Effective Stochastic Automata Model Checking by Interval Abstraction* (extended abstract)

Pedro R. D'Argenio
Universidad Nacional de Córdoba and CONICET, Córdoba, Argentina

Arnd Hartmanns
University of Twente, Enschede, The Netherlands

Annabell Petri
University of Twente, Enschede, The Netherlands

In dependable systems like satellites and deep-space space missions, faults happen randomly over time, with mitigation schemes based on redundancy and the use of space-hardened components attempting to reduce the number of faults and in particular prevent them from escalating into overall system failure. At the same time, these systems are subject to performance requirements related to their actual (real-time) tasks and communication with ground stations, with random scheduling effects, message loss probabilities, and signal transmission and propagation delays together determining the key performance properties such as throughput or response times. Two widely-used formalisms are continuous-time Markov chains (CTMCs) and generalized stochastic Petri nets (GSPNs) [13, 22]. In both, all delays must be exponentially distributed. The resulting memoryless nature of these models admits scalable analytical analysis methods such as probabilistic model checking [2, 3]. In reality, however, many inter-event times are not exponentially distributed, and hard to approximate by CTMCs: time to failure is more realistically modelled by a Weibull distribution in most cases, and communication opportunities typically arise at known fixed intervals depending on e.g. the satellite's orbit. We thus need *non-Markovian* formalisms that directly incorporate general continuous probability distributions, such as stochastic automata (SA) [11] or Petri nets with general transitions (e.g. HPnGs [15]). In this paper, we use SA, because they are conceptually simple yet highly expressive and their compositionality makes them attractive for modelling complex component-based systems. Additionally, transformations from reliability block diagrams and dynamic fault trees, which both have been used to model satellite systems, to SA have been developed recently.

SA extend labelled transition systems with stochastic *timers* (that may be *reset* to values sampled from continuous probability distributions to then decrease over time and *expire* when reaching value zero) and edges that become enabled when all timers in their *guard set* have expired. Due to their non-Markovian nature combined with the ability to include nondeterministic choices, the analysis of SA is hard. Two SA model checking algorithms were proposed by Bryans et al. in 2003 [6]. They use a timer-based variant of SA with severe restrictions (timers may only be used out of locations in which they have just been reset, and all distributions must be non-negative and bounded). The first algorithm uses a "region tree" and requires solving nested integrals over the probability density functions of each expired timer, which becomes infeasible as the tree grows in depth. The second algorithm avoids the nested integrals by discretising the distributions. No tool implements either algorithm today. The only currently available tool with dedicated SA support, FIG [7], employs statistical model checking [1, 21] (i.e. Monte Carlo simulation) and is thus restricted to (weakly) deterministic SA [12]. Analytical approaches for

*Authors are sorted alphabetically. This work was supported by the EU's Horizon 2020 research and innovation programme under MSCA grant agreement 101008233 (MISSION), the Interreg North Sea project STORM_SAFE, NWO VIDI grant VI.Vidi.223.110 (TruSTy), and SeCyT-UNC grant 33620230100384CB (MECANO).

Submitted to:
MISSION Workshop 2026

more general models, such as HPnGs or stochastic timed automata (STA) [5, 17], do not scale for typical SA with many discrete locations and several timers possibly reset repeatedly on loops.

We recently developed [10], and summarise here, the first dedicated model checking approach for SA that is effective (i.e. that works for typical and nontrivial SA models) and implemented in an available tool. It is based on *interval abstraction* [14]: we replace each timer reset, i.e. each sampling from a continuous distribution μ, by a discrete distribution over a finite set of intervals that partition μ's support. The choice of concrete value from the selected interval is nondeterministic, and we propagate the intervals forward symbolically through the "big time steps" semantics of the SA. We thereby turn the SA into a Markov decision process (MDP) [4, 20] that overapproximates the SA's semantics. As a result, the maximum (minimum) probability of eventually reaching a set of MDP states corresponding to a goal location in the SA—which we compute by standard value iteration [19]—is an upper (lower) bound on the corresponding probability for the SA.

We specified our approach formally and showed the overapproximation property [10, Sec. 3]. The construction of an MDP $\mathsf{ia}(\mathscr{M})$ for a given SA $\mathscr{M}$ and interval abstraction $\mathfrak{p}$ follows two rules: First, the discrete step rule adds outgoing transitions to an MDP state whenever an edge in the corresponding SA location would be enabled. The transition leads into a discrete distribution $\pi^b_{R,\ell'}$ over target states, with one target state for every possible combination of intervals partitioning the supports of the timers that are reset. That is, the $\pi^b_{R,\ell'}$ randomly picks an interval for every timer that is reset. Second, the time step rule implements the big time steps semantics. It creates σ-labeled transitions, which represent the symbolic passage of some time until an edge in the SA becomes enabled. For every SA edge, one σ-labeled transition is created, resulting in a nondeterministic choice of which timers within overlapping intervals of time expire first. A condition for generating a σ transition is thus that no other guard set must definitely expire before. Then, a new state is created where the interval bounds for each timer within the guard set are set as expired, while the remaining timers are updated to reflect the possible valuation after the minimal and maximal delays. We formalized the operational semantics of both the time-step rule and discrete step rule [10, Sec. 3.1]. Furthermore, we provided a proof sketch, based on results from [16, Thm. 4.22], showing that our interval abstraction MDP does indeed underapproximate minimal reachability probabilities and overapproximate maximal reachability probabilities:

$$p^{\mathsf{ia}(\mathscr{M})}_{min} \leq p^{\mathscr{M}}_{min} \quad \text{and} \quad p^{\mathscr{M}}_{max} \leq p^{\mathsf{ia}(\mathscr{M})}_{max}.$$

Our new SA model checking approach is implemented in a prototype tool written in Rust and can be found at doi:10.5281/zenodo.19829349. The tool pipeline is shown in Fig. 1. A probability mass p_I must be specified as input alongside the SA model and the property to check, which queries for the probability of the set of paths satisfying a $\neg \mathit{avoid}\, \mathsf{U}\, \mathscr{G}$ until formula. For each distribution associated to a timer, the inverse cumulative distribution function (CDF) is then used to find intervals of probability mass p_I that partition the distribution's support. If $1/p_I$ is not an integer, then one (the last, with highest values) interval may have probability $< p_I$ to cover the remainder of μ's support. The tool currently supports timers uniform, exponential, Erlang, and Weibull distributions. For the Erlang distribution, we currently use the `rustst at` library's implementation of the inverse CDF for the gamma distribution. We also extended the MODEST [5, 18] and JANI [8] languages with support for timers and thus SA, providing a new user-friendly high-level modeling language for SA and a means to conveniently exchange SA models between tools [10, Sec. 4].

Using the prototype implementation, we evaluated the performance and scalability of the transformation approach on example SA from and inspired by the literature [10, Sec. 4]. These examples cover all common features of SA that we support: discrete nondeterministic choices, discrete probability distributions over target locations, immediate and timer-guarded edges, including guards with multiple timers

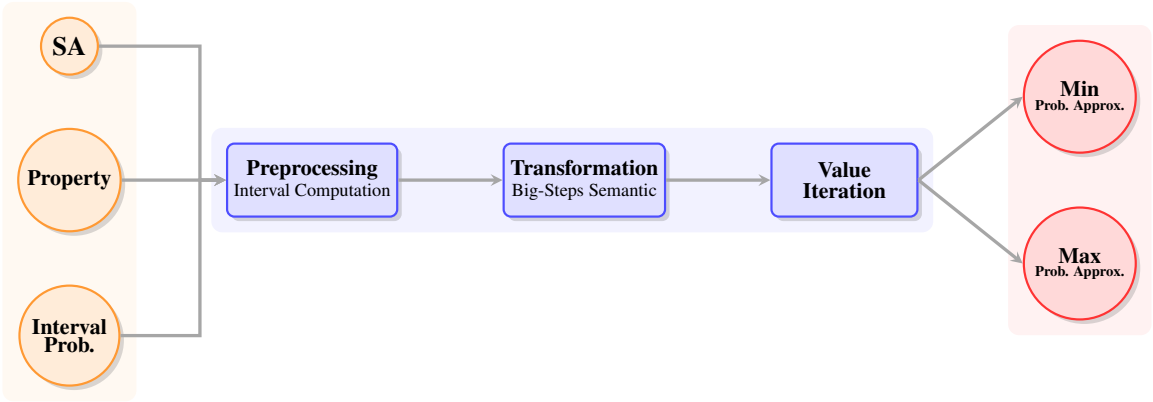

Figure 1: Tool pipeline of the prototype implementation in Rust.

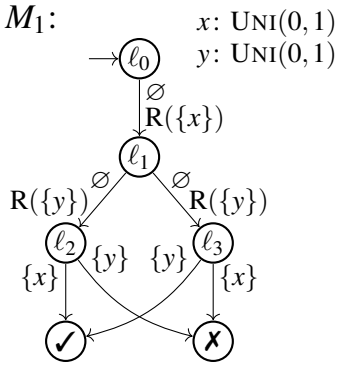

Figure 2: SA M_1

Figure 3: Results for M_1 from [9] with different per-interval probabilities.

p_I	0.1		0.01		0.005	
Property	p_{min}	p_{max}	p_{min}	p_{max}	p_{min}	p_{max}
Result	0.200	0.800	0.245	0.755	0.247	0.752
MDP states	957		82657		325565	
Transformation	2.4 ms		1.4 s		12.3s	
VI	0.9 ms		0.5 s		4.6s	
Runtime total	3.3 ms		1.9 s		17s	

and races between edges, and timers following several different distributions. We ran the tool on SA M_1, M_2, and M_5 from [9] and obtained good approximations of the actual results despite the overapproximation from the abstraction within relatively short runtimes, outperforming the results obtained by strategy sampling in [9].

The experiments confirmed that the accuracy of the results depends on the specified probability p_I as expected: for decreasing p_I, p_{max} decreases and p_{min} increases. The results obtained for M_1, shown in Figure Fig. 2, can be seen in Table Fig. 3 and demonstrate that the tool gives a close approximation of the actual probabilities of $p_{max} = 0.75$ and $p_{min} = 0.25$ within a reasonable runtime. We also evaluated our approach on SA with a larger state space using SA models of non-Markovian queueing systems with a buffer size of 100. Despite the state space size of millions of MDP states after transformation, the runtime for the entire tool pipeline remains below 5 minutes. Finally, we compared our tool against the approach for stochastic timed automata (STA) [5,17] implemented in the MODEST TOOLSET using a file server model introduced in [17]. We find that for comparable runtimes between both tools, our prototype implementation yields more accurate results.

In summary, we find that, despite its prototypical nature, our tool already works well for reasonably-sized SA even when using many intervals to obtain results with good accuracy. With some additional optimisations, it shows strong potential to significantly outperform existing techniques that handle superclasses of SA, benefiting from being an SA-specific technique.

References

[1] Gul Agha & Karl Palmskog (2018): *A Survey of Statistical Model Checking*. *ACM Trans. Model. Comput. Simul.* 28(1), pp. 6:1–6:39, doi:10.1145/3158668.

[2] Christel Baier, Luca de Alfaro, Vojtech Forejt & Marta Kwiatkowska (2018): *Model Checking Probabilistic Systems*. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith & Roderick Bloem, editors: *Handbook of Model Checking*, Springer, pp. 963–999, doi:10.1007/978-3-319-10575-8_28.

[3] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns & Joost-Pieter Katoen (2003): *Model-Checking Algorithms for Continuous-Time Markov Chains*. *IEEE Trans. Software Eng.* 29(6), pp. 524–541, doi:10.1109/TSE.2003.1205180.

[4] Richard Bellman (1957): *A Markovian decision process*. *Journal of Mathematics and Mechanics* 6(5), pp. 679–684.

[5] Henrik C. Bohnenkamp, Pedro R. D'Argenio, Holger Hermanns & Joost-Pieter Katoen (2006): *MoDeST: A Compositional Modeling Formalism for Hard and Softly Timed Systems*. *IEEE Trans. Software Eng.* 32(10), pp. 812–830, doi:10.1109/TSE.2006.104.

[6] Jeremy W. Bryans, Howard Bowman & John Derrick (2003): *Model checking stochastic automata*. *ACM Trans. Comput. Log.* 4(4), pp. 452–492, doi:10.1145/937555.937558.

[7] Carlos E. Budde (2022): *FIG: the Finite Improbability Generator v1.3*. *SIGMETRICS Perform. Evaluation Rev.* 49(4), pp. 59–64, doi:10.1145/3543146.3543160.

[8] Carlos E. Budde, Christian Dehnert, Ernst Moritz Hahn, Arnd Hartmanns, Sebastian Junges & Andrea Turrini (2017): *JANI: Quantitative Model and Tool Interaction*. In Axel Legay & Tiziana Margaria, editors: *23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2017)*, *Lecture Notes in Computer Science* 10206, pp. 151–168, doi:10.1007/978-3-662-54580-5_9.

[9] Pedro R. D'Argenio, Marcus Gerhold, Arnd Hartmanns & Sean Sedwards (2018): *A Hierarchy of Scheduler Classes for Stochastic Automata*. In Christel Baier & Ugo Dal Lago, editors: *21st International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2018)*, *Lecture Notes in Computer Science* 10803, Springer, pp. 384–402, doi:10.1007/978-3-319-89366-2_21.

[10] Pedro R. D'Argenio, Arnd Hartmanns & Annabell Petri (2026): *Effective Stochastic Automata Model Checking by Interval Abstraction*. Available online at modestchecker.net/Publications/PDF/DHP26.pdf.

[11] Pedro R. D'Argenio & Joost-Pieter Katoen (2005): *A theory of stochastic systems part I: Stochastic automata*. *Inf. Comput.* 203(1), pp. 1–38, doi:10.1016/J.IC.2005.07.001.

[12] Pedro R. D'Argenio & Raúl E. Monti (2018): *Input/Output Stochastic Automata with Urgency: Confluence and Weak Determinism*. In Bernd Fischer & Tarmo Uustalu, editors: *15th International Colloquium on Theoretical Aspects of Computing (ICTAC 2018)*, *Lecture Notes in Computer Science* 11187, Springer, pp. 132–152, doi:10.1007/978-3-030-02508-3_8.

[13] Christian Eisentraut, Holger Hermanns, Joost-Pieter Katoen & Lijun Zhang (2013): *A Semantics for Every GSPN*. In José Manuel Colom & Jörg Desel, editors: *34th International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2013)*, *Lecture Notes in Computer Science* 7927, Springer, pp. 90–109, doi:10.1007/978-3-642-38697-8_6.

[14] Martin Fränzle, Ernst Moritz Hahn, Holger Hermanns, Nicolás Wolovick & Lijun Zhang (2011): *Measurability and safety verification for stochastic hybrid systems*. In Marco Caccamo, Emilio Frazzoli & Radu Grosu, editors: *14th ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2011)*, ACM, pp. 43–52, doi:10.1145/1967701.1967710.

[15] Hamed Ghasemieh, Anne Remke & Boudewijn R. Haverkort (2014): *Hybrid Petri nets with multiple stochastic transition firings*. In Moshe Haviv, William J. Knottenbelt, Lorenzo Maggi & Daniele Miorandi, editors: *8th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2014)*, ICST.

[16] Ernst Moritz Hahn (2013): *Model checking stochastic hybrid systems*. Ph.D. thesis, Saarland University, doi:10.22028/D291-26494. Available at `http://scidok.sulb.uni-saarland.de/volltexte/2013/5259/`.

[17] Ernst Moritz Hahn, Arnd Hartmanns & Holger Hermanns (2014): *Reachability and Reward Checking for Stochastic Timed Automata*. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.* 70, doi:10.14279/TUJ.ECEASST.70.968.

[18] Ernst Moritz Hahn, Arnd Hartmanns, Holger Hermanns & Joost-Pieter Katoen (2013): *A compositional modelling and analysis framework for stochastic hybrid systems*. *Formal Methods Syst. Des.* 43(2), pp. 191–232, doi:10.1007/S10703-012-0167-Z.

[19] Arnd Hartmanns, Sebastian Junges, Tim Quatmann & Maximilian Weininger (2023): *A Practitioner's Guide to MDP Model Checking Algorithms*. In Sriram Sankaranarayanan & Natasha Sharygina, editors: *29th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2023)*, *Lecture Notes in Computer Science* 13993, Springer, pp. 469–488, doi:10.1007/978-3-031-30823-9_-24.

[20] Ronald A. Howard (1960): *Dynamic Programming and Markov Processes*. MIT Press.

[21] Axel Legay, Anna Lukina, Louis-Marie Traonouez, Junxing Yang, Scott A. Smolka & Radu Grosu (2019): *Statistical Model Checking*. In Bernhard Steffen & Gerhard J. Woeginger, editors: *Computing and Software Science – State of the Art and Perspectives*, *Lecture Notes in Computer Science* 10000, Springer, pp. 478–504, doi:10.1007/978-3-319-91908-9_23.

[22] Marco Ajmone Marsan, Gianni Conte & Gianfranco Balbo (1984): *A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems*. *ACM Trans. Comput. Syst.* 2(2), pp. 93–122, doi:10.1145/190.191.