the uniform simulation strategy $\hat{y}$, taking the maximum on set X is equivalent to taking the maximum on the subset of pure strategies X^*.

Note that the goal of the verifier player is to minimise WCEVT, i.e., to find $\bar{y} = \text{argmin}_{y \in Y}\, \text{WCEVT}(y)$. Thus, $\bar{y}$ is the strategy for which the WCEVT takes the minimum value, defined as:

$$\text{MiniMaxEVT} = \min_{y \in Y} \max_{x \in X} \sum_{\delta \in \mathcal{A}} \sum_{\sigma \in \text{Sim}(\mathcal{A})} x(\delta)\chi(\sigma,\delta)y(\sigma).$$

Our main result consists in providing a value for MiniMaxEVT, thus providing a lower bound for the verifier payoff, and the conditions for a simulation strategy to be optimal. This is stated in Theorem 1 (proof in Sections 6.1 to 6.3), which is inspired by the Minimax Theorem of Von Neumann [36].

Theorem 1 *(Minimum WCEVT). Let $\mathcal{A} = \{\delta_1, \ldots, \delta_n\}$ be an ASE. Then the following statements hold:*

1. *The value for the minimum WCEVT is:*

   $\text{MiniMaxEVT} = \min_{y \in Y} \max_{x \in X} \text{EVT}(x, y) = \frac{n+1}{2}$.
2. *A simulation strategy $y \in Y$ is optimal if and only if it satisfies the following constraints:*

   $\sum_{t=1}^{n} t \sum_{\chi(\sigma,\delta_i)=t} y(\sigma) = \frac{n+1}{2}$ *for $i \in [1, n]$.*
3. *A simulation strategy attaining the optimal payoff* MiniMaxEVT *is the uniform simulation strategy $\hat{y}(\sigma) = \frac{1}{n!}$.*

Remark 2 *(Set of Optimal Simulation Strategies).* There is an infinite number of optimal simulation strategies. Namely, any solution to the (feasibility) LP problem:

$$\begin{cases} \sum_{t=1}^{n} t \sum_{\chi(\sigma,\delta_i)=t} y(\sigma) = \frac{n+1}{2} \text{ for } i \in [1, n] \\ \sum_{\sigma \in \text{Sim}(\mathcal{A})} y(\sigma) = 1 \\ 0 \le y(\sigma) \le 1 \text{ for} | \sigma \in \text{Sim}(\mathcal{A}). \end{cases}$$

Note that the set of solutions to the above equations is a closed bounded convex polytope (see, e.g., [34]).

Example 6 *(Set of Optimal Simulation Strategies).* Let us consider the ASE $\mathcal{A} = \{\delta_1, \delta_2, \delta_3\}$ and the six simulation campaigns in $\text{Sim}(\mathcal{A})$ in Example 3. From Theorem 1 (statement 1) we have that $\text{MiniMaxEVT} = \frac{n+1}{2} = \frac{3+1}{2} = 2$. Furthermore, any simulation strategy y satisfying the constraints in item 2 of Theorem 1 will be optimal. It is easy to see that both simulation strategies y_1 and y_2 in Example 4 satisfy the constraints in statement 2 of Theorem 1 and are therefore optimal.

5. Monte Carlo–like simulation

Theorem 1 characterises optimal *off-line* (i.e., selecting *scenarios before* starting the simulation activity) simulation strategies. In this section we show how the results in Theorem 1 can be used to characterise optimal *on-line* (i.e., selecting *disturbances while* the simulation advances) simulation strategies. We will focus on Monte Carlo–based approaches (see Section 2.1) since typically on-line scenario generation rests on them.

First, we note that we may represent an ASE $\mathcal{A}$ with a directed (prefix) tree $T_{\mathcal{A}}$, which we call *disturbance tree*. The set of vertices of $T_{\mathcal{A}}$ is the set $\text{prefix}(T_{\mathcal{A}})$ of prefixes of the scenarios in $\mathcal{A}$, with the empty prefix being the *root* of $T_{\mathcal{A}}$. Each edge (u, v) of $T_{\mathcal{A}}$ is labelled with a disturbance $d \in \mathcal{D}$ such that $\langle u, d \rangle = v$. Accordingly, we will usually denote edges with pairs (u, d) where u is a node and d a disturbance.

A *probability matrix* for $T_{\mathcal{A}}$ is a map from edges of $T_{\mathcal{A}}$ to real values in $[0, 1]$ such that for each node u, $\sum_{\{d : \langle u,d \rangle \in \text{prefix}(T_{\mathcal{A}})\}} p(u, d) = 1$. Intuitively, $p(u, d)$ can be regarded as the probability of *injecting* disturbance d after having injected the sequence of disturbances u.

In the above setting we can easily compute the probability $P(u)$ of injecting a given sequence of disturbances as follows: $P(\langle\rangle) = 1$, $P(\langle u, d \rangle) = P(u)p(u, d)$. Hence, given an ASE $\mathcal{A}$, the corresponding *disturbance tree* $T_{\mathcal{A}}$, and the probability p of injecting disturbances, we obtain the simulation strategy y for $(\mathcal{A}, p)$. Namely, the simulation strategy y such that, for each $\langle \delta_1, \ldots \delta_n \rangle \in \text{Sim}(\mathcal{A})$, is obtained as $y_{\mathcal{A}}(\langle \delta_1, \ldots \delta_n \rangle) = \prod_{i=1}^{n} \frac{P(\delta_i)}{1 - \sum_{j=1}^{i-1} P(\delta_j)}$. Note that since the simulation scenarios in $\mathcal{A}$ are the leafs of $T_{\mathcal{A}}$, this definition is well posed. That is, $\sum_{\sigma \in \text{Sim}(\mathcal{A})} y_{\mathcal{A}}(\sigma) = 1$.

Remark 3. Theorem 1 provides optimality conditions for an $(\mathcal{A}, p)$ simulation strategy.

Note that the $(\mathcal{A}, p)$ simulation strategy y is *without* replacement, whereas Monte Carlo simulation strategies are usually implemented *with* replacement. This eases the implementation, since there is no need to store the already simulated scenarios, without sacrificing performance, since the probability of hitting the same scenario twice is negligible. Of course, from a theoretical point of view, a Monte Carlo sampling with replacement is always worse than a sampling approach without replacements. For example, our Monte Carlo–like $(\mathcal{A}, p)$ simulation strategy y is guaranteed to hit the error trace (if any) after at most $|\mathcal{A}|$ simulations, whereas a Monte Carlo simulation strategy with replacement never offers such a guarantee within a finite number of simulations (*Coupon Collector's Problem* [9]).

Proposition 1 provides a sufficient condition under which a probability matrix yields an optimal simulation strategy (proof is in Section 6.4).

Proposition 1 *(Optimal Monte Carlo Simulation Strategies). Let $\mathcal{A}$ be an ASE, $T_{\mathcal{A}}$ be the disturbance tree associated to $\mathcal{A}$ and p be a probability matrix for $T_{\mathcal{A}}$. If for all $\delta \in \mathcal{A}$, $P(\delta) = \frac{1}{|\mathcal{A}|}$, then the Monte Carlo simulation strategy y for $(\mathcal{A}, p)$ is optimal.*

Example 7 *(Optimal Monte Carlo Simulation Strategy).* Let us consider the ASE $\mathcal{A} = \{\delta_1, \delta_2, \delta_3\}$, where $\delta_1 = \langle d_1, d_1 \rangle$, $\delta_2 = \langle d_2, d_1 \rangle$, $\delta_3 = \langle d_2, d_2 \rangle$. The disturbance tree $T_{\mathcal{A}}$ associated to $\mathcal{A}$ is shown in Fig. 1. Let p be the probability matrix for $T_{\mathcal{A}}$ defined as follows: $p(\langle\rangle, d_1) = \frac{1}{3}$, $p(\langle\rangle, d_2) = \frac{2}{3}$, $p(\langle d_1 \rangle, d_1) = 1$, $p(\langle d_2 \rangle, d_1) = \frac{1}{2}$, $p(\langle d_2 \rangle, d_2) = \frac{1}{2}$. Then, $P(\delta_1) = P(\delta_2) = P(\delta_3) = \frac{1}{3}$. Thus, by Proposition 1, the corresponding simulation strategy y is optimal.

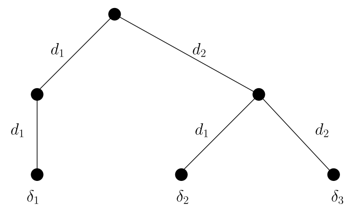

Fig. 1. Disturbance tree $T_\mathcal{A}$ for $\mathcal{A} = \{\delta_1, \delta_2, \delta_3\}$, where $\delta_1 = \langle d_1, d_1\rangle$, $\delta_2 = \langle d_2, d_1\rangle$, $\delta_3 = \langle d_2, d_2\rangle$ (Example 7 and 8).

Note that the (optimal) probability matrix in Example 7 does not select disturbances uniformly at random. However, many HILS-based verification techniques simulate the SUV by selecting disturbances *uniformly at random*. That is, by choosing the probability matrix p for $T_\mathcal{A}$ so that for each node u and disturbance d, $p(u, d) = \frac{1}{|\{r|\langle u,r\rangle \in \text{prefix}(T_\mathcal{A})\}|}$. This may not yield an optimal simulation strategy (Remark 4).

Remark 4 *(A Non-Optimal Monte Carlo Simulation Strategy).* Let $\mathcal{D}$ be a set of disturbances with $|\mathcal{D}| \geq 2$. Then there exists an ASE $\mathcal{A} \subset \mathcal{D}^+$ such that selecting disturbances uniformly at random does not yield an optimal simulation strategy y (Example 8).

Example 8 *(Non-Optimal Monte Carlo Simulation Strategy).* Without loss of generality, let $\mathcal{D} = \{d_1, d_2\}$. Let $\mathcal{A} \subset \mathcal{D}^+$ be the ASE shown in Example 7 with $n|\mathcal{A}| = 3$, whose disturbance tree is shown in Fig. 1. Let p be the probability matrix for $T_\mathcal{A}$ such that $p(u, d) = \frac{1}{k(u)}$, where $k(u)$ is the out-degree of node u. In other words, at each step, p selects disturbances uniformly at random. Then, $P(\delta_1) = \frac{1}{2}$, $P(\delta_2) = P(\delta_3) = \frac{1}{4}$. By Theorem 1, an optimal strategy y^* consists in selecting uniformly at random the simulation campaigns in $\text{Sim}(\mathcal{A})$. This yields a payoff $\text{MiniMaxEVT} = \frac{n+1}{2} = 2$. The Monte Carlo simulation strategy y for $(\mathcal{A}, p)$ is such that $\text{WCEVT}(y) > 2$, thus y is not optimal. By definition of $(\mathcal{A}, p)$, simulation strategy and Example 3, we have: $y(\sigma_1) = \frac{1}{4}$, $y(\sigma_2) = \frac{1}{4}$, $y(\sigma_3) = \frac{1}{6}$, $y(\sigma_4) = \frac{1}{12}$, $y(\sigma_5) = \frac{1}{6}$, and $y(\sigma_6) = \frac{1}{12}$.

We can compute $\text{WCEVT}(y) = \max_{x^* \in X^*} \text{EVT}(x^*, y)$ by considering only pure error injection strategies x^* (Lemma 1). In this case, $X^* = \{x_1^*, x_2^*, x_3^*\}$ (see Definition 4), and the values for $\text{EVT}(x_i^*, y)$ for $i \in [1, 3]$ are: $\text{EVT}(x_1^*, y) = \frac{5}{3}$, $\text{EVT}(x_2^*, y) = \frac{13}{6}$, and $\text{EVT}(x_3^*, y) = \frac{13}{6}$.

Thus $\text{WCEVT}(y) = \max\left\{\frac{5}{3}, \frac{13}{6}, \frac{13}{6}\right\} = \frac{13}{6} > 2$.

6. Proof of results

In this section we provide the proofs of the main results of this paper.

6.1. Theorem 1: Optimal value of MiniMaxEVT

In order to prove that $\text{MiniMaxEVT} = \frac{n+1}{2}$, we use the following two properties of MiniMaxEVT.

i) For any two-player zero-sum game, by the Minimax Theorem (see, e.g., [36]), we have: $\text{MiniMaxEVT} = \max_{x\in X} \min_{y \in Y} \text{EVT}(x, y) = \min_{y\in Y} \max_{x\in X} \text{EVT}(x, y)$.

ii) $\max_{x\in X} \min_{y\in Y} \text{EVT}(x, y) = \max_{x\in X} \min_{y^*\in Y^*} \text{EVT}(x, y^*)$. In fact, since the minimum is reachable for at least one pure strategy, we can write: $\min_{y\in Y} \text{EVT}(x, y) \geq \min_{y^*\in Y^*} \text{EVT}(x, y^*)$. On the other hand, since $Y^* \subseteq Y$, we have $\min_{y\in Y} \text{EVT}(x, y) \leq \min_{y^*\in Y^*} \text{EVT}(x, y^*)$, and consequently the property.

To prove the theorem it is sufficient to show that

$$V_1 = \max_{x\in X} \min_{y\in Y} \text{EVT}(x, y) \geq \frac{n+1}{2}$$

and that

$$V_2 = \min_{y\in Y} \max_{x\in X} \text{EVT}(x, y) \leq \frac{n+1}{2}.$$

Since $\text{MiniMaxEVT} = V_1 = V_2$, we may then conclude that $\text{MiniMaxEVT} = \frac{n+1}{2}$.

We start by showing that $V_1 \geq \frac{n+1}{2}$.

From property *ii)* we can write:

$$\begin{aligned} V_1 = \;& \max_{x\in X} \min_{y\in Y} \text{EVT}(x, y) = \\ & \max_{x\in X} \min_{y^*\in Y^*} \text{EVT}(x, y^*) = \\ & \max_{x\in X} \min_{\sigma\in\text{Sim}(\mathcal{A})} \textstyle\sum_{\delta\in\mathcal{A}} x(\delta)\chi(\sigma, \delta) = \\ & \max_{x\in X} \min_{1\leq j\leq n!} \textstyle\sum_{i=1}^{n} x(\delta_i)\chi(\sigma_j, \delta_i). \end{aligned}$$

By choosing the uniform error injection strategy $\hat{x}(\delta) \equiv \frac{1}{n}$ (uniform strategy over set $\mathcal{A}$), we have:

$$\begin{aligned} V_1 \geq \;& \max_{x\in X} \min_{1\leq j\leq n!} \textstyle\sum_{i=1}^{n} \hat{x}(\delta_i)\chi(\sigma_j, \delta_i) = \\ & \min_{1\leq j\leq n!} \textstyle\sum_{i=1}^{n} \frac{1}{n}\chi(\sigma_j, \delta_i) = \\ & \frac{1}{n} \min_{1\leq j\leq n!} \textstyle\sum_{i=1}^{n} i = \frac{1}{n}\sum_{i=1}^{n} i \end{aligned}$$

which implies $V_1 \geq \frac{n+1}{2}$.

In order to show that $V_2 \leq \frac{n+1}{2}$, we consider the uniform simulation strategy $\hat{y}(\sigma) \equiv \frac{1}{n!}$ and also use Lemma 1. Then we have:

$$\begin{aligned} V_2 = \;& \min_{y\in Y} \max_{x\in X} \text{EVT}(x, y) \leq \\ & \max_{x\in X} \text{EVT}(x, \hat{y}) = \max_{x^*\in X^*} \text{EVT}(x^*, \hat{y}) = \\ & \max_{1\leq i\leq n} \textstyle\sum_{j=1}^{n!} \chi(\sigma_j, \delta_i)\hat{y}(\sigma_j) = \\ & \frac{1}{n!} \max_{1\leq i\leq n} \textstyle\sum_{j=1}^{n!} \chi(\sigma_j, \delta_i) = \\ & \frac{1}{n!} \max_{1\leq i\leq n} \textstyle\sum_{h=1}^{n} (n-1)!h = \\ & \frac{(n-1)!}{n!} \textstyle\sum_{h=1}^{n} h = \frac{n+1}{2}. \end{aligned}$$

Hence $\frac{n+1}{2} \leq V_1 = V_2 \leq \frac{n+1}{2}$.

6.2. Theorem 1: Optimality of simulation strategies

Without loss of generality, we focus on the case in which the adversary places just one counterexample (see Section 1.1), say in scenario δ_i. The probability that δ_i is in position t in a simulation campaign $\sigma \in \text{Sim}(\mathcal{A})$ is $\sum_{\chi(\sigma,\delta_i)=t} y(\sigma)$. Multiplying by t and summing out we have the expected value for the verification time. Then a simulation strategy $y \in Y$ is optimal if and only if it satisfies statement 1 of Theorem 1, namely: $\sum_{t=1}^{n} t \sum_{\chi(\sigma,\delta_i)=t} y(\sigma) = \frac{n+1}{2}$ for $i \in [1, n]$.

6.3. Theorem 1: Optimality of uniform simulation strategy

In order to prove that the uniform simulation strategy $\hat{y} \equiv \frac{1}{n!}$ is an optimal one for the verifier player, it is sufficient to rewrite the expression for EVT by using $\hat{y}$. We have: $\sum_{t=1}^{n} t \sum_{\chi(\sigma,\delta_i)=t} y(\sigma) = \sum_{t=1}^{n} t \sum_{\chi(\sigma,\delta_i)=t} \frac{1}{n!}$. Observing that the number of simulation campaigns having δ_i in position t is $(n-1)!$, we have $\sum_{t=1}^{n} t \frac{1}{n!}(n-1)! = \frac{n(n+1)}{2}\frac{1}{n} = \frac{n+1}{2}$.

6.4. Proposition 1: Sufficient condition for optimality of Monte Carlo simulation strategy

From the hypothesis it follows that, for all $\langle \delta_1, \ldots, \delta_n \rangle \in \text{Sim}(\mathcal{A})$:

$$y_{\mathcal{A}}(\langle \delta_1, \ldots, \delta_n \rangle) = \prod_{i=1}^{n} \frac{P(\delta_i)}{1-\sum_{k=1}^{i-1} P(\delta_k)} =$$

$$\prod_{i=1}^{n} \frac{\frac{1}{n}}{1-\sum_{k=1}^{i-1} \frac{1}{n}} = \prod_{i=1}^{n} \frac{\frac{1}{n}}{\frac{n-i+1}{n}} = \prod_{i=1}^{n} \frac{1}{n-i+1} = \frac{1}{n!}.$$

By Theorem 1 this is an optimal simulation strategy.

7. Conclusions

In the framework of simulation-based verification we addressed the problem of identifying an ordering on the scenarios (i.e., sequences of disturbances) to be simulated so as to minimise the maximum expected time to find an error (WCEVT). Our results can be summarised as follows.

First, the minimum WCEVT is $\frac{n+1}{2}$, where n is the number of scenarios to be simulated.

Second, there is an infinite set of optimal simulation strategies, i.e., strategies for which the minimum WCEVT is attained. Furthermore, we show that such a set forms a bounded convex polytope.

Third, ordering *simulation scenarios* uniformly at random yields an optimal simulation strategy.

Fourth, within an *online Monte Carlo*–based simulation setting, we show how to select probability distribution on *disturbances* so that the resulting simulation strategy is optimal.

Acknowledgements

This research has been partially supported by FP7 projects SmartHG (317761) and PAEON (600773).

References

[1] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivančić, A. Gupta, Probabilistic temporal logic falsification of cyber-physical systems, ACM Trans. Embed. Comput. Syst. 12 (2s) (2013).
[2] H. Abbas, B. Hoxha, G. Fainekos, K. Ueda, Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems, in: Proc. IEEE CYBER 2014, IEEE, 2014.
[3] A. Adir, E. Almog, L. Fournier, E. Marcus, M. Rimon, M. Vinov, A. Ziv, Genesys-Pro: innovations in test program generation for functional processor verification, IEEE Des. Test Comput. 21 (2) (2004).
[4] V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, E. Tronci, Automatic control software synthesis for quantized discrete time hybrid systems, in: Proc. CDC 2012, IEEE, 2012.
[5] V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, E. Tronci, On model based synthesis of embedded control software, in: Proc. EMSOFT 2012, ACM, 2012.
[6] V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, E. Tronci, A map-reduce parallel approach to automatic synthesis of control software, in: Proc. SPIN 2013, in: Lect. Notes Comput. Sci., vol. 7976, Springer, 2013.
[7] V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, E. Tronci, On-the-fly control software synthesis, in: Proc. SPIN 2013, in: Lect. Notes Comput. Sci., vol. 7976, Springer, 2013.
[8] R. Alur, Formal verification of hybrid systems, in: Proc. EMSOFT 2011, ACM, 2011.
[9] A. Arcuri, M. Iqbal, L. Briand, Random testing: theoretical results and practical implications, IEEE Trans. Softw. Eng. 38 (2) (2012).
[10] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Liebana, S. Samothrakis, S. Colton, A survey of Monte Carlo tree search methods, IEEE Trans. Comput. Intell. AI Games 4 (1) (2012).
[11] E. Clarke, A. Donzé, A. Legay, On simulation-based probabilistic model checking of mixed-analog circuits, Form. Methods Syst. Des. 36 (2) (2010).
[12] E. Clarke, O. Grumberg, D. Peled, Model Checking, MIT, 1999.
[13] E. Clarke, T. Henzinger, H. Veith, Handbook of Model Checking, Springer, 2016.
[14] R. Dechter, K. Kask, E. Bin, R. Emek, Generating random solutions for constraint satisfaction problems, in: Proc. AAAI 2002, AAAI, 2002.
[15] G. Della Penna, B. Intrigila, I. Melatti, E. Tronci, M. Venturini Zilli, Finite horizon analysis of Markov chains with the Murphi verifier, Int. J. Softw. Tools Technol. Transf. 8 (4–5) (2006).
[16] A. Dokhanchi, A. Zutshi, R. Sriniva, S. Sankaranarayanan, G. Fainekos, Requirements driven falsification with coverage metrics, in: Proc. EMSOFT 2015, IEEE, 2015.
[17] P. Duggirala, S. Mitra, M. Viswanathan, M. Potok, C2E2: a verification tool for stateflow models, in: Proc. TACAS 2015, in: Lect. Notes Comput. Sci., vol. 9035, Springer, 2015.
[18] C. Grimm, C. Radojicic, Verification and validation of AMS systems: towards coverage of uncertainties, in: Proc. IMSTW 2015, IEEE, 2015.
[19] R. Grosu, S. Smolka, Monte Carlo model checking, in: Proc. TACAS 2005, in: Lect. Notes Comput. Sci., vol. 3440, Springer, 2005.
[20] B. Hayes, I. Melatti, T. Mancini, M. Prodanovic, E. Tronci, Residential demand management using individualised demand aware price policies, IEEE Trans. Smart Grid (2016), http://dx.doi.org/10.1109/TSG.2016.2596790.
[21] D. Jansen, J. Katoen, M. Oldenkamp, M. Stoelinga, I. Zapreev, How fast and fat is your probabilistic model checker? An experimental performance comparison, in: Proc. HVC 2007, in: Lect. Notes Comput. Sci., vol. 4899, Springer, 2008.
[22] S. Jha, E. Clarke, C. Langmead, A. Legay, A. Platzer, P. Zuliani, A bayesian approach to model checking biological systems, in: Proc. CMSB 2009, in: Lect. Notes Comput. Sci., vol. 5688, Springer, 2009.
[23] K. Kalajdzic, C. Jégourel, A. Lukina, E. Bartocci, A. Legay, S. Smolka, R. Grosu, Feedback control for statistical model checking of cyber-physical systems, in: Proc. ISoLA 2016, in: Lect. Notes Comput. Sci., vol. 9952, Springer, 2016.
[24] T. Mancini, F. Mari, A. Massini, I. Melatti, F. Merli, E. Tronci, System level formal verification via model checking driven simulation, in: Proc. CAV 2013, in: Lect. Notes Comput. Sci., vol. 8044, Springer, 2013.
[25] T. Mancini, F. Mari, A. Massini, I. Melatti, E. Tronci, Anytime system level verification via random exhaustive hardware in the loop simulation, in: Proc. DSD 2014, IEEE, 2014.
[26] T. Mancini, F. Mari, A. Massini, I. Melatti, E. Tronci, System level formal verification via distributed multi-core hardware in the loop simulation, in: Proc. PDP 2014, IEEE, 2014.
[27] T. Mancini, F. Mari, A. Massini, I. Melatti, E. Tronci, SyLVaaS: system level formal verification as a service, in: Proc. PDP 2015, IEEE, 2015.
[28] T. Mancini, F. Mari, A. Massini, I. Melatti, E. Tronci, Anytime system level verification via parallel random exhaustive hardware in the loop simulation, MicPro 41 (2016).
[29] T. Mancini, F. Mari, I. Melatti, I. Salvo, E. Tronci, J. Gruber, B. Hayes, M. Prodanovic, L. Elmegaard, Demand-aware price policy synthesis and verification services for smart grids, in: Proc. SmartGridComm 2014, IEEE, 2014.
[30] F. Mari, I. Melatti, I. Salvo, E. Tronci, Synthesis of quantized feedback control software for discrete time linear hybrid systems, in: Proc. CAV 2010, in: Lect. Notes Comput. Sci., vol. 6174, Springer, 2010.

[31] F. Mari, I. Melatti, I. Salvo, E. Tronci, Undecidability of quantized state feedback control for discrete time linear hybrid systems, in: Proc. ICTAC 2012, in: Lect. Notes Comput. Sci., vol. 7521, Springer, 2012.

[32] F. Mari, I. Melatti, I. Salvo, E. Tronci, Model based synthesis of control software from system level formal specifications, ACM Trans. Softw. Eng. Methodol. 23 (1) (2014).

[33] S. Sankaranarayanan, R. Chang, G. Jiang, F. Ivancic, State space exploration using feedback constraint generation and Monte-Carlo sampling, in: Proc. ACM SIGSOFT 2007, ACM, 2007.

[34] A. Schrijver, Theory of Linear and Integer Programming, Wiley, 1998.

[35] H. Sivaraj, G. Gopalakrishnan, Random walk based heuristic algorithms for distributed memory model checking, Electron. Notes Theor. Comput. Sci. 89 (1) (2003).

[36] L. Thomas, Games, Theory and Applications, Dover, 1980.

[37] E. Tronci, G. Della Penna, B. Intrigila, M. Venturini Zilli, A probabilistic approach to automatic verification of concurrent systems, in: Proc. APSEC 2001, IEEE, 2001.

[38] E. Tronci, T. Mancini, I. Salvo, S. Sinisi, F. Mari, I. Melatti, A. Massini, F. Davì, T. Dierkes, R. Ehrig, S. Röblitz, B. Leeners, T.H.C. Krüger, M. Egli, F. Ille, Patient-specific models from inter-patient biological models and clinical records, in: Proc. FMCAD 2014, IEEE, 2014.

[39] G. Verzino, F. Cavaliere, F. Mari, I. Melatti, G. Minei, I. Salvo, Y. Yushtein, E. Tronci, Model checking driven simulation of sat procedures, in: SpaceOps 2012, 2012.

[40] C.-H. Yang, G. Zhabelova, C.-W. Yang, V. Vyatkin, Cosimulation environment for event-driven distributed controls of smart grid, IEEE Trans. Ind. Inform. 9 (3) (2013).

[41] P. Zuliani, A. Platzer, E. Clarke, Bayesian statistical model checking with application to Stateflow/Simulink verification, Form. Methods Syst. Des. 43 (2) (2013).

[42] A. Zutshi, S. Sankaranarayanan, J. Deshmukh, X. Jin, Symbolic-numeric reachability analysis of closed-loop control software, in: Proc. HSCC 2016, ACM, 2016.