$\mathcal{S}$BIP 2.0: Statistical Model Checking Stochastic Real-Time Systems

Braham Lotfi Mediouni[1](✉), Ayoub Nouri[1], Marius Bozga[1], Mahieddine Dellabani[1], Axel Legay[2], and Saddek Bensalem[1]

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP*, VERIMAG, 38000 Grenoble, France
braham-lotfi.mediouni@univ-grenoble-alpes.fr
[2] INRIA, Rennes, France

Abstract. This paper presents a major new release of $\mathcal{S}$BIP, an extensible statistical model checker for Metric (MTL) and Linear-time Temporal Logic (LTL) properties on respectively Generalized Semi-Markov Processes (GSMP), Continuous-Time (CTMC) and Discrete-Time Markov Chain (DTMC) models. The newly added support for MTL, GSMPs, CTMCs and rare events allows to capture both real-time and stochastic aspects, allowing faithful specification, modeling and analysis of real-life systems. $\mathcal{S}$BIP is redesigned as an IDE providing project management, model edition, compilation, simulation, and statistical analysis.

1 Introduction

Statistical Model Checking (SMC) is a powerful alternative to classical numerical probabilistic model-checking that generally fail to handle large state-space systems. SMC was successfully applied in the assessment of different real-life systems in various application domains. Classical model checkers [4,8] now include SMC as part of their analysis engines and have been recently joined by a variety of specialized ones [1,6,9,12]. All these tools mainly differ in their modeling and properties specification formalisms. Uppaal-smc [4] considers *Networks of Priced Timed Automata*, which are high-level representations of D/CTMCs for system modeling, and weighted MTL for properties specification. Prism [8] treats in addition *Markov Decision Processes* and *Probabilistic Timed Automata* for modeling, and *Probabilistic Computation Tree*, *Continuous Stochastic Logic* (CSL), and LTL for specification. Plasma Lab [6] is a modular statistical model checker that allows to use external simulators and checkers. Its default configuration supports DTMCs specified in a Prism dialect and bounded LTL. Ymer [12] is one of the rare tools to implement SMC (Hypothesis testing) for GSMPs

The research leading to these results has received funding from the EU's H2020 programme under grant agreements no. 700665 (CITADEL), 7300080 (ESROCOS).
* Institute of Engineering Univ. Grenoble Alpes.

S. K. Lahiri and C. Wang (Eds.): ATVA 2018, LNCS 11138, pp. 536–542, 2018.
https://doi.org/10.1007/978-3-030-01090-4_33

and CSL, however it is no more maintained. Finally, COSMOS [1] relies on *Generalized Stochastic Petri Nets* as input models and *Hybrid Automata Stochastic Logic*, a more expressive formalism, for properties specifications.

In this paper, we present the newest release of $\mathcal{S}$BIP, a statistical model checker that enriches the existing BIP tool-set [2] with statistical analyses. BIP provides a general framework to support design activities ranging from specification and validation to implementation and deployment in a rigorous way. To implement this vision, a rich tool-set was built for modeling, languages embedding, functional validation, models transformation and distributed code generation.

In its previous version [9], $\mathcal{S}$BIP was limited to the analysis of DTMCs with respect to bounded LTL properties. In this release, it was redesigned and extended to support **GSMPs, CTMCs, MTL, parametric exploration of LTL and MTL properties and analysis of rare events**. The tool has also benefited from a major revision of its workflows and GUI. It now provides an Integrated Development Environment (IDE) where one can edit, compile, simulate models, and perform analyses. Additionally, $\mathcal{S}$BIP is now organized around well-structured projects that enclose models, properties and traces. It also includes support for graphical visualization of analysis results.

2 $\mathcal{S}$BIP Design and Functionalities

$\mathcal{S}$BIP is fully developed in Java and runs on GNU/Linux. It is freely available at http://www-verimag.imag.fr/Statistical-Model-Checking.html. The tool is distributed with a large set of case studies and a detailed documentation (e.g., user manual, installation details, video tutorials). For the sake of simplicity, we also provide a virtual machine with a pre-installed version of the tool.

This new release was designed in a modular fashion to allow more flexibility and extensibility. As depicted in Fig. 1, $\mathcal{S}$BIP consists of three generic functional modules: *Stochastic Simulation Engine*, *Monitoring*, and Statistical Analyses that currently include *Hypothesis Testing (HT)*, *Probability Estimation (PE)*, *Parametric Exploration (PX)* and *Importance Splitting (IP)* for rare events analysis. All these modules are fully independent and interact through well-defined Java interfaces. The latter also define a clean and easy way to extend the tool with further modules (simulators, monitors and analyzers). In practice, statistical analysis algorithms trigger the stochastic simulation engine to produce a new execution trace which is monitored against an input property to produce a local verdict. Depending on the used analysis method, several iterations are generally required, to produce the final verdict. The proposed design allows to perform different analyses in separate workflows, namely simple simulation, standard SMC analyses, parametric SMC exploration and analysis of rare events. These workflows rely on common features such as models and properties edition, compilation and generated traces inspection.

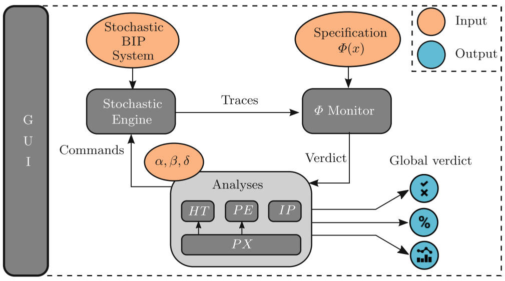

Fig. 1. $\mathcal{S}$BIP architecture

Stochastic Simulation Engine. Currently, $\mathcal{S}$BIP allows to use two different stochastic simulators, namely, for classical stochastic BIP [9] that enables to model discrete-time systems (DTMCs) and for the newly implemented Stochastic Real-Time BIP [10] for continuous-time systems with arbitrary distributions (GSMPs and CTMCs)[1]. The former produces untimed traces needed to verify bounded LTL properties (and to guarantee backward compatibility), whereas the latter generates timed traces necessary to verify MTL properties. We implemented simulators to produce traces in different modes, i.e., symbol-wise, piece-wise and trace-wise. We use the first mode for online monitoring and to be able to interrupt simulations as soon as a verdict is obtained. The second is primordial for rare events analysis and allows to generate traces as a concatenation of trace-fragments. Finally, we use the third mode for offline monitoring.

Monitor. The new release of the tool implements monitoring capabilities for MTL and bounded LTL formulas. Our monitoring algorithms are inspired from the rewrite-based procedures introduced in [3,11]. Given a formula and a trace, the monitor alternates rewriting and simplification phases. Rewriting consumes a symbol of the trace and partially evaluates the formula by unfolding temporal operators and evaluating atomic propositions to their truth value. Simplification applies Boolean reduction rules to the formula in order to conclude or to simplify it. The implemented MTL/LTL grammars and monitors allow for expressing properties with nested operators and having parameters, i.e., variables used to represent a range of properties in a compact way.

Statistical Analyses. In addition to classical SMC algorithms, i.e., *HT* [12] and *PE* [5], we propose in this release two additional analyses (exploitable via independent workflows) for the exploration of properties parameters, *Parametric Exploration (PX)*, and for rare events analysis, *Importance Splitting (IP)* [7]. To recall, *HT* allows to answer qualitative queries, i.e., given a stochastic system S and a property ϕ, it enables to assess whether the probability for S to satisfy

[1] SRT-BIP sources are available at https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/bip/compiler/tree/stochastic-real-time

ϕ is greater or equal to a given threshold θ. *PE* addresses quantitative queries, that is to compute a probability estimate p for S to satisfy ϕ.
Parametric Exploration (PX) is an automated way to perform statistical model checking on a family of properties, in a batch mode. A family of properties is specified in a compact way as a parametric property $\phi(x)$, where x is an integer parameter ranging over a finite instantiation domain Π. Similarly to Prism, our implemented algorithm returns a set of SMC verdicts corresponding to the verification of the parametric property instances $\phi(v_x)$ with respect to $v_x \in \Pi$. This can be very useful when exploring unknown system parameters such as, buffers sizes guaranteeing no overflow, or the amount of consumed energy. It automates the exploration for large parameters domains as opposed to tedious and time consuming manual procedures. This exploration differs from Uppaal-smc parametric SMC which explores the parameters of the input model.
Importance Splitting (IP) overcomes the problem of estimating the probability $P(S \models \phi)$ of a system S to satisfy a property ϕ representing a rare event. This is done by considering a set of intermediate levels l_i that corresponds to less rare properties ϕ_i, s.t., $\phi_n \Rightarrow \phi_{n-1} \Rightarrow \ldots \Rightarrow \phi_1$, where $\phi_n = \phi$. $P(S \models \phi)$ is therefore computed as the product of the conditional probabilities to reach l_i from l_{i-1}, i.e., $\Pi_{i=1}^{n} P(S \models \phi_i \mid S \models \phi_{i-1})$. In our implementation, the intermediate levels l_i and associated ϕ_i are defined via a score function given as input. To evaluate a system trace with respect to ϕ, we implemented a procedure that tells the level reached by the trace, i.e., the intermediate property it satisfies. Our algorithm is similar to the analysis procedure proposed in Plasma Lab. It iterates over levels, and for each one, it simulates m trace prefixes among which m_s reach the next level and m_f do not. The conditional probability to reach the next level is thus estimated as the ratio $^{m_s}/_{m}$. In the next iteration, the simulation of successful prefixes is resumed, while the rest (m_f) are replaced by successful ones sampled uniformly. We note that *IP* is currently limited to the analysis of DTMCs.

3 Case Studies

In this section, we briefly present experiments performed using $\mathcal{S}$BIP [2]. Different case studies covering various application domains were considered to validate the new release of the tool. We implemented models for communication protocols, namely Firewire, Bluetooth, and the Precision Time Protocol (PTP), for a vehicle gear controller, a Pacemaker and a mutual exclusion scenario. All the experiments were performed on a Dell Latitude 5480 with an i7-7820HQ processor and 32 GB of RAM, running Ubuntu 16.04.

On these models, we tackled different types of requirements. For the Firewire case study, we focused on analyzing its leader election protocol in different topologies (2, 3 and 5 nodes) with respect to convergence time, by considering the impact of contention ($\phi_{1,2,3}$) and regarding the impact of a node position on its probability to become the leader (ϕ_4). In this study, except ϕ_3 performed

[2] See details in http://www-verimag.imag.fr/TR/TR-2018-5.pdf

using *PE*, the other properties were performed using *PX*. We also built a parametric model of the Bluetooth device discovery mechanism with one sender and one receiver that can be either in an active (v1) or a sniff mode (v2). For this model, we were interested in studying the energy consumption of the receiver in both modes (ϕ_6) in addition to the convergence time (ϕ_5). The PTP protocol was subject to the analysis of the maximal drift between the master and the slave clocks (ϕ_7).

For the gearbox system, we investigated the minimum and maximum time required to complete a gear change (ϕ_8). We also verified requirements regarding the time relationships between atrial and ventricular events in the pacemaker model ($\phi_{9,10}$). Analyses of the Bluetooth, PTP and the gearbox models were performed using *PX*, while we used *PE* for the Pacemaker. We also considered a model of three concurrent processes arbitrarily requesting access to a shared resource. In this case study, the goal was to estimate the probability that each process is able to access the resource 10 times within 30 system steps (rare property ϕ_{11}). Using our *IP* implementation, we obtained 2.35×10^{-7} in less than 13 s, while it was not possible to observe the rare event using *PE* upon 3 min of execution.

Table 1. Summary of performance

Case study	Model	ϕ	Analysis	#smc loops	avg smc time
Firewire(2)	CTMC	ϕ_1	PX	11	1 m 21 s
		ϕ_2	PX	9	1 m 59 s
		ϕ_3	PE	–	2 m 28 s
		ϕ_4	PX	2	3 m 27 s
Firewire(3)	CTMC	ϕ_1	PX	17	1 m 53 s
		ϕ_2	PX	11	3 m 34 s
		ϕ_3	PE	–	3 m 38 s
		ϕ_4	PX	3	4 m 43 s
Firewire(5)	CTMC	ϕ_1	PX	18	3 m 54 s
		ϕ_2	PX	17	12 m 36 s
		ϕ_3	PE	–	7 m 23 s
		ϕ_4	PX	5	10 m 16 s
Bluetooth v1	CTMC	ϕ_5	PX	9	2 m 27 s
		ϕ_6	PX	16	3 m 11 s
Bluetooth v2	CTMC	ϕ_5	PX	11	3 m 0 s
		ϕ_6	PX	14	13 m 05 s
PTP	GSMP	ϕ_7	PX	15	8 m 42 s
Gear Control	CTMC	ϕ_8	PX	11	54 s
Pacemaker	CTMC	ϕ_9	PE	–	1 h 28 m
		ϕ_{10}	PE	–	1h 30m
Mutual Exclusion	DTMC	ϕ_{11}	IP	–	13 s
			PE	–	3 m 37 s

In addition to these experiments summarized in Table 1, we report in the last two columns some performance measures of the tool, namely, the number of SMC loops performed for parametric exploration, and the average SMC time for a single loop. We observed that depending on the model size and the property complexity, the time varies from some seconds to a dozen of minutes, except for the pacemaker model where it took more than an hour. In this particular case, *PE* required 4883 long execution traces, representing approximately 8 min of real system execution.

4 Discussion

Most SMC tools [1,4,6,8,12] use dedicated abstract models as input for verification. In contrast, $\mathcal{S}$BIP uses BIP, a full-fledged expressive component-based framework developed to support system design from specification to analysis and implementation. It allows for incrementally building complex systems from elementary components and offers real-time capabilities, in addition to high-level coordination and synchronization primitives e.g. multi-party interactions and priorities. Furthermore, it enables including external C++ code, e.g. for modeling complex data structures and integrating legacy code.

We briefly discuss $\mathcal{S}$BIP capabilities with respect to major SMC tools. Regarding the analyses, $\mathcal{S}$BIP implements the HT and PE algorithms similarly to Uppaal-smc [4], Prism [8] and Plasma Lab [6]. Besides, only Prism offers a parametric functionality similar to PX. Furthermore, to the best of our knowledge only Plasma Lab and COSMOS [1] support rare events analysis. The former is the only one implementing IP as in our tool, while the latter rather relies on importance sampling. Our underlying modeling formalism allows for expressing arbitrary probability distributions over time. It offers built-in standard distributions, e.g. Normal, and a simple mechanism for specifying custom distributions. In contrast, Prism is restricted to uniform and exponential distributions, whereas in Uppaal-smc one need to define such distributions manually by using a subset of the C language. The expressiveness of BIP together with the reliance on concrete executions result in lower runtime performance compared to Uppaal-smc and Prism. Comparatively, the authors of Plasma Lab chose to focus on modularity at the expense of performance. In the future, we plan to optimize our simulation engine to improve the overall performance.

References

1. Ballarini, P., Barbot, B., Duflot, M., Haddad, S., Pekergin, N.: Hasl: a new approach for performance evaluation and model checking from concepts to experimentation. Perform. Eval. **90**, 53–77 (2015)
2. Basu, A., et al.: Rigorous component-based system design using the BIP framework. IEEE Softw. **28**(3), 41–48 (2011)
3. Bulychev, P.E., David, A., Larsen, K.G., Legay, A., Li, G., Poulsen, D.B.: Rewrite-based statistical model checking of WMTL. RV, **7687**, 260–275 (2012)
4. David, A., Larsen, K.G., Legay, A., Mikuăionis, M., Poulsen, D.B.: Uppaal SMC tutorial. STTT **17**(4), 397–415 (2015)
5. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: Steffen, B., Levi, G. (eds.) VMCAI 2004. LNCS, vol. 2937, pp. 73–84. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24622-0_8
6. Jegourel, C., Legay, A., Sedwards, S.: A platform for high performance statistical model checking – PLASMA. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 498–503. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_37
7. Jegourel, C., Legay, A., Sedwards, S.: Importance splitting for statistical model checking rare properties. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 576–591. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_38
8. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47

9. Nouri, A., Bensalem, S., Bozga, M., Delahaye, B., Jegourel, C., Legay, A.: Statistical model checking QoS properties of systems with SBIP. In: STTT 2015, vol. 17, pp. 171–185
10. Nouri, A., Mediouni, B.L., Bozga, M., Combaz, J., Legay, A., Bensalem, S.: Performance evaluation of stochastic real-time systems with the SBIP framework. Technical Report TR-2017-6, Verimag Research Report (2017)
11. Roşu, G., Havelund, K.: Rewriting-based techniques for runtime verification. Autom. Softw. Eng. **12**(2), 151–197 (2005)
12. Younes, H.L.S.: Verification and Planning for Stochastic Processes with Asynchronous Events. Ph.D. thesis, Carnegie Mellon (2005)