An Overview of Modest Models and Tools
for Real Stochastic Timed Systems

Arnd Hartmanns*

University of Twente
Enschede, The Netherlands
`a.hartmanns@utwente.nl`

We depend on the safe, reliable, and timely operation of cyber-physical systems ranging from smart grids to avionics components. Many of them involve time-dependent behaviours and are subject to randomness. Modelling languages and verification tools thus need to support these quantitative aspects. In my invited presentation at MARS 2022, I gave an introduction to quantitative verification using the Modest modelling language and the Modest Toolset, and highlighted three recent case studies with increasing demands on model expressiveness and tool capabilities: A case of power supply noise in a network-on-chip modelled as a Markov chain; a case of message routing in satellite constellations that uses Markov decision processes with distributed information; and a case of optimising an attack on Bitcoin via Markov automata model checking. This paper summarises the presentation.

1 Introduction

Cyber-physical systems consist of discrete (usually digital, often implemented in software) controllers interacting with a continuous physical environment. Control is often networked, sometimes wirelessly. Many cyber-physical systems are safety- or performance-critical, or economically vital. We thus need to ensure that they operate as desired, which includes dependability requirements such as reliability assurances, availability levels, or response time guarantees. Reliability and availability are stochastic timed properties: the probability of avoiding unsafe behaviour within a certain time horizon, and the expected fraction of time that the system is ready to provide service, respectively. The critical systems themselves are also typically subject to randomisation, for example due to random message loss in wireless communication or due to employing randomised algorithms, and they are timed systems dealing with e.g. transmission delays and timeouts or faults occurring unpredictably over time. Thus, to assure their dependability by way of modelling and verification (ideally at design-time), we need stochastic timed formalisms and modelling languages supported by tools able to check stochastic timed properties.

In this overview paper accompanying my invited presentation at the 5th Workshop on Models for Formal Analysis of Real Systems (MARS 2022), I outline two such modelling languages, Modest and JANI, and one such set of tools, the Modest Toolset (in Section 2). I then briefly summarise how Modest and the Modest Toolset have been used to study power supply noise in a two-by-two network-on-chip system by way of a discrete-time Markov chain (DTMC) model and probabilistic model checking with the mcsta tool (in Section 3); to find routes through sparse constellations of nanosatellites using an abstract Markov decision process (MDP) [7, 35] model analysed with a statistical model checking approach that employs scheduler sampling under distributed information as implemented in the modes tool (in Section 4); and to optimise an attack on the Bitcoin cryptocurrency system via a Markov automata (MA) [22] model that

---

Figure 1: The family tree of automata-based quantitative formalisms

The key for Figure 1:

Key:	
SHA	stochastic hybrid automata [24]
PHA	probabilistic hybrid automata [46]
STA	stochastic timed automata [8]
HA	hybrid automata [2]
PTA	probabilistic timed automata [40]
MA	Markov automata
TA	timed automata [3]
MDP	Markov decision processes
CTMDP	continuous-time Markov decision processes
LTS	labelled transition systems
DTMC	discrete-time Markov chains
CTMC	continuous-time Markov chains

permits mcsta to synthesise the strategy that minimises the expected time to success or maximises the probability of success within a certain time bound (in Section 5).

2   Modest Languages and Tools

A well-defined semantics in terms of some mathematically well-understood object is a cornerstone of formal models. For quantitative models, we use automata-based formalisms—that represent the evolution of a system from state to state via (randomised) transitions—building on labelled transition systems (LTS, or Kripke structures) and discrete- and continuous-time Markov chains (DTMC and CTMC, respectively) [6]. By combining these basic mathematical formalisms in various ways, and extending them with features such as real-time clocks and continuous variables evolving according to differential equations, we obtain further formalisms as depicted in Figure 1. Since writing real-life models as, say, large Markov chains would be cumbersome, we specify them using a higher-level modelling language that offers at least discrete variables with standard arithmetic and Boolean operators plus a notion of parallel composition for the natural specification of distributed and component- or actor-based systems.

The Modest Language.   One such language is Modest, originally the <u>mo</u>delling and <u>de</u>scription language for <u>s</u>tochastic <u>t</u>imed systems [8]. Its formal semantics was first defined in terms of STA and later extended to SHA [29]. Modest is a textual modelling language; its syntax is designed to be similar to widely used programming languages like C or Java to lower the barrier of entry for domain experts. At the same time, it is a process algebra in spirit, based on standard operators such as sequential and parallel composition, allowing the definition of and recursive calls to processes, and emphasising compositionality. In fact, Modest consists of two largely orthogonal languages: one to define *behaviour*, which is the one based on process-algebraic ideas, and one to manipulate *data* such as the values of discrete variables. The latter provides arrays, recursive datatypes (e.g. allowing the definition of a linked list type via pairs of a head containing a data item and a linked list option tail), and mutually recursive functions. These features allow for concise and natural models of complex real-life systems.

The JANI model interchange format.   While Modest is a convenient modelling language for end-users, the work required to implement code that parses Modest models and transforms the parsed syntax

into its symbolic semantics (a parallel composition of SHA with discrete variables) is nontrivial. The same problem affects many other modelling languages, e.g. Prism's [38], too. To ease tool development and facilitate the exchange of models between different tools, in 2016, the developers of several quantitative verification tools defined the JSON-based JANI [13] format. It is not designed to be human-writable, but rather serve as a model interchange format that is generated by tools from other modelling languages, such as Modest. Today, JANI is supported by the Modest Toolset (see below), Storm [21], Momba [37], and several other tools. All models in the quantitative verification benchmark set (QVBS) [33] are available in both their original formats as well as in JANI. The QVBS served as the foundation for the QComp 2019 [28] and QComp 2020 [14] tool competitions.

The Modest Toolset. To support the creation of Modest models, and to compute the values of properties or check requirements specified as part of models, the Modest Toolset [30] provides a collection of visualisation, model transformation, model checking, and simulation tools. The Modest Toolset has been in development since 2008; it is written in C#, and is available as precompiled binaries for common Linux distributions, macOS, and Windows at modestchecker.net. As input languages, it supports Modest and JANI; its moconv tool can convert between the two and apply various transformations, such as converting a suitable PTA model into its digital clocks [39] MDP. The mosta tool visualises a model's symbolic semantics, helping in learning Modest and in debugging models. The mopy tool converts a model into Python code implementing a first-state-next-state interface [9] that can be used to quickly prototype explicit-state verification algorithms and that is used by the author as part of the programming project of a Master's-level course on quantitative verification at the University of Twente.

The main implementation of probabilistic model checking (PMC) [4, 5] in the Modest Toolset is in mcsta [31]: an explicit-state model checker that provides a unique disk-based approach to mitigate the state space explosion problem [31]. It includes efficient model reductions such as the essential states abstraction [19], and provides state-of-the-art algorithms for model checking MA [15]. The Modest Toolset's statistical model checker modes [12] complements mcsta's capabilities for cases where model checking cannot be applied, such as when facing state space explosion or models with non-Markovian probability distributions like STA. Statistical model checking (SMC) [1] is, in essence, Monte Carlo simulation applied to formal models and properties. A constant-memory technique, it however incurs an explosion in runtime when faced with rare events, and does not directly support nondeterministic models such as MDP. The modes tool addresses these shortcomings by providing rare event simulation [45] via a highly automated implementation of importance splitting [11], and by offering the lightweight scheduler sampling technique [41] for MDP, PTA [17, 34], and (with limitations) stochastic-time models like MA and STA [18]. Other members of the Modest Toolset provide specialised analysis algorithms such as variants of the probabilistic planning algorithm LRTDP [10] for MDP in modysh [36] or an abstraction-based approach to safety verification of SHA in prohver [29].

3 Power Supply Noise in a Network-On-Chip System

As the complexity of distributed many-core systems advances, the network-on-chip (NoC) architecture has become the de-facto standard for on-chip communication. A NoC is typically composed of topologically homogeneous routers operating synchronously in a decentralized manner using a predefined routing protocol. Changes in the supply voltage—*power supply noise* (PSN)—can influence the performance of the transistor devices in a NoC. PSN is created by the simultaneous switching of logic devices, which causes a drop in the effective power supply voltage. PSN is composed of two major components:

resistive noise (related to the current drawn and the resistance of the circuit) and inductive noise (which is proportional to the rate of change of current through the inductance of the power grid).

To study PSN in NoC architectures, we modelled in Modest and analysed with mcsta first a single central router of a NoC [42] and later a two-by-two NoC consisting of four symmetric routers [44] as shown in Figure 2. We focus on the latter in this section. Our goal is to compute the probability for behavioural patterns that are likely to result in resistive resp. inductive noise to occur at least *n* times within *t* clock cycles, starting from an initial state where all buffers are empty. We consider two different data packet (flit) generation patterns: one

Figure 2: Architecture of the 2×2 NoC [44]

where each router receives a flit into its local buffer (e.g. from the one core it is connected to) every other cycle, and one where flits are generated in bursts. We assume the destination of a flit to be one of the other router's local outputs, with the actual router selected uniformly at random for each flit. The routers use a specific round-robin style routing protocol. Thus, with all decisions fixed to be either deterministic (flit generation times and routing choices) or random (flit destinations), and the whole NoC running on a discrete clock, this system can naturally be modelled as a DTMC. The main challenge for model checking with mcsta is to avoid the state space explosion problem.
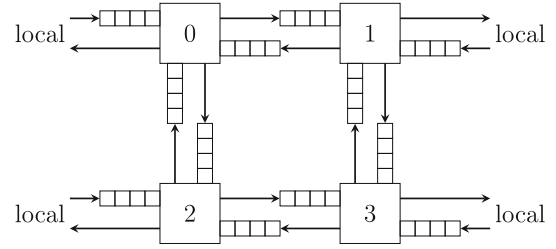
For a first concrete model, which exploited the availability of complex user-defined datatypes in Modest to represent the state of the network's routers and buffers in full detail, we were unable to perform model checking for more than $t = 4$ clock cycles. We then manually applied a series of abstractions to achieve tractability: predicate abstraction to replace the details in the complex datatype's values by only the relevant predicates; a probabilistic choice abstraction that delays random assignments to discrete variables to the point where the assigned value is first tested; and an abstraction of the buffers that includes replacing them by bounded integer variables counting the number of waiting flits only.

The resulting model could be model-checked for up to 30 clock cycles with every-other-cycle flit generation by unfolding the clock cycle counter into the state space, and up to any number of clock cycles by using the unfolding-free modified iteration technique of [27], in essence computing the entire cumulative distribution function (CDF) as shown in Figure 3. This is due to an interesting effect of the different flit generation patterns: With every-other-cycle generation, the buffers slowly fill up with flits to various destinations; the full state space that includes all combinations of buffer occupancies with different flits is too large to handle today. Restricting the state space exploration to a bounded number of clock cycles, where initially few flits are present throughout the system, results in a sequence of manageable state spaces of ever-increasing size. With bursty flit generation, all

Figure 3: CDF for inductive noise events [44]

buffers periodically return to an empty state; the period is small enough for the entire state space to fit into memory, i.e. buffers do not fill up far enough for the number of combinations of buffer states to grow too large, if clock cycles are managed as rewards.
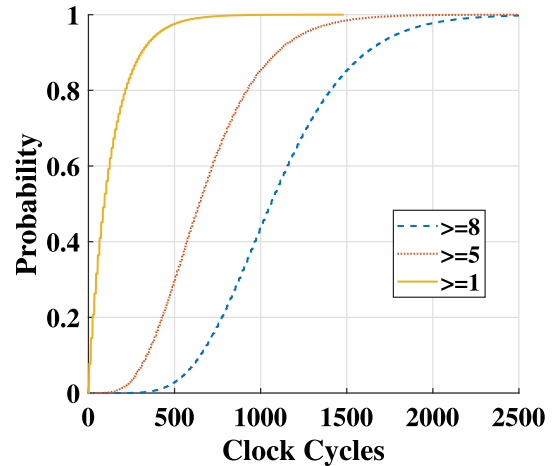
We also applied SMC, which however was limited in the case of every-other-cycle generation by
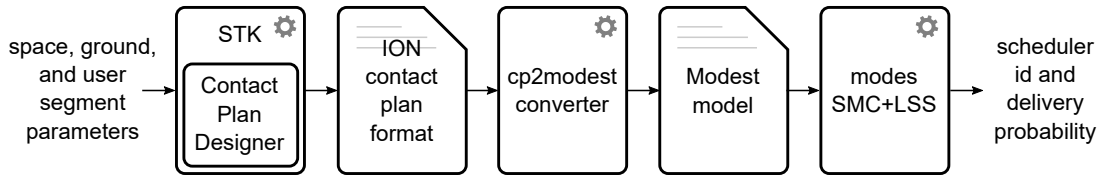
Figure 5: Satellite routing scheduling toolchain for uncertain delay-tolerant networks [16]

noise events being relatively rare, and in the case of bursty generation by not being able to compete in terms of runtime with the modified iteration technique that can compute the probabilities for the entire sequence of values of t up to any upper bound in one go. Similarly, our attempts to use Storm's binary decision diagram-based state space exploration did not provide scalability improvements, possibly due to the model not being as structured as we think it is, or simply due to a bad variable ordering in the model. For further details on this first case study, we refer the interested reader to the original paper that was presented at FMICS 2021 [44].

4 Routing in Satellite Constellations

Satellite networks in low-Earth orbit are increasingly used to collect and distribute information across the globe, including access to the Internet. For real-time applications like Internet access, this requires very large constellations (such as the Starlink constellation being deployed by SpaceX); even if low-cost satellites based on off-the-shelf components that are not space-qualified are used, the entire constellation becomes extremely expensive. A different and more sustainable approach is to relax the real-time constraint and leverage the store-carry-and-forward principle where nodes store received messages for later forwarding to other nodes in the network, once a communication window—a contact—appears. This gives rise to a delay-tolerant network.

In satellite constellations, the orbits are known with sufficient precision to calculate the upcoming contacts over the next few days, giving rise to a *contact plan*. However, message transmissions may fail for various reasons such as unreliable (low-cost) components, contact mispredictions, or interference during the wireless communication. If statistical data is available or the error margins of calculations are known, we can assign a success probability to each contact, giving rise to an uncertain contact plan. We show an abstract representation of such a plan in Figure 4. This plan comprises four satellites (or ground stations) N_1 through N_4 with contacts over five time slots T_1 through T_5. The numbers annotating contacts are the transmission success probabilities.
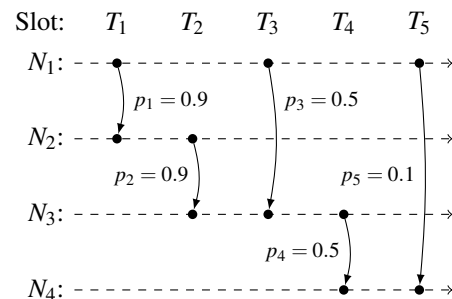

Figure 4: Uncertain contact plan [16]

Now, given the source and destination of a message, and a limit n on the number of message copies present in the network to avoid exhausting the satellites' limited resources, we would like to compute the routing strategy that maximises the probability of message delivery within the time window covered by the contact plan. Due to the combination of randomness (in transmission failures) with nondeterministic decisions to be optimised (which contacts to use to send how many copies) in a discrete-time setting (a sequence of contacts), MDP are the perfect match among the formalisms of Figure 1 to model this problem. The goal is to find an optimal scheduler (i.e. routing strategy) for the MDP.

We have tackled the problem by developing the toolchain outlined in Figure 5 that converts a concrete contact plan (with exact contact timings) into an abstract Modest MDP model. At that point, one could apply PMC via e.g. mcsta to obtain the optimal scheduler. However, PMC works with complete, global information: Consider the contact plan of Figure 4 with $n = 2$. N_1 will definitely send one copy to N_2 in slot T_1, and if successful, N_2 will forward that copy in slot T_2. In slot T_3, the best course of action computed by PMC for satellite N_1 is to send its remaining copy to N_3 if any only if N_3 did not receive the first copy. Thus the model checker "sees" the state of all satellites. Satellites, however, do not have global information about the state of all other satellites in the constellation, making the optimal strategies found by PMC potentially unimplementable. In fact, what we need are distributed schedulers [25]. Unfortunately, the model checking problem under distributed schedulers is undecidable, and even with simplifications such as restricting to memoryless schedulers remains practically intractable [26]. Recently, an approximative model checking-based approach that is specifically tailored to the uncertain delay-tolerant networks case has become available [43], which however still remains limited by state space explosion as n increases.

We instead propose to adapt the lightweight scheduler sampling (LSS) approach to sample distributed schedulers [16]. In LSS, each scheduler is represented by a fixed-size (e.g. 32-bit) integer. Performing an SMC analysis for each of m randomly sampled such integers and keeping the maximum (minimum) estimate provides an underapproximation (overapproximation) for the maximum (minimum) probability achievable with the unknown optimal schedulers. During an SMC analysis for scheduler i, when the simulator needs to decide between k actions in state s, it concatenates the bitstring representations of s and i, applies a hash function mapping this value to a fixed-size integer j, and selects the $((j \bmod k) + 1)$-th action. To perform the same analysis w.r.t. distributed schedulers, all we need to change is the input to the hash function: instead of the bitstring for s, we use that for a projection of s to the variables observable by the currently active component (here: satellite). We also introduce a condition of good-for-distribution models that, when satisfied, ensures that no two components may have a decision at the same time instant, making a global arbiter to break such ties unnecessary. Our Modest models generated for uncertain contact plans are good for distribution by construction.

We implemented LSS for distributed schedulers as described above in modes, and applied this implementation to a small example contact plan as well as a realistic Walker-formation constellation. Our results, presented at NFM 2020 [16], show that LSS is able to find good and implementable routing strategies, and that restricting to distributed schedulers may actually result in better strategies than sampling from all (global-information) schedulers by virtue of restricting the sampling space.

5   Optimally Attacking Bitcoin

The Bitcoin cryptocurrency records its transactions in a blockchain to which blocks are added via the proof-of-work principle: participants need to solve a computationally intensive problem to be able to generate or *mine* a valid block. Generally, the first new valid block mined gets appended to the chain, and a certain number of Bitcoins is awarded to the participant that found the block as a reward. However, as a distributed system spanning the globe via the Internet, Bitcoin has to deal with asynchrony: If multiple participants find new blocks at roughly the same time, there are different alternative forks of the Bitcoin blockchain, and a consensus must be reached on which is the valid one. In Bitcoin, the longest available chain is considered the valid one.

As the computational power used for mining new blocks (the hash rate) changes, the Bitcoin network periodically adjusts the hardness of the problem such that the average time to find a new block (the

```
const real M;                 // fraction of hash rate controlled by malicious mining pool
const int CD;                 // confirmation depth required by victim
const int DB = CD;            // attacker gives up when this far behind

action sln;                   // indicates that the honest pool mined a new block
action rst;                   // indicates that the attacker restarts from the public fork
action cnt;                   // indicates that the attacker continues

int(0..CD+1) m_len;           // length of the secret fork
int(-DB..CD+1) m_diff = 0;    // length of secret fork minus honest fork

process HonestPool()
{
   rate(1/12 * (1 - M)) tau;  // wait 12 / (1 − M) minutes on average
   sln;                       // signal that a new block was found
   HonestPool()               // repeat
}

process TrustAttacker()
{
   do {
   :: rate((1/12) * M) {= m_len = min(CD, m_len + 1), m_diff++ =} // new secret block
   :: sln {= m_diff-- =};                // public fork extended
      alt {                              // strategy choice: restart or continue malicious fork
      :: rst {= m_len = 0, m_diff = 0 =} // can always restart
      :: when(m_diff > -DB) cnt          // can continue if not too far behind
      }
   }
}

par {
:: HonestPool()
:: TrustAttacker()
}
```

Figure 6: Modest model for optimising the trust attack on Bitcoin [32]

confirmation time) is 10 minutes. In practice, the actual confirmation time varies; it was about 12 minutes in 2017 [23]. This time is truly random, and the mining of new blocks can abstractly be modelled by a CTMC in which the transition from a chain with n blocks to one with $n+1$ blocks has rate $\frac{1}{12}$, i.e. the time until the transition is taken is exponentially distributed with that rate.

If a large amount of the hash rate (say M percent) is controlled by one malicious entity, they could feasibly implement various attacks on the Bitcoin network by secretly working on their own fork until it becomes longer than the "public" one, and then broadcasting the secret fork. For example, Bitcoins could be spent twice: once on the public fork in block b_i, and once on the secret fork that branches off from publicly known block b_j that is before b_i in the chain. This behaviour can be integrated into an abstract CTMC model of Bitcoin to e.g. compute the expected time until the attack succeeds for various values of M. We built such a model in Modest and studied similar properties using mcsta and modes [32].

A more interesting and somewhat easier attack, which however does not have the individual benefit of doubly-spent coins but rather attempts to undermine the public trust in Bitcoin, is to simply try to obtain a secret fork that is longer than the official one by a certain margin, and then publish that fork. If done repeatedly, regular users could no longer rely on the persistence of transactions that initially

appeared to have become a part of the valid Bitcoin blockchain. In this attack, every time the public fork is extended, the malicious entity may decide between (a) continuing to work on its current secret fork and (b) restarting its secret fork from the new public block. This is because it is no longer necessary to purge a specific block b_i from the public chain as in the double-spending attack. Due to the presence of a nondeterministic choice to be optimised, this attack can thus no longer be represented in a CTMC model.

The attack on trust in Bitcoin was first analysed by Fehnker and Chaudhary [23] using statistical model checking with UPPAAL SMC [20]. As a consequence of using SMC, they had to run a separate analysis for every possible strategy determining the conditions for when to continue and when to restart, and their results came with a statistical error. We later modelled the same scenario as the Modest MA model shown in Figure 6 and let mcsta synthesise the optimal strategy [32], which it could do in a matter of a few seconds. We found that the optimal strategy is to restart the attack if (i) the public chain is extended when the secret fork is still empty, (ii) the secret fork has one block and the public fork adds a third new block, or (iii) the secret fork has ≥ 2 blocks and becomes three blocks shorter than the public one, and to continue the attack in all other cases. If the malicious entity controls just 20 % of the hash rate, which is not an uncommon situation for the Bitcoin network, then the expected time to success under this strategy is only approximately 2.5 days.

6   Summary

Different case studies have different needs in terms of conceptual modelling power, modelling language features, and analysis tool capabilities. I highlighted three examples that were modelled in the Modest language and analysed using different tools from the Modest Toolset: First, in the case of **power supply noise in a NoC**, the simple formalism of DTMC was sufficient. For the detailed concrete model, however, the Modest language feature of declaring and using one's own complex data types was very helpful. PMC via mcsta was the analysis method of choice, however significant effort was needed to abstract the model until it became tractable for PMC due to the state space explosion problem. Second, for **routing in satellite constellations**, nondeterministic choices needed to be modelled, and optimised over by the analysis tool. Here, MDP fit the problem very well with their ability to model decision-making under uncertainty. We auto-generated Modest models from contact plans computed by domain-specific software. Due to the need to find implementable routing strategies in the distributed-information setting of satellite constellations, we could not use PMC; instead, we adapted the LSS approach to allow SMC to handle both distributed information and nondeterminism. Finally, to **optimally attack Bitcoin**, we showed that MA fit the problem well due to the combination of the stochastic time-to-next-block with the nondeterministic choices between continuing and restarting the secret fork. Using PMC with mcsta again, we were able to compute an optimal strategy with little computational effort.

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] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger & Pei-Hsin Ho (1992): *Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems*. In Robert L. Grossman, Anil Nerode, Anders P. Ravn & Hans Rischel, editors: *Hybrid Systems, Lecture Notes in Computer Science* 736, Springer, pp. 209–229, doi:10.1007/3-540-57318-6_30.

[3] Rajeev Alur & David L. Dill (1994): *A Theory of Timed Automata*. *Theor. Comput. Sci.* 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[4] Christel Baier (2016): *Probabilistic Model Checking*. In Javier Esparza, Orna Grumberg & Salomon Sickert, editors: *Dependable Software Systems Engineering, NATO Science for Peace and Security Series – D: Information and Communication Security* 45, IOS Press, pp. 1–23, doi:10.3233/978-1-61499-627-9-1.

[5] 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.

[6] Christel Baier & Joost-Pieter Katoen (2008): *Principles of model checking*. MIT Press.

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

[8] 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.

[9] Henrik C. Bohnenkamp, Holger Hermanns, Joost-Pieter Katoen & Ric Klaren (2003): *The Modest Modeling Tool and Its Implementation*. In Peter Kemper & William H. Sanders, editors: *13th International Conference on Computer Performance Evaluations, Modelling Techniques and Tools (TOOLS), Lecture Notes in Computer Science* 2794, Springer, pp. 116–133, doi:10.1007/978-3-540-45232-4_8.

[10] Blai Bonet & Hector Geffner (2003): *Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming*. In Enrico Giunchiglia, Nicola Muscettola & Dana S. Nau, editors: *13th International Conference on Automated Planning and Scheduling (ICAPS)*, AAAI, pp. 12–21.

[11] Carlos E. Budde, Pedro R. D'Argenio & Arnd Hartmanns (2019): *Automated compositional importance splitting*. *Sci. Comput. Program.* 174, pp. 90–108, doi:10.1016/j.scico.2019.01.006.

[12] Carlos E. Budde, Pedro R. D'Argenio, Arnd Hartmanns & Sean Sedwards (2020): *An efficient statistical model checker for nondeterminism and rare events*. *Int. J. Softw. Tools Technol. Transf.* 22(6), pp. 759–780, doi:10.1007/s10009-020-00563-2.

[13] 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), Lecture Notes in Computer Science* 10206, pp. 151–168, doi:10.1007/978-3-662-54580-5_9.

[14] Carlos E. Budde, Arnd Hartmanns, Michaela Klauck, Jan Kretínský, David Parker, Tim Quatmann, Andrea Turrini & Zhen Zhang (2020): *On Correctness, Precision, and Performance in Quantitative Verification (QComp 2020 Competition Report)*. In Tiziana Margaria & Bernhard Steffen, editors: *9th International Symposium on Leveraging Applications of Formal Methods (ISoLA), Lecture Notes in Computer Science* 12479, Springer, pp. 216–241, doi:10.1007/978-3-030-83723-5_15.

[15] Yuliya Butkova, Arnd Hartmanns & Holger Hermanns (2021): *A Modest Approach to Markov Automata*. *ACM Trans. Model. Comput. Simul.* 31(3), pp. 14:1–14:34, doi:10.1145/3449355.

[16] Pedro R. D'Argenio, Juan A. Fraire & Arnd Hartmanns (2020): *Sampling Distributed Schedulers for Resilient Space Communication*. In Ritchie Lee, Susmit Jha & Anastasia Mavridou, editors: *12th International NASA Formal Methods Symposium (NFM), Lecture Notes in Computer Science* 12229, Springer, pp. 291–310, doi:10.1007/978-3-030-55754-6_17.

[17] Pedro R. D'Argenio, Arnd Hartmanns, Axel Legay & Sean Sedwards (2016): *Statistical Approximation of Optimal Schedulers for Probabilistic Timed Automata*. In Erika Ábrahám & Marieke Huisman, editors: *12th*

International Conference on Integrated Formal Methods (iFM), Lecture Notes in Computer Science 9681, Springer, pp. 99–114, doi:10.1007/978-3-319-33693-0_7.

[18] Pedro R. D'Argenio, Arnd Hartmanns & Sean Sedwards (2018): *Lightweight Statistical Model Checking in Nondeterministic Continuous Time*. In Tiziana Margaria & Bernhard Steffen, editors: *8th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA)*, Lecture Notes in Computer Science 11245, Springer, pp. 336–353, doi:10.1007/978-3-030-03421-4_22.

[19] Pedro R. D'Argenio, Bertrand Jeannet, Henrik Ejersbo Jensen & Kim Guldstrand Larsen (2002): *Reduction and Refinement Strategies for Probabilistic Analysis*. In Holger Hermanns & Roberto Segala, editors: *Second Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM-PROBMIV)*, Lecture Notes in Computer Science 2399, Springer, pp. 57–76, doi:10.1007/3-540-45605-8_5.

[20] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis & Zheng Wang (2011): *Time for Statistical Model Checking of Real-Time Systems*. In Ganesh Gopalakrishnan & Shaz Qadeer, editors: *23rd International Conference on Computer Aided Verification (CAV)*, Lecture Notes in Computer Science 6806, Springer, pp. 349–355, doi:10.1007/978-3-642-22110-1_27.

[21] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen & Matthias Volk (2017): *A Storm is Coming: A Modern Probabilistic Model Checker*. In Rupak Majumdar & Viktor Kuncak, editors: *29th International Conference on Computer Aided Verification (CAV)*, Lecture Notes in Computer Science 10427, Springer, pp. 592–600, doi:10.1007/978-3-319-63390-9_31.

[22] Christian Eisentraut, Holger Hermanns & Lijun Zhang (2010): *On Probabilistic Automata in Continuous Time*. In: *25th Annual IEEE Symposium on Logic in Computer Science (LICS)*, IEEE Computer Society, pp. 342–351, doi:10.1109/LICS.2010.41.

[23] Ansgar Fehnker & Kaylash Chaudhary (2018): *Twenty Percent and a Few Days – Optimising a Bitcoin Majority Attack*. In Aaron Dutle, César A. Muñoz & Anthony Narkawicz, editors: *10th International NASA Formal Methods Symposium (NFM)*, Lecture Notes in Computer Science 10811, Springer, pp. 157–163, doi:10.1007/978-3-319-77935-5_11.

[24] 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)*, ACM, pp. 43–52, doi:10.1145/1967701.1967710.

[25] Sergio Giro & Pedro R. D'Argenio (2007): *Quantitative Model Checking Revisited: Neither Decidable Nor Approximable*. In Jean-François Raskin & P. S. Thiagarajan, editors: *5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, Lecture Notes in Computer Science 4763, Springer, pp. 179–194, doi:10.1007/978-3-540-75454-1_14.

[26] Sergio Giro & Pedro R. D'Argenio (2009): *On the Expressive Power of Schedulers in Distributed Probabilistic Systems*. Electron. Notes Theor. Comput. Sci. 253(3), pp. 45–71, doi:10.1016/j.entcs.2009.10.005.

[27] Ernst Moritz Hahn & Arnd Hartmanns (2016): *A Comparison of Time- and Reward-Bounded Probabilistic Model Checking Techniques*. In Martin Fränzle, Deepak Kapur & Naijun Zhan, editors: *Second International Symposium on Dependable Software Engineering: Theories, Tools, and Applications (SETTA)*, Lecture Notes in Computer Science 9984, pp. 85–100, doi:10.1007/978-3-319-47677-3_6.

[28] Ernst Moritz Hahn, Arnd Hartmanns, Christian Hensel, Michaela Klauck, Joachim Klein, Jan Kretínský, David Parker, Tim Quatmann, Enno Ruijters & Marcel Steinmetz (2019): *The 2019 Comparison of Tools for the Analysis of Quantitative Formal Models (QComp 2019 Competition Report)*. In Dirk Beyer, Marieke Huisman, Fabrice Kordon & Bernhard Steffen, editors: *25 Years of TACAS: TOOLympics*, Lecture Notes in Computer Science 11429, Springer, pp. 69–92, doi:10.1007/978-3-030-17502-3_5.

[29] 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.

[30] Arnd Hartmanns & Holger Hermanns (2014): *The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification*. In Erika Ábrahám & Klaus Havelund, editors: *20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science 8413, Springer, pp. 593–598, doi:10.1007/978-3-642-54862-8_51.

[31] Arnd Hartmanns & Holger Hermanns (2015): *Explicit Model Checking of Very Large MDP Using Partitioning and Secondary Storage*. In Bernd Finkbeiner, Geguang Pu & Lijun Zhang, editors: *13th International Symposium on Automated Technology for Verification and Analysis (ATVA)*, Lecture Notes in Computer Science 9364, Springer, pp. 131–147, doi:10.1007/978-3-319-24953-7_10.

[32] Arnd Hartmanns & Holger Hermanns (2019): *A Modest Markov Automata Tutorial*. In Markus Krötzsch & Daria Stepanova, editors: *15th International Reasoning Web Summer School*, Lecture Notes in Computer Science 11810, Springer, pp. 250–276, doi:10.1007/978-3-030-31423-1_8.

[33] Arnd Hartmanns, Michaela Klauck, David Parker, Tim Quatmann & Enno Ruijters (2019): *The Quantitative Verification Benchmark Set*. In Tomás Vojnar & Lijun Zhang, editors: *25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science 11427, Springer, pp. 344–350, doi:10.1007/978-3-030-17462-0_20.

[34] Arnd Hartmanns, Sean Sedwards & Pedro R. D'Argenio (2017): *Efficient simulation-based verification of probabilistic timed automata*. In: *2017 Winter Simulation Conference (WSC)*, IEEE, pp. 1419–1430, doi:10.1109/WSC.2017.8247885.

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

[36] Michaela Klauck & Holger Hermanns (2021): *A Modest Approach to Dynamic Heuristic Search in Probabilistic Model Checking*. In Alessandro Abate & Andrea Marin, editors: *18th International Conference on Quantitative Evaluation of Systems (QEST)*, Lecture Notes in Computer Science 12846, Springer, pp. 15–38, doi:10.1007/978-3-030-85172-9_2.

[37] Maximilian A. Köhl, Michaela Klauck & Holger Hermanns (2021): *Momba: JANI Meets Python*. In Jan Friso Groote & Kim Guldstrand Larsen, editors: *27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Lecture Notes in Computer Science 12652, Springer, pp. 389–398, doi:10.1007/978-3-030-72013-1_23.

[38] Marta Z. Kwiatkowska, Gethin Norman & David Parker (2011): *PRISM 4.0: Verification of Probabilistic Real-Time Systems*. In Ganesh Gopalakrishnan & Shaz Qadeer, editors: *23rd International Conference on Computer Aided Verification (CAV)*, Lecture Notes in Computer Science 6806, Springer, pp. 585–591, doi:10.1007/978-3-642-22110-1_47.

[39] Marta Z. Kwiatkowska, Gethin Norman, David Parker & Jeremy Sproston (2006): *Performance analysis of probabilistic timed automata using digital clocks*. Formal Methods Syst. Des. 29(1), pp. 33–78, doi:10.1007/s10703-006-0005-2.

[40] Marta Z. Kwiatkowska, Gethin Norman, Roberto Segala & Jeremy Sproston (2002): *Automatic verification of real-time systems with discrete probability distributions*. Theor. Comput. Sci. 282(1), pp. 101–150, doi:10.1016/S0304-3975(01)00046-9.

[41] Axel Legay, Sean Sedwards & Louis-Marie Traonouez (2014): *Scalable Verification of Markov Decision Processes*. In Carlos Canal & Akram Idani, editors: *4th Workshop on Formal Methods in the Development of Software (WS-FMDS)*, Lecture Notes in Computer Science 8938, Springer, pp. 350–362, doi:10.1007/978-3-319-15201-1_23.

[42] Benjamin Lewis, Arnd Hartmanns, Prabal Basu, Rajesh Jayashankara Shridevi, Koushik Chakraborty, Sanghamitra Roy & Zhen Zhang (2019): *Probabilistic Verification for Reliable Network-on-Chip System Design*. In Kim Guldstrand Larsen & Tim A. C. Willemse, editors: *24th International Conference on Formal Methods for Industrial Critical Systems (FMICS)*, Lecture Notes in Computer Science 11687, Springer, pp. 110–126, doi:10.1007/978-3-030-27008-7_7.

[43] Fernando D. Raverta, Juan A. Fraire, Pablo G. Madoery, Ramiro A. Demasi, Jorge M. Finochietto & Pedro R. D'Argenio (2021): *Routing in Delay-Tolerant Networks under uncertain contact plans*. Ad Hoc Networks 123, p. 102663, doi:10.1016/j.adhoc.2021.102663.

[44] Riley Roberts, Benjamin Lewis, Arnd Hartmanns, Prabal Basu, Sanghamitra Roy, Koushik Chakraborty & Zhen Zhang (2021): *Probabilistic Verification for Reliability of a Two-by-Two Network-on-Chip System*. In Alberto Lluch-Lafuente & Anastasia Mavridou, editors: *26th International Conference on Formal Methods for Industrial Critical Systems (FMICS)*, Lecture Notes in Computer Science 12863, Springer, pp. 232–248, doi:10.1007/978-3-030-85248-1_16.

[45] Gerardo Rubino & Bruno Tuffin, editors (2009): *Rare Event Simulation using Monte Carlo Methods*. Wiley, doi:10.1002/9780470745403.

[46] Jeremy Sproston (2000): *Decidable Model Checking of Probabilistic Hybrid Automata*. In Mathai Joseph, editor: *6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT)*, Lecture Notes in Computer Science 1926, Springer, pp. 31–45, doi:10.1007/3-540-45352-0_5.