Estimation-Based Verification of Cyber-Physical Systems via Statistical Model Checking

Marco Esposito, Leonardo Picchiami*

Computer Science Dept., Sapienza University of Rome, via Salaria 113, 00198, Italy

Abstract

In this paper, we focus on verifying Cyber-Physical System via Statistical Model Checking and numerical simulation. Our verification problem is to establish whether the expected value of a given Key Performance Indicator exceeds the desired threshold. We propose to use an optimal approximation algorithm to get an (ϵ, δ)-approximation of a desired expected value through Monte Carlo experiments.

We prove the feasibility of our approach on the Pumping System, an example derived from the Modelica Standard Library. We were able to verify the system in a reasonable time, taking advantage of a parallel implementation of the approximation algorithm.

Keywords

Statistical Model Checking, Verification, Cyber-Physical Systems, Numerical Simulation.

1. Introduction

Many mission/safety-critical Cyber-Physical Systems (CPSs) [3], such as, for example, smart grids [24, 33, 46, 37], automotive systems [20, 11, 62] and biological systems [25, 27, 28, 41, 53, 54, 18, 17, 19, 57, 2, 5], must be provably correct. There are several limitations to overcome to formally verify CPSs. For example, the huge number of operational scenarios (*scenario explosion*, *e.g.*, [34, 39, 40, 45]) to be considered or the ever-increasing complexity of the systems. They make the problem intractable for approaches such as numerical techniques, logics or automata (such as,*e.g.*, those discussed in [15, 16, 10, 9, 30, 21, 29, 7, 43, 12, 42]).

Statistical Model Checking (SMC) holds the promise to reduce time and cost needed for the verification. SMC is a Monte Carlo-based approach that aims at *sampling* scenarios until the desired statistical assurance over a certain property is reached. Mainly, it takes advantage of *Hypothesis Testing* [22, 35, 58], *Estimation* [33, 44, 47, 13] and *Bayesian analysis* [63, 8] to evaluate *qualitative* as well as *quantitative* properties of interest. The verification of qualitative properties consists in choosing between two mutually exclusive

HYDRA - RCRA 2022: 1st International Workshop on HYbrid Models for Coupling Deductive and Inductive ReAsoning and 29th RCRA workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion

*Corresponding author.

EMAIL: esposito@di.uniroma1.it (M. Esposito); picchiami@di.uniroma1.it (L. Picchiami)

ORCID: 0000-0003-4543-8818 (M. Esposito); 0000-0001-5477-6419 (L. Picchiami)

hypotheses with a given statistical confidence, whereas quantitative properties provide a measurable interpretation of some stochastic behaviour.

The literature (see, *e.g.*, [1, 49]) shows a plethora of tools (*e.g.*, MultiVeSta [50], PRISM [26], UPPAAL-SMC [14], COSMOS [6], Ymer [60]) that enable SMC via numerical simulation. Many SMC approaches, even based on such tools, require the system modelling through some kind of structure (*e.g.*, *Discrete Time Markov Chain*, *Continuous Time Markov Chain* [61, 4], *Probabilistic Timed Automata* [48]), and a specification through a class of logic (*e.g.*, *Probabilistic Computational Tree Logic* [23], *Continuous Stochastic Logic* [52]) to carry out the verification. We note that specific modelling of the system is needed to generate trajectories on demand.

In this paper, instead, we focus on verifying CPSs by relying on a purely black-box approach where the System Under Verification (SUV) is available only via a *simulator* (*e.g.*, OpenModelica, Dymola, Simulink). Our setting requires establishing if the expected value of a given Key Performance Indicator (KPI) does not exceed a given threshold. We propose to estimate such an expected value using an optimal approximation algorithm (namely, the *Approximation Algorithm*, $\mathcal{AA}$ [13]) that interfaces in a black-box fashion [55] with the SUV. Mainly, the approximation algorithm asks for a new sample when needed and outputs the estimate when enough samples have been collected. So, our approach aims to be compliant with any simulation driver and SUV model. Furthermore, we prove the feasibility of such an approach on a Modelica model. Note that, to the best of our knowledge, no other contributions exist that apply an estimation algorithm on a Modelica system for purely verification purposes.

Black-box approaches distant from ours have been proposed in, *e.g.*, [51, 59]; in those works, the authors use hypothesis testing to verify qualitative properties when trajectories can not be generated on demand.

The paper is organised as follows. We provide a formalisation of the verification problem in Section 2 and a description of the tool in Section 3. In Section 4, we present the case study used to prove the feasibility of our approach, whereas, in Section 5, we summarise and discuss the results obtained. Finally, we draw our conclusions in Section 6.

2. Background

We denote with $\mathbb{R}$, $\mathbb{R}_{0+}$, and $\mathbb{R}_{+}$, respectively, the sets of all, non-negative and positive reals. $\mathbb{N}$ and $\mathbb{N}_{+}$ are the sets of non-negative and strictly-positive integers.

2.1. Statistical model checking

The system verification via SMC consists of three components: the SUV, the specifications, and the SUV environment. In our setting, a SUV is typically a simulable CPS model, *i.e.*, a system described by continuous as well as discrete variables. The continuous part describes the physical component, whereas the discrete part models the dynamics of the software component. We treat (see Definition 1) our SUV as a black-box input-output

deterministic causal dynamical system (see, *e.g.*, [56, 32, 38]) that depends on its inputs and outputs and can be evaluated only through simulation.

Definition 1. *A black-box SUV $\mathcal{S}$ is a tuple $(\mathcal{T}, U, Y, S)$ where $\mathcal{T}$ is the time-set, U and Y are the input and output spaces and $S : U^{\mathcal{T}} \to Y^{\mathcal{T}}$ is the output (simulation) function that defines the time evolution of the system for each time point in $\mathcal{T}$.*

A black-box SUV $\mathcal{S}$ is strictly causal if, for any $t \in \mathcal{T}$ and for any $\mathbf{u}_1, \mathbf{u}_2 \in U^{\mathcal{T}}$ such that $\mathbf{u}_1(t') = \mathbf{u}_2(t')$ for all $t' < t$, it holds $S(t; \mathbf{u}_1) = S(t; \mathbf{u}_2)$ where $S(t; \mathbf{u})$ denotes the value $S(\mathbf{u})$ at time t.

The SUV environment defines all *operational scenarios* the SUV must withstand (*e.g.*, see [31, 36, 45]). Such inputs are *uncontrollable*, *i.e.*, not under the system's control (*e.g.*, faults or disturbances). Since we need to sample the operational scenarios needed for the verification, we focus of finitely parametrisable environments, and sample within such parameter spaces.

Definition 2. *A finitely parametrisable SUV environment is a function $I : W \to U^{\mathcal{T}}$ with $W \subseteq \mathbb{R}^m$, $m \in \mathbb{N}$.*

Of course, not all scenarios have the same occurrence. We model this feature through a probability density function $p_W(w)$ on the set of environment parameters W. This seamlessly includes both discrete and continuous parameter spaces.

The SUV must satisfy several requirements for all possible scenarios. The specifications of such requirements model KPIs on the safety properties of the system. Note that in our framework, an output function $\mathbf{y} \in Y^{\mathcal{T}}$ of the SUV can be a system trajectory (possibly needed to compute a KPI) and the KPI itself. In particular, we are interested in *quantitative* specifications that give a metric interpretation of the properties of interest.

Definition 3. *A quantitative KPI is a function $f : Y^{\mathcal{T}} \to [0, 1]$.*

2.2. Verification problem

We want to know whether the expected value of a given KPI over the set of operational scenarios is less than or equal to a given threshold P. We need to compute the expected value μ of $f(S(\mathbf{u}))$ for any $\mathbf{u} = I(w)$ for some $w \in W$, where μ ranges in [0, 1] with probability density function $p_W(w)$ on W. Formally,

$$\mu = \int_W f(S(I(w)))p_W(w)dw \tag{1}$$

So we can define the verification problem as follows:

$$\int_W f(S(I(w)))p_W(w)dw - P \leq 0 \tag{2}$$

Unfortunately, W may be, and usually is, infinite and computing all operational scenarios is impossible and, even if they are a finite number, typically prohibitive. To avoid such a scenario explosion, we can use Monte Carlo based approaches that provide

an (ϵ, δ)-approximation $\hat{\mu}$ of $\int_W f(S(I(w)))p_W(w)dw$, that is, with probability at least $1-\delta$,

$$\mu(1-\epsilon) \leq \hat{\mu} \leq \mu(1+\epsilon) \tag{3}$$

with $0 < \epsilon \leq 1$ and $0 < \delta \leq 1$.

3. CPSs Verification via Statistical Model Checking

We focus on verifying CPSs via Statistical Model Checking. Our goal is to perform Monte Carlo experiments to estimate the expected value of a given KPI and establish whether it exceeds the desired threshold. We may empirically compute the expected value through a Monte Carlo approach that uses an arbitrary large number of samples. Unfortunately, such an approach does not provide any formal guarantees, in terms of error bound and statistical confidence, about the accuracy of the expected value estimate.

To overcome this obstacle, we adopt the optimal *Approximation Algorithm* ($\mathcal{AA}$) [13] as our estimator. $\mathcal{AA}$ computes an (ϵ, δ)-approximation $\tilde{\mu}_Z$ of the expected value $\mu_Z > 0$ of a univariate non-negative random variable Z, *i.e.*, the computed estimate $\tilde{\mu}_Z$ of the expected value μ_Z of Z is guaranteed to be within $\mu(1-\epsilon)$ and $\mu(1+\epsilon)$ with probability at least $1-\delta$. The algorithm optimally uses Monte Carlo experiments for the approximation, that is, within a constant factor, it requires the minimum number of experiments to output the estimate. Note that in our setting, we get samples via numerical simulation and a non-optimal estimator could be very expensive. Note that in case an absolute (and not relative) error is sought, other algorithms might seamlessly be used, *e.g.*, based on the Bernstein Inequality [13].

3.1. Univariate Approximation Algorithm AA

Below we briefly summarise the main concepts of the $\mathcal{AA}$. For a detailed explanation see [13].

$\mathcal{AA}$ is the three-phase algorithm illustrated in Algorithm 1. It computes an approximation $\tilde{\mu}_Z$ of the mean $\mu_Z > 0$ of a univariate non-negative random variable Z distributed in $[0, 1]$.

Given the user-defined input parameters $\epsilon \in (0, 1]$ and $\delta \in (0, 1]$, the following values are defined: $\lambda = (e-2) \approx .72$, $\Upsilon = 4\lambda ln(2/\delta)/\epsilon^2$ and $\rho_Z = max\{\sigma_Z^2, \epsilon\mu_Z\}$. $\mathcal{AA}$ uses the first two phases to decide the number of samples required to output $\tilde{\mu}_Z$ in the third phase. In the first phase, it computes an initial $(\sqrt{\epsilon}, \delta/3)$-approximation $\hat{\mu}_Z$ of the mean μ_Z using the Stopping Rule Algorithm (SRA). Note that in this phase the number of samples cannot be determined in advance. The second phase (Estimate Variance, EV) provides an initial estimate $\hat{\rho}_Z$ of the variance σ_Z^2 using $2 \cdot N_2$ samples where $N_2 = \Upsilon_2 \cdot \epsilon/\hat{\mu}$. $\hat{\rho}_Z$ is within a constant factor ρ_Z with probability of at least $1-\delta$.

Estimate Mean (EM) is the third phase of the $\mathcal{AA}$ algorithm. It outputs an (ϵ, δ)-approximation $\tilde{\mu}_Z$ of μ_Z by taking advantage from $\hat{\mu}_Z$ and $\hat{\rho}_Z$ with $N_3 = \Upsilon_2 \cdot \hat{\rho}_Z/\hat{\mu}_Z^2$ samples.

Algorithm 1: Approximation Algorithm

Input: error bound ϵ, statistical confidence δ

function: $\mathcal{AA}(\epsilon, \delta)$

1: $\Upsilon_2 \leftarrow 2(1+\sqrt{\epsilon})(1+2\sqrt{\epsilon})(1+ ln(\frac{2}{3})/ln(\frac{2}{\delta}))\Upsilon$
2: $\hat{\mu}_Z \leftarrow \text{SRA}(min(1/2, \sqrt{\epsilon}), \delta/3)$
3: $\hat{\rho}_Z \leftarrow \text{EV}(\hat{\mu}_Z, \epsilon, \Upsilon_2)$
4: $\tilde{\mu}_Z \leftarrow \text{EM}(\hat{\mu}_Z, \hat{\rho}_Z, \Upsilon_2)$
5: **return** $\tilde{\mu}_Z$

function: $\text{SRA}(\epsilon, \delta)$

1: $\Upsilon_1 \leftarrow 1 + (1+\epsilon)\Upsilon$
2: $N \leftarrow 0$
3: $S \leftarrow 0$
4: **while** $S < \Upsilon_1$ **do**
5: $\quad N \leftarrow N + 1$
6: $\quad S \leftarrow S + Z_N$
7: **return** S/N

function: $\text{EV}(\hat{\mu}_Z, \epsilon, \Upsilon_2)$

1: $N \leftarrow \Upsilon_2 \cdot \epsilon / \hat{\mu}_Z$
2: $S \leftarrow 0$
3: **for** i=1 to N **do**
4: $\quad S \leftarrow S + (Z'_{2i-1} - Z'_{2i})^2/2$
5: $\hat{\rho}_Z \leftarrow max\{S/N, \epsilon\hat{\mu}_Z\}$
6: **return** $\tilde{\rho}_Z$

function: $\text{EM}(\hat{\mu}, \hat{\rho}, \Upsilon_2)$

1: $N \leftarrow \Upsilon_2 \cdot \hat{\rho}_Z / \hat{\mu}_Z^2$
2: $S \leftarrow 0$
3: **for** i=1 to N **do**
4: $\quad S \leftarrow S + Z_i$
5: $\tilde{\mu}_Z \leftarrow S/N$
6: **return** $\tilde{\mu}_Z$

3.2. Parallel implementation

We developed a parallel implementation of $\mathcal{AA}$ based on a message-passing paradigm. The workflow of each $\mathcal{AA}$ phase is shown Figure 1. Our architecture envisions 1 master (orchestrator) and n slaves (workers). The orchestrator handles both the $\mathcal{AA}$ algorithm and the input for all workers, whereas each worker waits for new input and simulates the system to produce a new sample.

The orchestrator keeps the computation status of the $\mathcal{AA}$ algorithm and sends new input to idle workers as long as new samples are required. Our implementation takes into account whether too many samples have been produced at the end of the first phase so that we can use them during the second phase. In such a way, we efficiently use the computational power without wasting any model simulation. The dispatching of the inputs is centralised to sort the obtained samples deterministically. Such a feature is not strictly required, but ensures the full reproducibility of all experiments.

The $\mathcal{AA}$ is implemented as explained in the pseudocode of Section 3.1. The orchestrator updates the approximation algorithm only if new samples have been produced; otherwise, it freezes the algorithm. In such a context, the computational time required to produce new samples via simulation of the SUV model is the main bottleneck of the addressed problem, and the simulation time affects its performance.

Note that computing the expected value of multiple KPIs can be addressed seamlessly by using the *same* random samples, where, for each sample, the SUV simulator computes the value of all KPIs. Hence, the approximation algorithm terminates when enough samples are simulated so that (ϵ, δ)-approximations of *all* KPIs are computed.

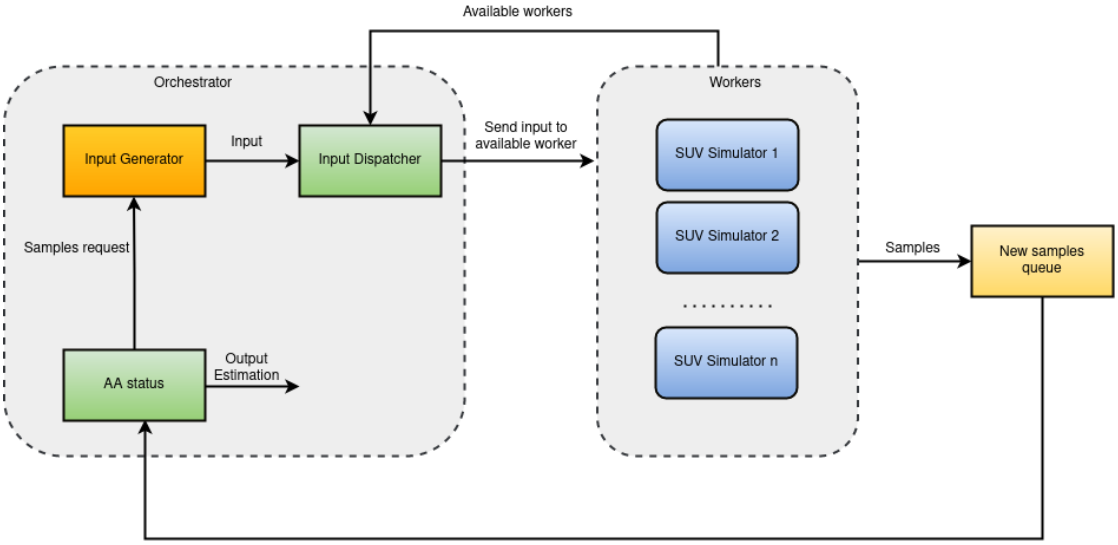


Figure 1: Parallel $\mathcal{AA}$ phase workflow.

4. Case Study

We evaluated the effectiveness of our approach on the Pumping System (PS) model provided by the Modelica Standard Library. PS represents a pumping control system for drinking water. The water is pumped from a source by a pump, fitted with check valves, into a reservoir. The reservoir is described by a (nominal) height H of 3 m and an area A of 50 m^2 and it is connected to another valve that models the users, *i.e.*, the outgoing water sink. The source and water sink work at an environmental pressure p (101 325 Pa) and temperature T (20 °C) to get the ingoing and outgoing water dynamics where p is the average value of atmospheric pressure in standard conditions. In addition, H models a constraint on the system dynamics, that is, the water level must be within $[0, H]$ during the course of the whole simulation.

The control component outputs the rotational speed of the pump engine through a first-order system. The component sends a small non-zero value to put the pump in a standby state when other water is not required, whereas other values indicate that the pump must inject the water into the reservoir.

Each experiment of our Monte Carlo-based approach simulates the system with an horizon h of 2000 seconds. After $t = 200$ seconds, PS opens the sink valve, and the controller starts turning on and off the pump to keep the level water value around 2.2 meters so that the gauge pressure is around 200 mbar.

We added a stochastic environment that injects a disturbance on the environmental pressure so that the ingoing and the outgoing water pressure are unbalanced, *i.e.*, the sink and source stochastic pressure take different values. Our stochastic pressure p_s ranges from $p(1 - \theta)$ to $p(1 + \theta)$ where θ is the percentage of disturbance. We chose a disturbance of $\theta = 0.07$ (7%) as it is the maximum value that still yields plausible

δ	ϵ	$\mathbb{E}[Z]$	Samples	Time (hh:mm:ss)
0.05	0.05	0.1472	19322.8	01:53:40
0.05	0.01	0.1474	127417.1	12:29:32
0.01	0.01	0.1474	176579.9	17:18:45
0.05	0.005	0.1473	375593.9	36:49:28
0.005	0.005	0.1473	584357.7	57:17:33

Table 1
Experimental results

trajectories (for instance, with $\theta = 0.08$, we witnessed trajectories in which the water level exceeded the tank height).

To verify the correctness of the PS, we modelled the Mean Relative Absolute Error (MRAE) as desired KPI on a current water level to quantify the ability of the system to keep it around 2.2 meters in presence of a stochastic environmental pressure. Given the current water level l_c and reference water level $l_r = 2.2$, we define the Relative Absolute Error (RAE) at time t as follows:

$$\text{RAE}(t) = \frac{|l_c(t) - l_r|}{l_r} \tag{4}$$

Thus, we define the MRAE at time t as follows:

$$\text{MRAE}(t) = \frac{\int_0^t \text{RAE}(\tau) d\tau}{t} \tag{5}$$

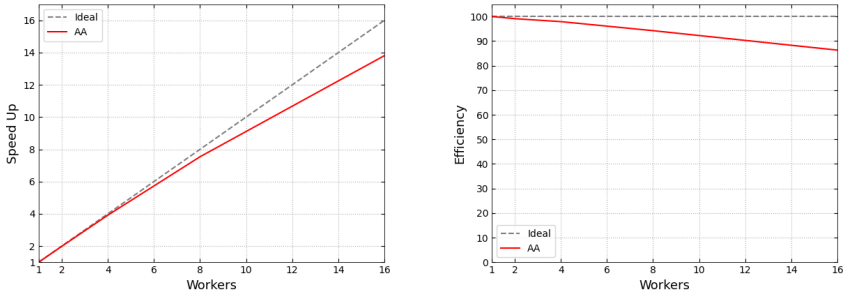
where t is the current time instant over the simulation. Finally, given $P_1 = |H - l_r|/l_r = 0.3637$ and $P_2 = |0 - l_r|/l_r = 1$, we set our threshold to $P = min(P_1, P_2)/2 = 0.1819$.

5. Results

In this section, we report the experimental results obtained on the PS. The experiments were run on a computer equipped with AMD EPYC 7301 16-Core CPU and 256 GB RAM. We estimated the expected value of the MRAE on the water level on several configurations for ϵ and δ. From preliminary experiments, we have observed that the simulation time drastically increases when we run more than 16 copies of the Modelica system in parallel. In our setting, the simulator is particularly expensive and requires so many computational resources that the simulation time degrades. This motivates the fact that we limited our experiment settings up to 16 workers.

Table 1 summarises the average results of $n = 10$ experiments using 1 orchestrator and 16 workers. It reports the average number of samples required by $\mathcal{AA}$, the average value of the estimated percentage error ($\mathbb{E}[Z]$) and the average value of the time needed to carry out the experiments.

In each configuration, on average, the PS has a percentage error of 14.7%. This indicates that our safety property is satisfied since $0.147 = \mathbb{E}[Z] \leq P = 0.182$ holds and



(a) Speed-Up. (b) Efficiency.

Figure 2: Performance scaling of our parallel $\mathcal{AA}$ implementation.

our KPI can range on average in $[2.2 - 0.323, 2.2 + 0.323]$ where 0.323 is 14.7% of the reference water level.

The system simulation time and hyperparameters ϵ and δ impact the experiments. Values for ϵ and δ (mainly ϵ) affect the number of samples needed to obtain the desired estimate. In fact, accurate estimations with higher statistical confidence, *i.e.*, smaller values for ϵ and δ, require more samples. The simulation time, instead, heavily dominates the execution time of our experiments in which we can get each sample only via numerical simulation of the SUV. Such an aspect is not negligible and becomes increasingly relevant when non-optimal approximation algorithms are used. In [47], the authors explain how approximation algorithms that do not take advantage of the variance have worse performance. Note that the number of required samples is the criterion to use to compare different Monte Carlo approaches. In such a context, having worse performance with the same ϵ and δ also increases the number of required simulations to estimate the expected value of the target KPI. For example, if a system takes 4-5 seconds for every simulation as in our case, performing too many of simulations could make the problem intractable even when using Statistical Model Checking.

Another observation is about the performance of the developed parallel tool. Figure 2a and Figure 2b illustrate, respectively, speed-up and efficiency by varying the number of workers used for the simulation with $\epsilon = \delta = 0.05$. Values for other configurations, even with smaller ϵ and δ, follow the same trend. The speed-up is the ratio between the time required using a single worker and the time required using n workers, whereas the efficiency is the ratio between the speed-up with n workers and n. Both figures show that we are able to efficiently parallelise up to 16 workers with an efficiency greater than 85% and a speed-up grater than 13.5x.

6. Conclusion

In this work, we performed formal verification of CPSs via Statistical Model Checking. In our setting, solving the verification problem means establishing whether the expected

value of a desired KPI does not exceed a given threshold P. We developed a parallel implementation of $\mathcal{AA}$ and evaluated our approach's viability on an example provided by the Modelica Standard Library, the Pumping System. We observed that, even in limited presence of disturbances, the system, on average, is able to satisfy the desired specification. An interesting aspect is the need for parallel approaches when dealing with simulation-based verification. Although ϵ and δ heavily impact the number of required samples by $\mathcal{AA}$, the parallelisation makes the verification feasible.

Several directions may be pursued in future work. Among those, we mention (1) the improvement of the efficiency of our parallel approach and comparison of several implementations to establish the most suitable parallel implementation of $\mathcal{AA}$, and (2) the systematic analysis of the behaviour of PS with stochastic environments that present stronger disturbances and bigger tanks.

Acknowledgments

This work was partially supported by: Italian Ministry of University and Research under grant "Dipartimenti di eccellenza 2018–2022" of the Department of Computer Science of Sapienza University of Rome; INdAM "GNCS Project 2020"; Sapienza University projects RG12117A8B393BDC, RG11816436BD4F21, RG11916B892E54DB, RP11916B8665242F; Lazio POR FESR projects E84G20000150006, F83G17000830007.

References

[1] G. Agha and K. Palmskog. A survey of statistical model checking. *ACM Trans Model Comput Simul*, 28(1), 2018.

[2] R. Allen, *et al.* Efficient generation and selection of virtual populations in quantitative systems pharmacology models. *CPT: Pharmacom Sys Pharmacol*, 5(3), 2016.

[3] R. Alur. *Principles of Cyber-Physical Systems.* MIT, 2015.

[4] C. Baier, *et al.* Model-checking algorithms for continous-time markov chains. *IEEE Trans Softw Eng*, 29, 2003.

[5] P. Balazki, *et al.* A quantitative systems pharmacology kidney model of diabetes associated renal hyperfiltration and the effects of sglt inhibitors. *CPT: Pharmacom Sys Pharmacol*, 7(12), 2018.

[6] P. Ballarini, *et al.* HASL: a new approach for performance evaluation and model checking from concepts to experimentation. *Perf Eval*, 90, 2015.

[7] L. Bordeaux, *et al.* CSP properties for quantified constraints: Definitions and complexity. In *AAAI 2005*. AAAI, 2005.

[8] L. Bortolussi, *et al.* Smoothed model checking for uncertain continuous-time markov chains. *Inf Comput*, 247, 2016.

[9] M. Cadoli and T. Mancini. Combining relational algebra, SQL, constraint modelling, and local search. *TPLP*, 7(1-2), 2007.

[10] M. Cadoli, *et al.* SAT as an effective solving technology for constraint problems. In *ISMIS 2006*, *LNCS*, vol. 4203. Springer, 2006.

[11] S. Chakraborty, *et al.* Automotive cyber–physical aystems: A tutorial introduction. *IEEE Des&Test*, 33(4), 2016.
[12] Q. Chen, *et al.* MILP, pseudo-boolean, and OMT solvers for optimal fault-tolerant placements of relay nodes in mission critical wireless networks. *Fundam Inform*, 174(3–4), 2020.
[13] P. Dagum, *et al.* An optimal algorithm for Monte Carlo estimation. *SICOMP*, 29(5), 2000.
[14] A. David, *et al.* Uppaal smc tutorial. *Int Soft Tech Trans*, 17(4), 2015.
[15] G. Della Penna, *et al.* Automatic verification of a turbogas control system with the murphi verifier. In *HSCC 2003*, *LNCS*, vol. 2623. Springer, 2003.
[16] G. Della Penna, *et al.* Finite horizon analysis of Markov chains with the Murphi verifier. *STTT*, 8(4–5), 2006.
[17] M. Esposito and L. Picchiami. Intelligent search for personalized cancer therapy synthesis: an experimental comparison. In *RCRA 2021*, *CEUR W.P.*, vol. 3065. CEUR, 2021.
[18] M. Esposito and L. Picchiami. Simulation-based synthesis of personalised therapies for colorectal cancer. In *OVERLAY 2021*, *CEUR W.P.*, vol. 2987. CEUR, 2021.
[19] M. Esposito and L. Picchiami. A comparative study of AI search methods for personalised cancer therapy synthesis in copasi. In *AI*IA 2022*, *LNCS*, vol. 13196. Springer, 2022.
[20] D. Goswami, *et al.* Challenges in automotive cyber-physical systems design. In *SAMOS 2012*. IEEE, 2012.
[21] G. Gottlob, *et al.* Conditional constraint satisfaction: Logical foundations and complexity. In *IJCAI 2007*. 2007.
[22] R. Grosu and S. Smolka. Monte Carlo model checking. In *TACAS 2005*, *LNCS*, vol. 3440. Springer, 2005.
[23] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Form Asp Comp*, 6(5), 1994.
[24] B. Hayes, *et al.* Residential demand management using individualised demand aware price policies. *IEEE Trans Smart Grid*, 8(3), 2017.
[25] M. Hengartner, *et al.* Negative affect is unrelated to fluctuations in hormone levels across the menstrual cycle: Evidence from a multisite observational study across two successive cycles. *J Psycho Res*, 99, 2017.
[26] M. Kwiatkowska, *et al.* Prism 4.0: Verification of probabilistic real-time systems. In *CAV 2011*, *LNCS*, vol. 6806. Springer, 2011.
[27] B. Leeners, *et al.* Associations between natural physiological and supraphysiological estradiol levels and stress perception. *Front Psycol*, 10, 2019.
[28] F. Maggioli, *et al.* SBML2Modelica: Integrating biochemical models within open-standard simulation ecosystems. *Bioinformatics*, 36(7), 2020.
[29] T. Mancini, *et al.* Evaluating ASP and commercial solvers on the CSPLib. *Constraints*, 13(4), 2008.
[30] T. Mancini, *et al.* Combinatorial problem solving over relational databases: View synthesis through constraint-based local search. In *SAC 2012*. ACM, 2012.
[31] T. Mancini, *et al.* System level formal verification via model checking driven

simulation. In *CAV 2013*, *LNCS*, vol. 8044. Springer, 2013.
[32] T. Mancini, *et al.* Anytime system level verification via random exhaustive hardware in the loop simulation. In *DSD 2014*. IEEE, 2014.
[33] T. Mancini, *et al.* Demand-aware price policy synthesis and verification services for smart grids. In *SmartGridComm 2014*. IEEE, 2014.
[34] T. Mancini, *et al.* System level formal verification via distributed multi-core hardware in the loop simulation. In *PDP 2014*. IEEE, 2014.
[35] T. Mancini, *et al.* Computing biological model parameters by parallel statistical model checking. In *IWBBIO 2015*, *LNCS*, vol. 9044. Springer, 2015.
[36] T. Mancini, *et al.* SyLVaaS: System level formal verification as a service. In *PDP 2015*. IEEE, 2015.
[37] T. Mancini, *et al.* User flexibility aware price policy synthesis for smart grids. In *DSD 2015*. IEEE, 2015.
[38] T. Mancini, *et al.* Anytime system level verification via parallel random exhaustive hardware in the loop simulation. *Microprocessors and Microsystems*, 41, 2016.
[39] T. Mancini, *et al.* SyLVaaS: System level formal verification as a service. *Fundam Inform*, 149(1–2), 2016.
[40] T. Mancini, *et al.* On minimising the maximum expected verification time. *Inf Proc Lett*, 122, 2017.
[41] T. Mancini, *et al.* Computing personalised treatments through in silico clinical trials. A case study on downregulation in assisted reproduction. In *RCRA 2018*, *CEUR W.P.*, vol. 2271. CEUR, 2018.
[42] T. Mancini, *et al.* An efficient algorithm for network vulnerability analysis under malicious attacks. In *ISMIS 2018*. Springer, 2018.
[43] T. Mancini, *et al.* Optimal fault-tolerant placement of relay nodes in a mission critical wireless network. In *RCRA 2018*, *CEUR W.P.*, vol. 2271. CEUR, 2018.
[44] T. Mancini, *et al.* Parallel statistical model checking for safety verification in smart grids. In *SmartGridComm 2018*. IEEE, 2018.
[45] T. Mancini, *et al.* Any-horizon uniform random sampling and enumeration of constrained scenarios for simulation-based formal verification. *IEEE TSE*, 2021.
[46] I. Melatti, *et al.* A two-layer near-optimal strategy for substation constraint management via home batteries. *IEEE Trans Ind Elect*, 69(8), 2022.
[47] V. Mnih, *et al.* Empirical bernstein stopping. In *ICML 2008*. Ass. Comp. Mach., 2008.
[48] G. Norman, *et al.* Model checking for probabilistic timed automata. *Form Meth Sys Des*, 43(2), 2013.
[49] A. Pappagallo, *et al.* Monte Carlo based Statistical Model Checking of Cyber-Physical Systems: a Review. *Inf*, 11(12), 2020.
[50] S. Sebastio and A. Vandin. MultiVeStA: Statistical model checking for discrete event simulators. In *ValueTools 2013*. ICST/ACM, 2013.
[51] K. Sen, *et al.* "statistical model checking of black-box probabilistic systems". In *CAV 2004*, *LNCS*, vol. 3114. Springer, 2004.
[52] K. Sen, *et al.* On statistical model checking of stochastic systems. In *CAV 2005*, *LNCS*, vol. 3576. Springer, 2005.

[53] S. Sinisi, *et al.* Complete populations of virtual patients for in silico clinical trials. *Bioinformatics*, 36(22–23), 2020.
[54] S. Sinisi, *et al.* Optimal personalised treatment computation through in silico clinical trials on patient digital twins. *Fundam Inform*, 174(3–4), 2020.
[55] S. Sinisi, *et al.* Reconciling interoperability with efficient verification and validation within open source simulation environments. *Simul Model Pract Theory*, 109, 2021.
[56] E. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems (2nd Ed.)*. Springer, 1998.
[57] D. Teutonico, *et al.* Generating virtual patients by multivariate and discrete re-sampling techniques. *Pharm Res*, 32(10), 2015.
[58] E. Tronci, *et al.* Patient-specific models from inter-patient biological models and clinical records. In *FMCAD 2014*. IEEE, 2014.
[59] H. Younes. Probabilistic verification for "black-box" systems. In *CAV 2005*, *LNCS*, vol. 3576. Springer, 2005.
[60] H. Younes. Ymer: A statistical model checker. In *CAV 2005*, *LNCS*, vol. 3576. Springer, 2005.
[61] H. Younes and R. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *CAV 2002*, *LNCS*, vol. 2404. Springer, 2002.
[62] L. Zhang. Modeling automotive cyber physical systems. In *DCABES 2013*. IEEE, 2013.
[63] P. Zuliani, *et al.* Bayesian statistical model checking with application to Stateflow/Simulink verification. *Form Meth Sys Des*, 43(2), 2013.