Model checking randomized distributed algorithms

Nathalie Bertrand, Univ. Rennes, Inria, CNRS, IRISA – Rennes (France)

Randomization is a powerful paradigm to solve hard problems, especially in distributed computing. Proving the correctness, and assessing the performances, of randomized distributed algorithms, is a very challenging research objective, that the verification community has started to address. In this article, we review existing model checking approaches to the verification of randomized distributed algorithms and identify further research directions.

1. RANDOMIZED DISTRIBUTED ALGORITHMS

Distributed algorithms appear in a variety of applications and of frameworks. Emblematic applications include telecommunications, scientific computing, and Blockchain that received recently a lot of attention. Although one could think distributed algorithms necessarily run on processors that are geographically distributed, the term also applies to algorithms running on shared-memory multiprocessors. Lynch identifies four main features to classify distributed algorithms [Lynch 1996]: the communication paradigm, the timing model, the type of failures, and the problem they solve. As for the *communication paradigm*, nodes in distant sites generally communicate via message passing (broadcast or rendez-vous), whereas multithreaded programs rather use global shared variables. The *timing model* ranges from synchrony to asynchrony. In the synchronous model, communications are immediate and processes take step simultaneously so that executions happen in synchronous rounds. In contrast, in the asynchronous model, processes can take steps in any order and at arbitrary respective speeds. Especially in a distributed settings, *failures* may need to be taken into account. Some algorithms assume complete reliability of the communication means and of the processes themselves, whereas fault-tolerant algorithms are –to some extent– robust to failures, for instance message losses, crashes of processes, or even malicious participants, the so-called *Byzantine* processes. Finally, a main feature to differentiate between distributed algorithms is the *addressed problem*: consensus, election of a leader, communication, database consistency, deadlock detection, etc.

Adversaries. Distributed algorithms are subject to several sources of non-determinism, especially in the asynchronous timing model (but not only). Indeed, non-determinism lies in the scheduling of the processes or their relative speeds, in the order of reception of messages, in the moment failures happen and the type of failures that happen, etc. The non-determinism is traditionally resolved by means of a global *adversary*, that *e.g.* schedules when messages are received, but also which process performs a step, etc. The distributed algorithm community has considered various classes of adversaries (weak, strong, fair, etc.) depending on their abilities. For instance fair adversaries schedule each process infinitely often and eventually deliver all sent messages; also, weak adversaries only have a limited view of the global system. When a new algorithm is proposed, beyond the communication paradigm, the timing model, the failure types, one has to make explicit the class of adversaries it is designed for.

Randomization in distributed algorithms. Since the seminal work of Rabin [Rabin 1976], randomization has proven to be a powerful tool to solve computationally hard problems. In particular, in the field of distributed computing, probabilities can yield more efficient solutions, or even permit to solve problems that are otherwise unsolvable.

The celebrated result by Fischer, Lynch and Paterson establishes that no distributed algorithm in the asynchronous timing model can achieve consensus assuming at least one process can crash [Fischer et al. 1985]. Consensus algorithms should satisfy three main properties: no two correct processes decide different values (*agreement*), correct processes may only decide a value that was initially proposed (*validity*), and correct processes eventually decide (*termination*). The impossibility of asynchronous consensus shows that any algorithm that satisfies agreement and validity necessarily has non-terminating executions. One way to rule out these infinite executions is to rely on randomness, so as to make them negligible. The termination property is then replaced with *almost-sure termination*, that is, termination with probability 1. Ben Or was the first to propose a randomized distributed algorithm to solve asynchronous consensus [Ben Or 1983]. The idea of using randomization to solve otherwise unsolvable problems was already put forward by Lehman and Rabin, when they gave a randomized solution to the dining philosophers problem [Lehmann and Rabin 1981]. In this problem, processes are arranged in a ring tology and can only communicate with their neighbours. Probabilities are crucial there to break symmetry between the participants so as to allow each philosopher to eventually eat. As a third example, randomization can also improve efficiency, for instance to perform mutual exclusion with shared variables of much smaller size than in the deterministic setting [Kushilevitz and Rabin 1992].

Randomization comes in several flavours in asynchronous randomized distributed algorithms. On the one hand, randomization can be part of the code that processes run. In Ben Or's broadcast consensus algorithm for instance, each process can invoke a coin, which determines with uniform probability the binary value it will start the next round with. On the other hand, randomization can be delegated to the adversary, that schedules in which order the processes take a step. For example, Aspnes proposes to replace randomness in the code the processes run by randomness in the environment in order to solve asynchronous consensus for shared-memory systems [Aspnes 2002]. In his proposal, the schedule of events decided by the adversary is perturbed by random noise drawn from a given distribution. This induces fairness on the order in which write and read operations are performed, and is enough to ensure almost-sure termination.

Towards formally verified randomized distributed algorithms. Readers of this verification column are probably already convinced there is a need for rigorous techniques to verify the correctness or detect bugs in computer systems, especially at early phases of their design. As far as randomized distributed algorithms are concerned, the combination of distributed aspects, hence non-determinism, and probabilities makes human reasoning difficult, even for properties as simple as almost-sure termination. Quoting Lehmann and Rabin [Lehmann and Rabin 1981]: "proofs of correctness for probabilistic distributed systems are extremely slippery". Again on the example of Ben Or's algorithm, the paper-and-pencil proof of its almost-sure termination appeared only thirty years after the algorithm was published [Aguilera and Toueg 2012]. The proofs are all the more difficult that one needs to take into account all possible resolutions of non-determinism by adversaries, and all possible number of participants.

Indeed, distributed algorithms are aimed at being correct for any number of processes, possibly with some constraint on the proportion of malicious ones for fault-tolerant algorithms. Parameterized verification, by which we mean the verification of models composed of many identical anonymous agents, recently regained interest in the model checking community: see [Esparza 2014] for a survey on the verification of so-called *crowds*. The literature on model-checking techniques for probabilistic crowds is currently scarce. Yet, to address the concern of Lehman and Rabin, we argue in favor of the development of such techniques to automatically prove the correctness of randomized distributed algorithms. As written by Lamport: “Model-checking algorithms prior to submitting them for publication should become the norm” [Lamport 2006]. We believe the model-checking community must provide parameterized verification algorithms and tools to help the distributed algorithms researchers to tend towards this norm.

Outline. In this article we review existing model-checking approaches to the verification of randomized distributed algorithms. They mainly concern distributed algorithms in the asynchronous timing model. The communication paradigms are varied: shared variables, broadcast, pairwise interactions. Also, randomization sometimes is inherent to the code ran by each process, or only appears in the way the adversaries schedule processes. Most approaches do not handle failures, yet some consider fault-tolerant distributed algorithms. Finally, some approaches we review are supported by a tool implementation, while some are more theoretical. This review is certainly subjective, and probably non-exhaustive. The author apologizes for unintentional omissions, and is interested in any feedback on this personal view.

2. VERIFICATION OF RANDOMIZED DISTRIBUTED ALGORITHMS

2.1. Finite-state probabilistic model-checking techniques

In the last decades, the verification community has gained interest in probabilistic models and properties, with the objective to generalise the model-checking approach to probabilistic frameworks. This research track started already in the eighties with algorithms for the qualitative verification of probabilistic models, enabling for example the proof of almost sure termination for finite-state concurrent probabilistic programs, by graph-based methods [Hart et al. 1983]. The first algorithms for quantitative verification of probabilistic models appeared in the nineties [Courcoubetis and Yannakakis 1995]. Since then, richer models (including nondeterminism, time, etc.) and properties (branching-time, long-run behaviours, etc.) were considered, giving birth to more and more sophisticated probabilistic model-checking algorithms. Their implementation into mature tools, such as the prominent PRISM [Prism] and more recently STORM [Storm], enabled the verification of hundreds of case studies, among which many randomized distributed algorithms.

On these case studies, the objective has been to automatically prove the correctness and to assess the performances of various algorithms. We list here a few such case studies to illustrate the variety of randomized distributed algorithms that have been model checked by state-of-the-art probabilistic model checkers.

Dining philosophers. PRISM has been used to verify the original randomized solution to the dining philosophers problem [Lehmann and Rabin 1981], as well as a later variant, with no fairness assumption on the adversary [Duflot et al. 2004]. More precisely, on instances of limited sizes (up to 20 participants), one can automatically verify the qualitative starvation-free property: almost-surely, hungry philosophers eventually eat; one can also assess the performances of the algorithms by computing the probability that a philosopher eats within a given number of steps, or the expected number of steps a philosopher needs to wait before eating. Details are available at the PRISM case studies webpage [Prism].

Self-stabilizing algorithms on ring topology. Several network algorithms aim at reaching a stable configuration when started in an illegal one. One can use PRISM to check that from any initial configuration, almost-surely a stable configuration is reached, independently of the resolution of non-determinism. Moreover, one can compute the maximum expected number of steps to do so, or the minimal probability to reach a stable configuration in a bounded number of steps. Kwiatkowska *et al.* report on benchmarks conducted on Herman's self-stabilization algorithm performed for up to 21 processes [Kwiatkowska et al. 2012].

Byzantine agreement. Assuming an upper bound on the proportion of malicious processes, fault-tolerant randomized consensus algorithms were proposed to achieve that correct processes eventually decide on a common value with probability 1 [Ben Or 1983; Cachin et al. 2005]. On the one hand, the proof assistant Cadence SMV was successfully used to prove the safety properties of agreement and validity independently on the number of processes and of the number of rounds. On the other hand, the probabilistic reasoning underlying their almost-sure termination has been proved with PRISM [Kwiatkowska and Norman 2002; Kwiatkowska et al. 2001] for up to 20 processes.

In all these case studies, the number of processes is fixed before invoking a probabilistic model checker. All processes execute the same code, so that *e.g.* the renaming functionality of PRISM allows one to easily write models for several values of the number of processes. The automated verification of randomized distributed algorithms then suffers the usual limitations model checkers have, namely a rapid state-space explosion when the number of processes grows. In practice only small instances of randomized distributed algorithms (for 10 to 20 processes) can be verified automatically. The algorithms implemented in state-of-the-art probabilistic model checkers fall short at validating randomized distributed algorithms with a large number of processes, all the more if it is a parameter. In the sequel, we review the few approaches in parameterized verification that address the verification of randomized distributed algorithms.

2.2. Population protocols

Angluin *et al.* introduced population protocols as a model for distributed computing, quite appropriate for sensor networks formed of many identical processes with limited computational resources [Angluin et al. 2004; 2006]. In this model, the processes communicate by rendez-vous, and the interacting pair is chosen uniformly at random. The adversary is thus the source of randomization here. Population protocols were for instance designed to implement a majority vote: assuming initially every process has an input value 0 or 1, the objective is to converge to a situation which represents whether there was a majority of 0's or of 1's (see *e.g.* [Aspnes and Ruppert 2009]). Beyond algorithms running on sensor netwoks, they can also model chemical reactions, or be used in systems biology to represent gene regulatory networks. In their simplest version, population protocols are not robust to crashes or Byzantine faults.

Although in population protocols the adversary chooses interacting pairs uniformly at random, as far as qualitative properties are concerned, only its fairness is important. To prove almost-sure termination of a population protocol for majority vote for instance, it is sufficient to assume that the adversary schedules all possible pairs to interact infinitely often. Building on this observation, several standard model-checking tools (for instance SPIN, PAT, or PRISM) were used to prove the correctness of some population protocols when the initial configuration is fixed [Pang et al. 2008; Clément et al. 2011]. Similarly to the case studies we listed in Section 2.1, the models are then finite-state transition systems or finite-state Markov chains, and when their sizes are reasonable –corresponding to a limited number of processes– the tools automatically prove correctness.

Beyond finite-state instances, many recents papers by Esparza and his colleagues develop parameterized verification techniques for population protocols. We recommend the surveys [Esparza 2017; Blondin et al. 2018c] as entry points to more results. Angluin *et al.* already proved that *well-formed* population protocols compute exactly Presburger-definable predicates [Angluin et al. 2006; Angluin et al. 2006]. When researchers from the verification community got interested into population protocols, they focused on decision problems, with the objective to come up with algorithms that would automatically "prove" population protocols. For instance, given as input a population protocol and a Presburger-definable predicate, a natural question is whether the population protocol computes that predicate. Or even the more elementary question of whether a given population protocol is well-formed, that is, computes a predicate. The latter problem is decidable, using non-trivial techniques from Petri nets analysis. Moreover, for positive instances, one can compute a representation of the computed predicate [Esparza et al. 2015; 2017].

Moving to quantitative questions, many problems turn out to be undecidable, as for instance the existence of an initial configuration such that the termination probability is at least $\frac{1}{2}$ [Esparza et al. 2016]. Yet, towards a quantitative analysis of the performances of population protocols, Blondin *et al.* provided an algorithm to bound the expected termination time: upon termination, it outputs a function $f(n)$, such that for every initial configuration with n participants, the expected time before a consensus is reached is bounded by $O(f(n))$ [Blondin et al. 2018b]. These developments on population protocols, as well as a simulation engine, are now implemented into the tool Peregrine [Blondin et al. 2018a].

2.3. Shared registers systems

Bouyer *et al.* introduced a model of a parameterized number of identical processes sharing finitely many registers they can read from, and write to [Bouyer et al. 2016] (see also [Stan 2017]). The local behaviour of each process is represented by an automaton, yet the adversary, that schedules in which order the processes take steps, is randomized: it chooses uniformly at random a process, and an enabled transition (depending on its current local state and the values of the shared registers). The semantics is thus an infinite-state Markov chain, although for each fixed initial configuration the reachable part is finite.

Given a local target state, and for the initial configuration with n processes, they consider the almost-sure reachability problem, that asks whether with probability 1, the Markov chain reaches a configuration where at least one process is in the target state. Interestingly, they show that, although the answer naturally depends on the precise value of n, it is asymptotically constant: there exists a *cutoff* value n_0 such that for every $n \geq n_0$ the answer is always positive (resp. always negative). Moreover, they show one can decide which is the case (ultimately positive, or ultimately negative) on a finite *symblicit* –partly symbolic and partly explicit– abstraction of the infinite-state Markov chain. The resulting algorithm runs in exponential space in the size of the description of the local automaton. To the best of our knowledge, there is currently no implementation of this algorithm.

Their work happens to be a first step towards verification of randomized distributed algorithms solving the consensus problem with communication via shared variables, such as the one by Aspnes [Aspnes 2002]. The latter algorithm works in rounds, and for each round there are finitely many shared registers. However, the number of rounds is *a priori* unbounded, thus so is the number of shared registers. Probabilities are induced by an unknown environment which perturbs the schedule of processes, resulting in a randomized adversary. Modelling Aspnes' consensus algorithm in noisy environment would therefore require an extension of the model studied in [Bouyer et al. 2016] to multiple rounds. The fact that in Aspnes' algorithm, a process may only read values from registers corresponding to the last and current rounds is an interesting property. It might be exploited to come up with an algorithm for the almost-sure reachability problem for models with multiple rounds, paving the way to the automated verification of the almost-sure termination of Aspnes' algorithm and the like.

2.4. Regular model-checking for algorithms on simple topologies

Focusing on algorithms for message-passing systems on simple network topologies (*e.g.* lines, or rings, or stars), Lin and his co-authors adopt a regular model-checking approach to parameterized verification of randomized distributed algorithms [Lin and Rümmer 2016; Lengál et al. 2017]. The idea of regular model checking is to represent a configuration of the system by a finite word, so that suitable sets of configurations are regular languages, and the transition relation between configurations is described by a transducer [Abdulla 2012].

In contrast to proving safety properties, proving liveness properties on probabilistic and parameterized models that represent randomized distributed algorithms is non-trivial, because the randomization is crucial to guarantee liveness properties such as almost-sure termination. As already explained, the objective is to prove, *e.g.* almost-sure termination, for every initial configuration, and under every possible adversary. It thus amounts to verifying an infinite family of finite-state Markov decision processes at once. Lin and Rümmer are the first to design a regular model-checking framework for the verification of liveness properties for randomized distributed algorithms running on simple topologies [Lin and Rümmer 2016]. To do so, they view the Markov decision process as a 2-player game between the adversary and the processes, and design a counter-example guided method to compute winning strategies for processes. Their technique applies in particular to the almost-sure termination of Lehmann and Rabin's randomized dining philosophers algorithm on a ring [Lehmann and Rabin 1981] and of Herman's self-stabilising randomized algorithm on a line [Herman 1990].

The almost-sure termination of the latter on a ring topology is however guaranteed only under *fair* adversaries. More precisely, the typical fairness constraint that imposes every process to be scheduled infinitely often is not sufficient. Therefore, the stronger notion of finitary fairness is preferred: delaying a process with an enabled action is only possible for some fixed –but unknown– number of steps. Incorporating this fairness hypothesis in the termination analysis can be done by *abstract program transformation*, a concept from concurrent systems [Francez 1986; Alur and Henzinger 1998]. Adapting abstract program transformation to randomized distributed algorithms, and showing this transformation could be expressed within the framework of regular model checking enabled the automated proof of liveness properties under fairness assumptions [Lengál et al. 2017].

2.5. Threshold automata for fault-tolerant algorithms

Fault-tolerant distributed algorithms are designed to work even under the failure of some of the processes. We consider here a setting with asynchronous communications by broadcast and Byzantine faults. The correctness of fault-tolerant distributed algorithms is always subject to an upper bound on the proportion of failures formalized by a *resilience condition*. For instance, Ben Or's randomized consensus algorithm is claimed to be correct when t –the maximal number of Byzantine processes– is smaller than $n/2$ –half of the total number of processes involved. The algorithm itself proceeds in rounds, and each has two phases. The broadcast of messages in the second phase is bound to the reception of sufficiently many messages of a given type. Such a threshold is typically a linear constraint on the parameters t and n, for instance $(n+t)/2$. Threshold automata were precisely defined to model (non-randomized) fault-tolerant distributed algorithms in which the steps are guarded by arithmetic constraints on the parameters [Konnov et al. 2017].

In order to model round-based randomized fault-tolerant distributed algorithm such as the randomized consensus by Ben Or, the model of threshold automata was recently extended with probabilistic transitions and multiple rounds [Bertrand et al. 2019]. More importantly, this work provides the first automated proofs of consensus algorithms that follow the ideas of Ben Or. To do so, one had to overcome several challenges. On the one hand, for safety properties already, even if the probabilistic choices can be replaced with non-deterministic ones, one must identify round invariants that enable a reduction to the verification of more complex specifications, yet on a single-round. On the other hand, to be able to prove almost-sure termination, inspired by the arguments in the hand-written proof of [Aguilera and Toueg 2012], one must justify that it is sufficient to prove termination with positive probability within a single round. For this reduction to be correct, it is necessary to restrict to *round-rigid adversaries*, that is adversaries that respect the round ordering. For safety properties as well as liveness ones, it is thus possible to reduce to checking properties on traditional threshold automata (with no probabilistic choices, and no rounds). Using the Byzantine model checker ByMC [Konnov and Widder 2018; ByMC], agreement, validity and almost-sure termination of several randomized consensus algorithms from the literature were successfully verified. This constitutes first steps towards parameterized verification of fault-tolerant randomized distributed algorithms such as the ones in Paxos or Blockchain.

3. CURRENT CHALLENGES

After this overview of existing approaches to model check randomized distributed algorithms, we would like to mention potential research directions. Quantitative analysis on the one hand, and synthesis on the other hand appear to be two grails in this area.

Towards quantitative analysis. As far as parameterized verification is concerned, the analysis currently restricts to qualitative properties, to the notable exception of the computation of the expected termination time for population protocols. Beyond this class of algorithms, and apart from expected termination time, being able to assess the performances of randomized distributed algorithms is certainly very interesting. Similarly to proving almost-sure termination automatically, there is no doubt the distributed algorithms community would be happy to have push-button tools to *e.g.* compute as a function of the parameters the expected number of rounds before termination, or the probability that a consensus is reached within a given number of steps.

For some models of populations of chemical or biological agents, mean-field techniques can be exploited when the asymptotic behaviour for an infinite number of components coincides with the average behaviour of a single component [Boudec et al. 2007]. This limit result can be used to design efficient model-checking algorithms for these population models, relying on the so-called fast simulation [Bortolussi et al. 2013; Bortolussi and Hillston 2015]. To retain finer information than the mere average behaviour, moments closure techniques permit to approximate higher-order moments of the stochastic process. In contrast to chemical or biological population models, randomized distributed algorithms involve shared data structures and rely on rich communication means possibly with evolving communication topology. Exploring how such techniques can be used for probabilistic distributed systems is thus quite challenging.

Abstractions and a CEGAR (counterexample guided abstraction refinement) framework was developed for finite-state probabilistic systems [Hermanns et al. 2008; Dehnert et al. 2012], with the objective to start with very a abstract model, and automatically refine it as long as it is not precise enough to answer a given quantitative question. The refinement step is guided by spurious counterexamples, present in the abstract model but not in the real system. Monotonic abstractions are often relevant for parameterised systems, and a CEGAR framework for monotonic abstractions was proposed: it automatically extracts from spurious counterexamples a set of configurations that is used in the refinement step [Abdulla et al. 2010]. Conceiving a CEGAR framework for randomized distributed algorithms seems non-trivial: appropriate predicates must be defined, and counterexamples must be generic enough to account for sets of parameter values.

Towards synthesis. Beyond the qualitative verification and quantitative evaluation of randomized distributed algorithms that often happen after the design phase, the synthesis problem aims at automatically generating algorithms that are correct-by-design. Already in the non-probabilistic case, parameterised synthesis, *i.e.* the design from scratch of parameterised systems enjoying given properties, is a notably hard problem: it is closely related to distributed synthesis, and one can only hope for general for semi-algorithms [Jacobs and Bloem 2014], or for solutions to restricted problems. To start with, concerning fault-tolerant randomized distributed algorithms, given an algorithm pattern, one may expect to synthesise automatically, *e.g.* threshold guards on variables that guarantee a property to hold, as in the non-probabilistic case [Lazic et al. 2017]. In the same direction, it would also be relevant to synthesise resilience conditions on the parameters (*e.g.* $t \leq n/2$) that ensure correctness of the algorithm.

Conclusion. The verification of (randomized) distributed algorithms is blooming in the model checking community. Given the numerous challenges that remain to be solved, we encourage any interested researcher in joining forces to work on this topic!

REFERENCES

Parosh Aziz Abdulla. 2012. Regular model checking. *STTT* 14, 2 (2012), 109–118. DOI:http://dx.doi.org/10.1007/s10009-011-0216-8

Parosh A. Abdulla, Yu-Fang Chen, Giorgio Delzanno, Frédéric Haziza, Chih-Duo Hong, and Ahmed Rezine. 2010. Constrained Monotonic Abstraction: A CEGAR for Parameterized Verification. In *Proceedings of the 21st International Conference on Concurrency Theory (CONCUR'10) (Lecture Notes in Computer Science)*, Vol. 6269. Springer, 86–101. DOI:http://dx.doi.org/10.1007/978-3-642-15375-4_7

Marcos K. Aguilera and Sam Toueg. 2012. The correctness proof of Ben Or's randomized consensus algorithm. *Distributed Computing* 25, 5 (2012), 371–381. DOI:http://dx.doi.org/10.1007/s00446-012-0162-z

Rajeev Alur and Thomas A. Henzinger. 1998. Finitary Fairness. *ACM Transactions on Programming Langages Systems* 20, 6 (1998), 1171–1194. DOI:http://dx.doi.org/10.1145/295656.295659

Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. 2004. Computation in networks of passively mobile finite-state sensors. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing (PODC'04)*. ACM, 290–299. DOI:http://dx.doi.org/10.1145/1011767.1011810

Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. 2006. Computation in networks of passively mobile finite-state sensors. *Distributed Computing* 18, 4 (2006), 235–253. DOI:http://dx.doi.org/10.1007/s00446-005-0138-3

Dana Angluin, James Aspnes, and David Eisenstat. 2006. Stably computable predicates are semilinear. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC'06)*. ACM, 292–299. DOI:http://dx.doi.org/10.1145/1146381.1146425

James Aspnes. 2002. Fast deterministic consensus in a noisy environment. *Journal of Algorithms* 45, 1 (2002), 16–39. DOI:http://dx.doi.org/10.1016/S0196-6774(02)00220-1

James Aspnes and Eric Ruppert. 2009. An Introduction to Population Protocols. In *Middleware for Network Eccentric and Mobile Applications*. Springer, 97–120. DOI:http://dx.doi.org/10.1007/978-3-540-89707-1_5

Michael Ben Or. 1983. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. In *Proceedings of the Second Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC'83)*. ACM, 27–30. DOI:http://dx.doi.org/10.1145/800221.806707

Nathalie Bertrand, Igor Konnov, Marijana Lazic, and Josef Widder. 2019. Verification of Randomized Consensus Algorithms Under Round-Rigid Adversaries. In *30th International Conference on Concurrency Theory (CONCUR'19) (LIPIcs)*, Vol. 140. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 33:1–33:15. DOI:http://dx.doi.org/10.4230/LIPIcs.CONCUR.2019.33

Michael Blondin, Javier Esparza, and Stefan Jaax. 2018a. Peregrine: A Tool for the Analysis of Population Protocols. In *30th International Conference on Computer Aided Verification (CAV'18) (Lecture Notes in Computer Science)*, Vol. 10981. Springer, 604–611. DOI:http://dx.doi.org/10.1007/978-3-319-96145-3_34

Michael Blondin, Javier Esparza, Stefan Jaax, and Antonín Kucera. 2018c. Black Ninjas in the Dark: Formal Analysis of Population Protocols. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'18)*. ACM, 1–10. DOI:http://dx.doi.org/10.1145/3209108.3209110

Michael Blondin, Javier Esparza, and Antonín Kucera. 2018b. Automatic Analysis of Expected Termination Time for Population Protocols. In *Proceedings of the 29th International Conference on Concurrency Theory, (CONCUR'18) (LIPIcs)*, Vol. 118. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 33:1–33:16. DOI:http://dx.doi.org/10.4230/LIPIcs.CONCUR.2018.33

Luca Bortolussi and Jane Hillston. 2015. Model checking single agent behaviours by fluid approximation. *Information and Computation* 242 (2015), 183–226. DOI:http://dx.doi.org/10.1016/j.ic.2015.03.002

Luca Bortolussi, Jane Hillston, Diego Latella, and Mieke Massink. 2013. Continuous approximation of collective system behaviour: A tutorial. *Performance Evaluation* 70, 5 (2013), 317–349. DOI:http://dx.doi.org/10.1016/j.peva.2013.01.001

Jean-Yves Le Boudec, David D. McDonald, and Jochen Mundinger. 2007. A Generic Mean Field Convergence Result for Systems of Interacting Objects. In *Proceedings of the 4th International Conference on the Quantitative Evaluation of Systems (QEST'07)*. IEEE Computer Society, 3–18. DOI:http://dx.doi.org/10.1109/QEST.2007.8

Patricia Bouyer, Nicolas Markey, Mickael Randour, Arnaud Sangnier, and Daniel Stan. 2016. Reachability in Networks of Register Protocols under Stochastic Schedulers. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP'16) (LIPIcs)*, Vol. 55. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 106:1–106:14. DOI:http://dx.doi.org/10.4230/LIPIcs.ICALP.2016.106

ByMC. ByMC: Byzantine Model Checker. http://www.forsyte.at/software/bymc/. (????). Accessed Dec. 2019.

Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. *Journal of Cryptology* 18, 3 (2005), 219–246. DOI:http://dx.doi.org/10.1007/s00145-005-0318-0

Julien Clément, Carole Delporte-Gallet, Hugues Fauconnier, and Mihaela Sighireanu. 2011. Guidelines for the Verification of Population Protocols. In *International Conference on Distributed Computing Systems (ICDCS'11)*. IEEE Computer Society, 215–224. DOI:http://dx.doi.org/10.1109/ICDCS.2011.36

Costas Courcoubetis and Mihalis Yannakakis. 1995. The Complexity of Probabilistic Verification. *Journal of the ACM* 42, 4 (1995), 857–907. DOI:http://dx.doi.org/10.1145/210332.210339

Christian Dehnert, Daniel Gebler, Michele Volpato, and David N. Jansen. 2012. On Abstraction of Probabilistic Systems. In *Advanced Lectures from the International Autumn School on Stochastic Model Checking. Rigorous Dependability Analysis Using Model Checking Techniques for Stochastic Systems (ROCKS'12) (Lecture Notes in Computer Science)*, Vol. 8453. Springer, 87–116. DOI:http://dx.doi.org/10.1007/978-3-662-45489-3_4

Marie Duflot, Laurent Fribourg, and Claudine Picaronny. 2004. Randomized dining philosophers without fairness assumption. *Distributed Computing* 17, 1 (2004), 65–76.

Javier Esparza. 2014. Keeping a Crowd Safe: On the Complexity of Parameterized Verification (Invited Talk). In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS'14) (LIPIcs)*, Vol. 25. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 1–10. DOI:http://dx.doi.org/10.4230/LIPIcs.STACS.2014.1

Javier Esparza. 2017. Advances in Parameterized Verification of Population Protocols. In *Proceedings of the 12th International Computer Science Symposium in Russia (CSR'17) (Lecture Notes in Computer Science)*, Vol. 10304. Springer, 7–14. DOI:http://dx.doi.org/10.1007/978-3-319-58747-9_2

Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. 2015. Verification of Population Protocols. In *Proceedings of the 26th International Conference on Concurrency Theory*

(CONCUR'15) (LIPIcs), Vol. 42. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 470–482. DOI: http://dx.doi.org/10.4230/LIPIcs.CONCUR.2015.470

Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. 2016. Model Checking Population Protocols. In *Proceedings of the 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'16) (LIPIcs)*, Vol. 65. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 27:1–27:14. DOI: http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2016.27

Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. 2017. Verification of population protocols. *Acta Informatica* 54, 2 (2017), 191–215. DOI: http://dx.doi.org/10.1007/s00236-016-0272-3

Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM* 32, 2 (1985), 374–382. DOI: http://dx.doi.org/10.1145/3149.214121

Nissim Francez. 1986. *Fairness*. Springer. DOI: http://dx.doi.org/10.1007/978-1-4612-4886-6

Sergiu Hart, Micha Sharir, and Amir Pnueli. 1983. Termination of Probabilistic Concurrent Program. *ACM Transactions on Programming Languages and Systems* 5, 3 (1983), 356–380. DOI: http://dx.doi.org/10.1145/2166.357214

Ted Herman. 1990. Probabilistic Self-Stabilization. *Inform. Process. Lett.* 35, 2 (1990), 63–67. DOI: http://dx.doi.org/10.1016/0020-0190(90)90107-9

Holger Hermanns, Björn Wachter, and Lijun Zhang. 2008. Probabilistic CEGAR. In *Proceedings of the 20th International Conference on Computer Aided Verification (CAV'08) (Lecture Notes in Computer Science)*, Vol. 5123. Springer, 162–175. DOI: http://dx.doi.org/10.1007/978-3-540-70545-1_16

Swen Jacobs and Roderick Bloem. 2014. Parameterized Synthesis. *Logical Methods in Computer Science* 10, 1 (2014). DOI: http://dx.doi.org/10.2168/LMCS-10(1:12)2014

Igor Konnov and Josef Widder. 2018. ByMC: Byzantine Model Checker. In *8th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation - Distributed Systems (ISoLA'18) (Lecture Notes in Computer Science)*, Vol. 11246. Springer, 327–342. DOI: http://dx.doi.org/10.1007/978-3-030-03424-5_22

Igor V. Konnov, Helmut Veith, and Josef Widder. 2017. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. *Information and Computation* 252 (2017), 95–109. DOI: http://dx.doi.org/10.1016/j.ic.2016.03.006

Eyal Kushilevitz and Michael O. Rabin. 1992. Randomized Mutual Exclusion Algorithms Revisited. In *Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing (PODC'92)*. ACM, 275–283. DOI: http://dx.doi.org/10.1145/135419.135468

Marta Z. Kwiatkowska and Gethin Norman. 2002. Verifying Randomized Byzantine Agreement. In *FORTE*. 194–209.

Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2012. Probabilistic Verification of Hermans Self-Stabilisation Algorithm. *Formal Aspects of Computing* 24, 4 (2012), 661–670.

Marta Z. Kwiatkowska, Gethin Norman, and Roberto Segala. 2001. Automated Verification of a Randomized Distributed Consensus Protocol Using Cadence SMV and PRISM. In *Proceedings of the 13th International Conference on Computer Aided Verification (CAV'01) (Lecture Notes in Computer Science)*, Vol. 2102. Springer, 194–206. DOI: http://dx.doi.org/10.1007/3-540-44585-4_17

Leslie Lamport. 2006. Checking a Multithreaded Algorithm with $^{+}$CAL. In *Proceedings of the 20th International Symposium on Distributed Computing (DISC'06) (Lecture Notes in Computer Science)*, Vol. 4167. Springer, 151–163. DOI: http://dx.doi.org/10.1007/11864219_11

Marijana Lazic, Igor Konnov, Josef Widder, and Roderick Bloem. 2017. Synthesis of Distributed Algorithms with Parameterized Threshold Guards. In *Proceedings of the 21st International Conference on Principles of Distributed Systems (OPODIS'17)*. (to appear).

Daniel J. Lehmann and Michael O. Rabin. 1981. On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem. In *Proceedings of the 8th Annual ACM Symposium on Principles of Programming Languages (POPL'81)*. ACM Press, 133–138. DOI: http://dx.doi.org/10.1145/567532.567547

Ondrej Lengál, Anthony Widjaja Lin, Rupak Majumdar, and Philipp Rümmer. 2017. Fair Termination for Parameterized Probabilistic Concurrent Systems. In *23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'17) (Lecture Notes in Computer Science)*, Vol. 10205. Springer, 499–517. DOI: http://dx.doi.org/10.1007/978-3-662-54577-5_29

Anthony W. Lin and Philipp Rümmer. 2016. Liveness of Randomised Parameterised Systems under Arbitrary Schedulers. In *28th International Conference on Computer Aided Verification (CAV'16) (Lecture Notes in Computer Science)*, Vol. 9780. Springer, 112–133. DOI: http://dx.doi.org/10.1007/978-3-319-41540-6_7

Nancy A. Lynch. 1996. *Distributed Algorithms*. Morgan Kaufmann.

Jun Pang, Zhengqin Luo, and Yuxin Deng. 2008. On Automatic Verification of Self-Stabilizing Population Protocols. In *Proceedings of the 2nd IEEE/IFIP International Symposium on Theoretical Aspects of Software Engineering (TASE'08)*. IEEE Computer Society, 185–192. DOI: http://dx.doi.org/10.1109/TASE.2008.8

Prism. PRISM case studies. http://www.prismmodelchecker.org/casestudies/index.php. (????). Accessed Dec. 2019.

Michael O. Rabin. 1976. Probabilistic Algorithms. In *Algorithms and Complexity: New directions and recent results*. Academic Press, 21–39.

Daniel Stan. 2017. *Randomized strategies in concurrent games*. Ph.D. Dissertation. University of Paris-Saclay, France.

Storm. STORM model checker. http://www.stormchecker.org/index.html. (????). Accessed Dec. 2019.