Analysing Randomized Distributed Algorithms*

Gethin Norman

School of Computer Science, University of Birmingham,
Birmingham B15 2TT, United Kingdom
G.Norman@cs.bham.ac.uk

Abstract. Randomization is of paramount importance in practical applications and randomized algorithms are used widely, for example in co-ordinating distributed computer networks, message routing and cache management. The appeal of randomized algorithms is their simplicity and elegance. However, this comes at a cost: the analysis of such systems become very complex, particularly in the context of distributed computation. This arises through the interplay between probability and nondeterminism. To prove a randomized distributed algorithm correct one usually involves two levels: classical, assertion-based reasoning, and a probabilistic analysis based on a suitable probability space on computations. In this paper we describe a number of approaches which allows us to verify the correctness of randomized distributed algorithms.

1 Introduction

Distributed algorithms [66] are designed to run on hardware consisting of many interconnected processors. The term 'distributed' originates from algorithms that used to run over a geographically large network, but now applies equally well to shared memory multiprocessors. A *randomized* algorithm is one which contains an assignment to a variable based on the outcome of tossing a fair coin or a random number generator. Since the seminal paper by Michael Rabin [85] randomized algorithms have won universal approval, basically for two reasons: simplicity and speed. Randomized distributed algorithms include a number of theoretical algorithmic schemes, for example the dining philosophers protocol and the consensus of Aspnes and Herlihy [5], as well as the practical real-world protocols for Byzantine agreement due to Kursawe et al. [11] and IEEE 1394 FireWire root contention protocol [46].

A necessary consequence of using randomization is the fact that the correctness statements must combine probabilistic analysis, typically based on some appropriate probability space on computation paths, with classical, assertion-based reasoning. Examples include: "termination occurs *with probability* 1", "delivery of a video frame is guaranteed within time t with probability *at least* 0.98" and "the expected number of rounds until a leader is elected is at most 2^k". The

* Supported in part by the EPSRC grant GR/N22960.

analysis of randomized distributed algorithms becomes very complex, sometimes leading to errors.

In a distributed environment, where each of the concurrent processors must make decisions in a highly nondeterministic context, it can be shown that there are *no* (symmetric) deterministic solutions to certain network co-ordination problems [63, 33]. Randomization offers a powerful tool for *symmetry breaking*, in addition leading to faster solutions. It has been applied to a range of algorithms, for example the dining philosophers protocol [63], Byzantine agreement [8], self stabilization [41], and shared-memory consensus [5]. It should be emphasised that the standard liveness properties, for example termination, become probabilistic (hold with probability 1) in the context of randomized algorithms. This applies to the so called *Monte Carlo* algorithms, which are the focus of this paper, but not to the *Las Vegas* algorithms such as the randomized quicksort [77]. In Las Vegas algorithms termination can only be ensured with some appropriately high probability, and so non-termination simply becomes unlikely, albeit still possible.

Randomization is of paramount importance in practical applications: randomized algorithms are used widely, for example in co-ordinating distributed computer networks [66], message routing [1], data structures, cache management, and graph and geometric algorithms [77]. The appeal of randomized algorithms is their simplicity and elegance. However, this comes at a cost: the probability space, and with it the probabilistic analysis, become very complex, particularly in the context of distributed computation. This is caused by both the complex interplay between probabilistic and nondeterministic choices and by the introduction of *dependencies* between random variables that can easily be overlooked. To prove a randomized distributed algorithm correct one usually involves two levels: classical, assertion-based reasoning, and a probabilistic analysis based on a suitable probability space on computations, see e.g. [67, 82, 1]. A correctness argument often crucially relies on the statement "*If* the random variables are independent *then* the algorithm produces the correct answer with probability p". The simplicity of this statement obscures potential dangers, which can arise if the variables turn out *not* to be independent, since in such cases estimated probabilities can differ from the *actual* probabilities by a wide margin.

For example, an error in a mutual exclusion algorithm due to Rabin [84], which was pointed out by Saias [90] and later corrected by Rabin [52], can be explained in terms of an inadvertently introduced dependence of the probability of a process winning the draw on the total number of processes in the network, and not only on those taking part in the draw; when exploited by a malicious adversary, this has the effect of making the actual probability of winning smaller than it should be. In a different but no less critical situation – probabilistic modelling of nuclear power stations – an error can be traced to the incorrect assumption of events being independent, which resulted in the actual probabilities becoming unsatisfactorily large [97]. Thus, satisfactory correctness guarantees are essential supporting evidence when implementing a new randomized algorithm.

2 Background

In this section we give an overview of areas related to analyzing randomized distributed algorithms, and then outline the remainder of the paper.

2.1 Modelling Randomized Distributed Algorithms

The standard modelling paradigm for systems exhibiting probabilistic behaviour is based on Markov chains. Such models often provide sufficient information to assess the probability of events occurring during execution of a sequential system, as each such execution can be represented as a sequence of "steps" made according to the probability distribution. We note that, for such models it is straightforward to define a probability measure of the set of executions (paths) of the model. However, this must be extended to take account of the distributed scenario, in which concurrently active processors handle a great deal of unspecified nondeterministic behaviour exhibited by their environment, as well as make probabilistic choices. As argued in [91], randomized distributed algorithms require models *with nondeterministic choice between probability distributions.* The central idea is to allow a *set* of probability distributions in each state. The choice between these distributions is made *externally* and *nondeterministically*, either by a scheduler that decides which sequential subprocess takes the next "step" (as in e.g. the concurrent Markov chains [101]), or by an *adversary* that has the power to influence and potentially confuse the system [9, 7], as is the case with Byzantine failure. A number of essential equivalent models exist following this paradigm and including *probabilistic automata* [91], *Markov Decision Processes* [26], *probabilistic-nondeterministic systems* [9] and *concurrent probabilistic systems* [7].

Probabilistic choices are *internal* to the process and made according to the selected distribution. It should be emphasised that the complexity introduced through allowing nondeterminism affects the probabilistic modelling: a probability space can be associated with the space of computations *only if* nondeterministic choices have been pre-determined, which is typically achieved through the choice of an adversary (which chooses at most one distribution in a given state).

As in the non-probabilistic setting, we may have to impose *fairness constraints* in order to ensure that liveness properties can be verified. In a distributed environment fairness corresponds to a requirement such as each concurrent component to progress whenever possible. Without fairness, certain liveness properties may trivially fail to hold in the presence of simultaneously enabled transitions of a concurrent component. A number of fairness notions have been introduced [101, 20, 7] for systems containing probabilistic and nondeterministic behaviour, for example in [7] an adversary is called *fair* if any choice of transitions that becomes enabled infinitely often along a computation path is taken infinitely often, whereas in [22] an adversary is called fair if there exists $\varepsilon > 0$ such that all nondeterministic choices are taken with probability at least ε The interested reader is referred to [7, 22] for more information on the subject.

To allow the construction of complex probabilistic systems it is straightforward to extend the definition of *parallel composition* in standard labelled transition systems to this probabilistic setting. Alternatively, to specify complex randomized distributed algorithms one can employ the higher-level modelling languages developed in the field of concurrency. Much of this work is based on replacing the (nondeterministic) choice operator with a probabilistic variant, and hence is not suitable for modelling randomized distributed algorithms where nondeterminism and probabilistic behaviour coexist. Of process calculi developed to include both a nondeterministic and probabilistic choice operator, we mention [36, 35, 104] which are based on CSS [73], and [64, 65, 76] where languages based on CSP [88] are considered.

2.2 Specifying Properties of Randomized Distributed Algorithms

Often the specification of a randomized distributed algorithm can be written in a simple unambiguous form, for example "*eventually the protocol terminates*" or "*the algorithm terminates in at most $\mathcal{O}(n)$ rounds*", and hence there is no need to use any specification language. However, one must be careful as without a formal specification language it is easy to make errors. An example of this, mentioned earlier, is in [84] where there is an error in the proof of correctness which can be attributed to the fact that the properties of the protocol were not stated correctly. For further details on this error and methods for correctly specifying properties see [90, 89].

Probabilistic logics are a natural choice of specification formalisms to use for stating correctness of randomized distributed algorithms. Such a logic can be obtained from a temporal logic, e.g. CTL [12], in two ways: either by adding probability operators with *thresholds* for truth to the syntax [37] (within this we include 'with probability 1' [101, 79]), or by re-interpreting the formulae to denote (an estimate of) *the probability* instead of the usual truth values [45, 69]. In the threshold approach, path formulae ϕ are annotated with thresholds $[\cdot]_{\geq p}$ for $p \in [0, 1]$, with the formula $[\phi]_{\geq p}$ interpreted as "the probability assigned to the set of paths satisfying the path formula ϕ in the classical sense is *at least* p". The fact that this semantics is well defined follows from the observation that in logics such as CTL path formulae determine measurable sets [101]. For models with nondeterminism quantification over paths A (E) can be added, meaning "for all (some) schedulers"; this models the *worst* (*best*) probabilistic outcome.

2.3 Verification Techniques for Distributed Algorithms

Model checking [12] has become an established industry standard for use in ensuring correctness of systems, and has been successful in exposing flaws in existing designs, to mention e.g. the tools SMV, SPIN and FDR, none of which can handle probability. The process of model checking involves building a (finite-state) *model* of the system under consideration, typically built from components composed in parallel. Then the model checking tool reads the description, builds a model, and for each specification (in temporal/modal logic) attempts to exhaustively check whether it holds in this model; if not, then a diagnostic trace leading to error is output.

One problem with the model checking approach, which makes model checking complex systems infeasible, is the well know *state space explosion problem*: the number of states grows exponentially with the number of components in the design. Furthermore, often the system under consideration is either infinite state or parameterized, for example by the number of processes, and we would like to prove that the system is correct for *any* value of the parameter. In such cases we cannot use finite state model checking, however, one approach to overcome these difficulties and allow for the analysis of both parameterized and infinite state systems is to develop a *verification methodology* which reduces the verification problem for large/infinite state systems to *tractable finite-state* subgoals that can be discharged by a conventional model checker. In the non-probabilistic setting *abstraction* techniques have been used to reduce the complexity of the system under study by constructing a simpler abstract system which *weakly* preserves temporal properties: if a property holds in the abstract system then it holds true in the concrete system, while the converse does not hold. Other techniques developed in the non-probabilistic case include *compositional/refinement* methods [40, 72], *abstract interpretation* [15], *temporal case splitting* [71] and *data independent induction* [16].

For example, *data type reduction*, a specific instance of abstract interpretation, can be used to reduce large or *infinite* types to small finite types, and temporal case splitting breaks the proof into *cases* based on the value of a given variable. Combining data type reduction and temporal case splitting can reduce a complex proof to checking only a small number of simple subcases, thus achieving significant space savings.

We should note that the above techniques can no longer be described as fully automated, it is often up to the user to decide on the abstraction or how the problem should be decomposed. This may not be any easy task as it often requires a detailed understanding of both how the system being modelled works and the verification methodology.

2.4 Outline of Paper

In the next two sections we review techniques for verifying randomized distributed algorithms and give examples applying these methods. In Section 3 we consider methods for verifying *qualitative* properties, that is, properties which require satisfaction with probability 1. Whereas, in Section 4 we consider the case of the more general *quantitative* properties such as “the probability that a leader is elected within t steps is at least 0.75”. In Section 5 we outline the complete verification methodology for verifying a complex randomized distributed algorithm. Finally in Section 6 we summarise and consider future research directions.

3 Qualitative Analysis

In this section we outline several techniques used to verify correctness with probability 1, called P-validity in [51, 105]. Examples of such correctness statements include "*the protocol eventually terminates*", "*eventually a leader is elected*" and "*eventually the resource gets allocated*".

3.1 Verifying Finite-State Systems

In the case of finite state problems, the verification of probability 1 properties reduces to graph based analysis, i.e. conventional model checking. This result was first established in [38] in the case of termination of finite state systems. In [78] this is extended by introducing deductive proof rules for proving that termination properties of (finite state) probabilistic systems hold with probability 1. While in [79, 102] a sound and complete methodology for establishing correctness with probability 1 for general temporal logic properties are introduced, as well as model checking procedures. We also mention approaches based on the branching time framework [20, 7] where model checking algorithms for probability 1 properties under different notions of fairness are introduced. In all of the above techniques the verification is performed over a non-probabilistic version of the system, in which the probabilistic choices are replaced with nondeterminism with fairness constraints.

Applying such techniques has lead to the verification of a number of randomized distributed algorithms. In particular by using *symbolic techniques*, for example BDDs, one can verify very large randomized distributed algorithms. For example, using the probabilistic symbolic model checker PRISM [54, 83] qualitative analysis of randomized algorithms with over 10^{30} states has been performed. However, in general randomized distributed protocols are often too complex to apply these methods. Furthermore, often the protocols include parameters which are unbounded (for example, the number of rounds), or are genuinely infinite state. In such cases, we may apply model checking techniques to prove that certain instances of the protocol are correct, for example when there are 6 processes in the ring, but we cannot say that the protocol is correct for all instances, for example that the protocol is correct for a ring of *any* size. In the next section, we will consider alternative verification techniques which allow us to prove correctness in such cases.

3.2 Verifying Complex, Parameterized and Infinite-State Systems

As mentioned above, many randomized distributed algorithms are parameterized by, for example the number of processes or nodes in the network, or are infinite state, for example real-time protocols, in such cases the verification problem becomes harder. In this section we will consider methods for establishing qualitative (probability 1) properties of such systems (in Section 4.2 we will consider checking quantitative properties).

In certain cases, for example when considering real-time randomized protocols, although the protocol has an infinite state space one can reduce the verification problem to checking a finite-state quotient of the system, for example [2] present a method for probabilistic real-time systems, using the region graph approach [3]. In this work it is demonstrated that, verifying whether (infinite state) probabilistic real time systems satisfy branching time temporal properties with probability 1, reduces to analyzing a finite-state nondeterministic system.

In addition, abstraction/refinement methods can be applied to simplify the verification of probability 1 properties. For example, in [100] a sequence of stepwise abstractions are used in the verification of the (real-time) IEEE 1394 FireWire root contention protocol [46]. In each step of the refinement process one aspect of the protocol is abstracted, for example one step concerns abstracting timing information, where the correctness of each abstraction is validation through probabilistic simulations [94]. After the final step of the refinement processes one is left with a simple system which is straightforward to analyse.

In the non-probabilistic case, as mentioned in Section 2, much progress has been made in methods for the verification of parameterized and infinite state systems. For certain protocol and properties, after replacing the probabilistic behaviour with nondeterminism, one can directly use these techniques for the verification of randomized distributed algorithms. For example, using Cadence SMV [72], a proof assistant which allows the verification of large, complex, systems by reducing the verification problem to small subproblems that can be solved automatically by model checking, certain properties of the randomized consensus protocol of Aspnes and Herlihy [5] and the Byzantine agreement protocol of Cachin, Kursawe and Shoup [11] have been verified in [55] and [53] respectively. Note that, both these protocols have *unboundedly* many states and the proofs are for *any* number of processes.

In general however, to verify probability 1 properties as in the finite state case one must add fairness constraints when replacing the probabilistic behaviour with nondeterminism. The first example of verifying a non-trivial (parameterized) randomized distributed algorithm appeared in [78], where both a simplified version of Lehmann and Rabin's dining philosophers protocol [63] and a mutual exclusion algorithm are proved correct using deductive proof rules. The proofs presented are complex, require a detailed knowledge of the protocol and are constructed by hand.

The above technique has since been extended [51, 105] to allow one to employ "automated" methods when verifying correctness with probability 1 for parameterized probabilistic systems. This new method is based on the *network invariant* approach developed for proving liveness properties of non-probabilistic systems, see for example [50]. The advance in this work on the network invariant method, over the previous work in this area, which allows for the extension to probability 1 statements, is that in [50] fairness constraints of the system are taken into account. The idea behind the network invariant approach when proving correctness of a parameterized system with N processes $P_1, \ldots, P_N$, is to construct an abstraction of $N-1$ of the processes and the remaining process P_1.

Then show that: (a) composing in parallel two copies of the abstraction behave like the abstraction and (b) the property of interest holds over the abstraction composed with the single process P_1. The technique is not fully automated, although verifying the properties (a) and (b) can be performed mechanically, the construction of a suitable/correct abstraction must be performed manually and may require ingenuity and a deep understanding of the protocol under study. Furthermore, this approach is dependent on using *local arguments*, that is arguments that do not depend on the global state space since $N-1$ of the processes are abstracted, and therefore is only applicable when correctness can be proved using such arguments.

An alternative method for proving probability 1 properties of randomized distributed algorithms is presented in [29]. In this paper the techniques are based on results in Markov theory and in particular *recurrent* states and *0-1 laws*. The techniques developed are useful for proving "convergence" type properties of systems, for example *eventually a leader is elected* and *the protocol reaches a stable state*. The fundamental steps of this approach are finding a non-increasing measure over the states of the protocol and showing that, with some (non-negligible) probability, from any state this measure strictly decreases within a finite number of steps. This approach can be seen as a type of abstraction, where the set of states of the concrete protocol with the same measure are mapped to a single state in the abstract version. The main effort in using this method is finding the "correct" measure which depends on the protocol under consideration, however as the measure is defined on the states of the protocol on how the states are modelled can also influence this process. We will give a simple example illustrating this method in the next section.

3.3 Example: Verification of a Self-Stabilizing Protocol

In this section we consider the self stabilization algorithm of Israeli and Jalfon [47]. A self-stabilising protocol for a network of processes is a protocol which, when started from some possibly illegal start state, returns to a legal/stable state without any outside intervention within some finite number of steps. When the network is a ring of identical processes, the stable states are those where there is exactly one process designated as "privileged" (has a token). A further requirement in such cases is that the privilege (token) should be passed around the ring forever in a fair manner.

In the protocol of Israeli and Jalfon the network is a ring of N identical processes, $P_0, P_1, \ldots, P_{N-1}$. Each process P_i has a boolean variable q_i which represents the fact that the process has a token. A process is active if it has a token and *only* active processes can be scheduled. When an active process is scheduled, it makes a (uniform) random choice as to whether to move the token to its left (to process P_{i-1}) or right (to process P_{i+1}), and when a process ends up with two tokens, these tokens are merged into a single token. The stable states are those states where there is one token in the ring.

First, in any stable state (where there is only token), the privilege (token) is passed randomly around the ring, and hence is passed around the ring in a fair manner. Next we consider whether from any unstable state with probability 1 (under any scheduler) a stable state is reached. For simplicity we consider the case when in the unstable state there are two tokens in the ring, however the ring is of an arbitrary size. The correctness proof sketched below is based on that presented in [29] and is an example of a proof based on 0-1 laws and ergodic theory.

In any unstable state of the protocol (any state where there are two processes with tokens), the scheduler can only choose between the active processes who moves next, that is, between the two processes which have tokens. Without loss of generality, we can suppose that the two processes are i and j and that the minimum distance between the tokens is $d > 0$ as shown in Figure 1.

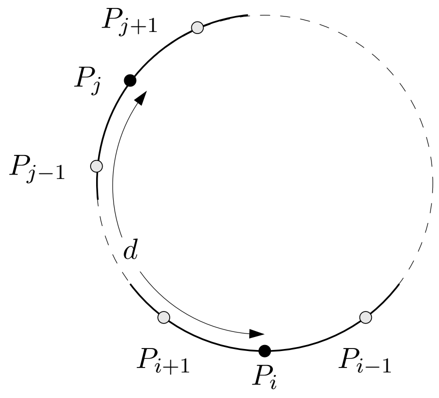

Fig. 1. Ring with two tokens

Now suppose that the scheduler chooses process i to move next, then with probability 1/2 the minimum distance between the two tokens will decrease by 1 (when P_i chooses to pass the token to P_{i+1}). On the other hand, if the scheduler chooses process P_j to move then again with probability 1/2 the minimum distance between the tokens will decrease by 1 (when P_j chooses to pass the token to P_{j-1}). Therefore, since i and j where arbitrary, in any unstable state, no matter what the scheduler decides – since it can only choose between the two processes which have tokens – with probability at least 1/2, the distance between the two tokens decreases by 1.

Letting $m = \lfloor N/2 \rfloor$, we have that the minimum distance between the tokens is less than or equal to m, and hence it follows that from *any* unstable state within m steps (and with probability greater than or equal to $1/2^m$) the distance between the tokens becomes 0 for any scheduler, that is, a stable state is reached (since when the minimum distance between the tokens is zero the tokens are merged into a single token). Using this and the fact that once a stable state is reached it is never left (since new tokens are never created), we can apply standard results from Markov theory to show that, for any adversary and from any unstable state, with probability 1, a stable state is reached.

For the complete formal proof for an arbitrary number of tokens see [29]. The interested reader is also referred to [30], where this method has been applied to a more complex setting: proving the correctness of a variant of Lehmann and Rabin's solution to the dining philosophers problem [63].

4 Quantitative Analysis

In this section we consider approaches and techniques for proving the correctness of quantitative properties. The verification of such properties is more complex than the probability 1 properties considered in the previous section. Examples of such properties include "*the probability of electing a leader within time t is at least 99%*" and *"the expected time until consensus is $\mathcal{O}(k)$"*. In the following section we will consider the verification of finite state systems, then in Section 4.2 consider the case for parameterized and infinite state system and finally in Section 4.3 we give a simple example demonstrating some of these techniques.

4.1 Verifying Finite-State Protocols

Much has been published on quantitative probabilistic model checking, see for example algorithms for probabilistic variants of CTL/CTL* [35, 9, 7], LTL [14] and the mu-calculus [45], and real-time probabilistic systems [56]. The fundamental method for assessing the probability of a path formula holding is through a reduction to solving a system of linear equations or linear optimization problem in the context of models with nondeterminism.

There are two approaches to extending temporal logics to the probabilistic setting. Using the threshold approach, probabilistic variants based on CTL* have been derived, see for example [9, 7], where minimum/maximum probability estimates replace exact probabilities. Furthermore, [20, 21, 23] extends the above logics to allowing the specification (and verification) of *expected time* and *long run average* properties using results from the field of Markov Decision Processes [26]. The alternative, *quantitative* interpretation, where formulae are maps from the set of states to the $[0, 1]$ interval instead of truth and falsity, has been considered in, for example [45]. We also mention the qualitative approach, based on probabilistic predicate transformers [75], presented in [69].

Model checking algorithms for the logic PCTL [9, 7] (an extension of CTL) have been implemented in the probabilistic model checker PRISM [54, 83] which has been used to verify a number of randomized distributed algorithms. For example, in [95] the probabilistic model checker PRISM has been used to verify the Crowds protocol [87], a randomized protocol designed to provide users with a mechanism for anonymous Web browsing. Further case studies can be found through the PRISM web page [83].

In [74] an alternative logic framework for reasoning about randomized distributed algorithm based on the program logic of "weakest preconditions" for Dijkstra's guarded command language *GCL* [27] is considered. The authors consider a probabilistic extension of *GCL*, called *pGCL* and the logic of "weakest precondition" is replaced with"greatest pre-expectation". More formally, in the non-probabilistic case [27], if P is a predicate over final states and *prog* is a (non-probabilistic) program, the "weakest precondition" predicate *wp.prog.P* is defined over initial states and holds only in the initial states from which the program *prog* is guaranteed to reach P. The authors' extension to the probabilistic framework is based on "greatest expectation": now a predicate returns for any initial state the maximum probability which the program *prog* is guaranteed to reach P from the initial state. In [68] it is shown how to formulate expected time directly in the logic, and the author demonstrates this by analysing Lehmann and Rabin's dining philosophers protocol [63].

As in the qualitative case, in general randomized distributed protocols are either too complex to apply probabilistic model checking, include parameters which are unbounded, for example the number of processes or the number of rounds, or are genuinely infinite state to apply these techniques. In the next section, we will consider alternative verification techniques capable of verifying such protocols.

4.2 Verifying Complex, Parameterized and Infinite-State Protocols

Methods for verifying complex randomized distributed algorithms include the approach developed in [103, 99, 98] which allow certain performance measures of I/O automata extended with probabilistic and real time behaviour to be computed compositionally. That is, the performance measures are computed component-wise and eliminate the need to explicitly construct the global state space, and hence combat the state explosion problem. We also mention the compositional, trace based, approach developed in [25] to allow for *assume-guarantee* type reasoning.

Alternatively, D'Argenio et al. [17, 18] present refinement strategies for checking quantitative reachability properties. The approach is based on constructing smaller *abstract* models of the system under study, which preserve reachability probabilities, in the sense that the probabilities obtained on the abstract system are upper and lower bounds on reachability probabilities of the concrete system. Furthermore, automated techniques are developed for refining the abstraction when the results obtained are inconclusive (i.e. when the bounds are too coarse). The authors have implemented these refinement strategies in the tool RAPTURE [86].

In [70] data refinement is extended to the probabilistic setting. Data refinement is a generalization of program refinement. Using the probabilistic programming language *pGCL* [74], the authors study the data refinement method in the probabilistic setting, and use this method to perform quantitative analysis of a probabilistic steam boiler.

In general to verify complex randomized distributed algorithms one can use equivalence relations to reduce the state space, and hence the complexity of the verification problem. Examples, include probabilistic simulation [48, 94], probabilistic bisimulation [61] and testing based equivalences [13].

As in the qualitative case, when studying infinite state systems, for example real-time randomized protocols, in certain cases one can reduce the verification problem to analyzing a finite-state quotient of the system. For general infinite state probabilistic systems methods for calculating *maximum reachability probabilities* have been developed in [57]. In the case of real-time systems, approaches have been developed in [56] for verifying properties of a probabilistic variant of the branching time logic TCTL [39]. The efficient algorithm for calculating bounds on the maximal reachability probability of probabilistic real-time automata presented in [56] has subsequently been implemented in [19]. For case studies concerning the qualitative verification of real-time randomized distributed algorithms protocols see [19, 59, 58].

An alternative approach is to use a theorem proving framework, such as the recent HOL support for the verification of probabilistic protocols [43, 44]. However, this technique has yet to be applied to *distributed* algorithms. We also mention the *approximate techniques* [62] which use Monte-Carlo algorithms to approximate the probability that a temporal formula is true. The advantage of this approach over model checking is that one does not need to construct the state space of the system, and hence reduce the state space explosion problem.

The final type of techniques we consider are those that try to separate the probabilistic and nondeterministic behaviour, in an attempt to isolate the probabilistic arguments required in the proof of correctness. The advantage of this *decomposition* of the verification problem is that simpler non-probabilistic methods can be used in the majority of the analysis, while the more complex probabilistic verification techniques need only be applied to some small isolated part of the protocol. Such approaches applicable to quantitative analysis include: *complexity statements* [91], *coin lemmas* [92, 34] and *scheduler luck games* [28]. We will now consider each of these methods in turn.

Probabilistic Complexity Statements [91, 67, 81, 93] are used to give time or complexity bounds for randomized distributed algorithms. A probabilistic complexity statement has the form:

$$U \xrightarrow{\phi \leq c}_{p} U'$$

where U and U' are sets of states, ϕ is a complexity measure, c is a nonnegative real number and $p \in [0,1]$. Informally the above probabilistic complexity statement means that:

whenever the protocol is in a state of U, under any adversary, the probability of reaching a state in U' within complexity c is at least p where the complexity is measured according to ϕ.

A complexity measure is used to determine the complexity of an execution of a system and examples of such measures include: the elapsed time, the number of updates of a variable, number of coin flips, and the number of rounds of a protocol. The key property of complexity statements is *compositionality*, that is complexity statements can be combined in the following way:

$$\text{if } U \xrightarrow{\phi \leq c}_{p} U' \text{ and } U' \xrightarrow{\phi \leq c'}_{p'} U'' \text{ then } U \xrightarrow{\phi \leq c+c'}_{p \cdot p'} U'' .$$

For this compositionality to hold there are certain conditions on the adversary that can be considered, however it has been shown that this compositionality property of complexity statements holds for fair adversaries [91]. This compositionality result can then be used to simplify the proof of correctness of randomized distributed algorithms into the verification of a number of smaller simpler problems. Furthermore, in [91] it is shown how using complexity statements one can derive upper bounds on the worst case performance of randomized distributed algorithms.

For correctness proofs employing probabilistic complexity statements, which also demonstrates how complexity statements can be used to verify expected time properties, see [81, 67, 82]. Additionally we will give an example of applying complexity statements in Section 5.

Coin Lemmas [92, 34, 93] are a tool for separating the probabilistic and nondeterministic arguments in the analysis of distributed probabilistic systems. They are used for proving upper and lower bounds on the probability of events. Their advantage lies in the reduction of probabilistic analysis to non-probabilistic steps and also force the user into a certain well-defined probabilistic scenario, drawing his or her attention to the possible interference between probability and nondeterministic choices, which has the effect of thus reducing the chance for making errors due to underestimating the complexity of the actual system execution scenario. We motivate the need for coin lemmas through the following example taken from [34].

> *Consider the experiment of rolling a die. We know that the probability of rolling any number between 1 and 6 is* $1/6$*. Now consider a simple protocol which can roll the die and sends a beep signal whenever the outcome is an even number. We could then say that the probability of the beep signal being sent is 1/2. However, it may not be the case that in every execution the protocol does roll the die, therefore the correct statement would be that in each execution the probability of either not rolling the die or observing a beep signal is at least 1/2.*

The observation in this simple example that one needs to take into the account whether the die is rolled or not is the basis for formulating coin lemmas. It may seem trivial in this simple example but as the complexity of the protocol grows and the number of experiments increases the result is no longer obvious. By adding the event "die is rolled" to the protocol, a coin lemma for this simple protocol is: with probability 1/2 either the event "die is rolled" is *not* observed or a beep signal is observed.

We now illustrate the use of coin lemmas through the following strategy for proving the correctness of probabilistic complexity statements. Suppose that we want to prove the correctness of the probabilistic complexity statement:

$$U \xrightarrow{\phi \leq c}_{p} U'$$

where the probability p arises through the probability that some designated random choices have a certain outcome. By fixing the outcome of these random choices and replacing all remaining random choices to nondeterministic choices we are left with a non-probabilistic system and given certain coin lemmas, by proving that *all* paths of this non-probabilistic system starting from any state in U reach a state in U' while ϕ increases by at most c, it follows that with probability p it holds in the original probabilistic system.

Scheduler Luck Games are introduced in [28], which can be seen as an instance of coin lemmas that can, in certain cases, provide a simpler and more concise correctness proof. For a given randomized distributed algorithm a game is set up between two players: *scheduler* and *luck*. The player *scheduler* decides how the nondeterminism is resolved in an attempt to disprove the correctness of the protocol, while the player *luck* chooses the outcome of some of the probabilistic choices in an attempt to verify the protocol as correct. The player *luck* is said to have a k winning strategy if by fixing at most k coin flips it is ensured that the protocol is correct. Intuitively, when the player has a k winning strategy, it follows that the protocol is correct with probability at least $1/2^k$.

In the cases of protocols with rounds, the authors show how such games can be used to infer expected time properties of randomized distributed algorithms. In particular, if luck has a winning strategy for the game in an expected number of at most r rounds with at most k interventions, (i.e. fixing at most k coin flips), then the protocol terminates within $r \cdot 2^k$ expected number of rounds.

4.3 Example: Verification of a Self Stabilizing Protocol

In this section we return to the self stabilizing protocol of Israeli and Jalfon [47] given in Section 3.3 and establish an upper bound on the expected time until a stable state is reached for an arbitrary sized ring. Again for simplicity restricting attention to when there are two tokens in the ring. The method is based on that given in [29].

We again consider the minimum distance between the two tokens in a ring. Now supposing the ring is of size N we have that the minimum distance between the two tokens can range from 0 to m, where $m = N/2$ if N is even and $(N-1)/2$ if N is odd. If we consider the behaviour of the adversary when the distance between the tokens is $0 < d < m$, then no matter which of the active processes (the two processes with tokens) the adversary chooses to scheduler, with probability 1/2 in the next state the distance is $d-1$ and with probability 1/2 the distance is $d+1$. If $d = 0$, then there is only one token (since the tokens are merged), and hence we have reached a stable state. On the other hand, if $d = m$, then when N is even with probability 1 in the next state the distance is $d-1$, on the other hand if N is odd with probability 1/2 in the next state the distance is $d-1$ and with probability 1/2 the distance is d. Intuitively, we can consider the protocol as a random walk with barriers $[0, m]$ (where 0 is absorbing) as shown in Figure 2.

More formally, we can show that there exists a *probabilistic simulation* [94] between the random walk and the protocol (by relating any unstable state of the protocol whose minimum distance between the tokens is d to the state of

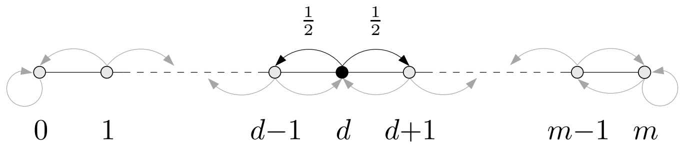

Fig. 2. Random walk with barriers 0 and m where 0 is absorbing

random walk which corresponds to being at d and any stable state of the protocol to being at 0). Then, using the fact that probabilistic simulation is sound with respect to trace inclusion [91], it follows that the expected time to reach a stable state is less than or equal to the maximum expected time to reach the barrier 0 from any state of the random walk. Hence, using random walk theory [32] it follows that the expected time to reach a stable state is bounded above by $\mathcal{O}(m^2) = \mathcal{O}(N^2)$.

5 Case Study: Byzantine Agreement

In this section we describe an approach to the formal verification of Cachin, Kursawe and Shoup's randomized Asynchronous Binary Byzantine Agreement protocol (ABBA) [11], which uses techniques for the verification of non-probabilistic parameterized protocols, (finite-state) probabilistic model checking and probabilistic complexity statements. Further details concerning the models we have constructed and the proof of correctness can be found at the PRISM web page [83]. The results presented in this section first appeared in [53].

Agreement problems arise in many distributed domains, for example, when it is necessary to agree whether to commit or abort a transaction in a distributed database. A *distributed agreement protocol* is an algorithm for ensuring that a collection of distributed parties, which start with some initial value (0 or 1) supplied by an environment, eventually terminate agreeing on the same value. The requirements for a randomized agreement protocol are:

***Validity*:** If all parties have the same initial value, then any party that decides must decide on this value.

***Agreement*:** Any two parties that decide must decide on the same value.

***Probabilistic Termination*:** Under the assumption that all messages between non-corrupted parties eventually get delivered, *with probability 1*, all initialized and non-corrupted parties eventually decide.

5.1 The Protocol

The ABBA protocol is set in a completely asynchronous environment, allows the maximum number of corrupted parties and makes use of cryptography and randomization. There are n parties, an adversary which is allowed to corrupt at most t of them (where $t < n/3$), and a trusted dealer. The parties proceed through possibly unboundedly many rounds: in each round, they attempt to agree by casting votes based on the votes of other parties. In addition to ***Validity*** and ***Agreement***, the protocol guarantees ***Probabilistic Termination*** in a constant expected number of rounds which is validated through the following property:

Fast Convergence: The probability that an honest party advances by more than $2r+1$ rounds is bounded above by $2^{-r}+\varepsilon$ where ε is a negligible function in the security parameter.

The Model and Cryptographic Primitives. The ABBA protocol is set in the *static* corruption model: the adversary must decide whom to corrupt at the very beginning of the execution of the protocol. Once the adversary has decided on the corrupted parties these are then simply absorbed into the adversary. The adversary also has complete control over the network: it can schedule and deliver the messages that it desires. The honest parties can therefore be considered as passive: they just respond to requests made by the adversary and do not change state in between such requests. Thus, the adversary can delay messages for an arbitrary length of time, except that it must eventually deliver each message.

The protocol uses two classes of cryptographic primitives. The first are *threshold random-access coin-tossing schemes*. Such a scheme models an unpredictable function F, of which each party holds a *share*, that maps the name of a coin for each round r to its value $F(r) \in \{0,1\}$. Each party can generate a share of each coin, where $n-t$ shares are both necessary and sufficient to construct the value of a particular coin. The second class of cryptographic primitives the protocol uses are *non-interactive threshold signature schemes*. These schemes are used to prevent the adversary from forging or modifying messages. In [11, 96] it has been shown that the cryptographic primitives have efficient implementations and are proved secure in the random oracle model. We therefore consider a version of the protocol which assumes the correctness of the cryptographic elements.

The Protocol. The protocol is given in Figure 3. Each party's initial value is sent to it by a trusted dealer. The parties proceed in rounds, casting pre-votes and main-votes. A party constructs both the *signature share* and *justification* for each pre-vote and main-vote it casts using threshold signature schemes. The justification for each vote is the *signature* obtained by combining the signature shares of the messages that the party used as the basis for this vote. For example, if a party casts a main-vote for 1 in round r, then the corresponding justification is the signature obtained through combining the signature shares present in the $n-t$ messages which contain pre-votes for 1 in round r that the party must have received. For further details on these justifications see [11].

Observe that the power of the adversary is limited by the requirement that all votes carry a signature share and a justification, and the assumption that the threshold signature scheme is secure (the adversary cannot forge either signature shares or signatures). The presence of the signature shares and this assumption implies that the adversary cannot forge any messages of the honest parties, that is, cannot send a message in which it pretends to be one of the honest parties.

Protocol ABBA for party i with initial value v_i.

0. Pre-Processing. Generate a signature share on the message

$$(\texttt{pre-process}, v_i)$$

and send all parties a message of the form

$$(\texttt{pre-process}, v_i, \textit{signature share}).$$

Collect $2t + 1$ pre-processing messages.

Repeat the following steps 1-4 for rounds $r = 1, 2, \ldots$

1. Pre-Vote. If $r = 1$, let v be the majority of the received pre-processing votes. Else, select $n - t$ justified main-votes from round $r - 1$ and let:

$$v = \begin{cases} 0 & \text{if there is a main-vote for 0} \\ 1 & \text{if there is a main-vote for 1} \\ F(r-1) & \text{if all main-votes are } \texttt{abstain}. \end{cases}$$

Produce a signature share on the message $(\texttt{pre-vote}, r, v)$ and the corresponding justification, then send all parties a message of the form

$$(\texttt{pre-vote}, r, v, \textit{justification}, \textit{signature share}).$$

2. Main-Vote. Collect $n-t$ properly justified round r pre-vote messages, and let

$$v = \begin{cases} 0 & \text{if there are } n-t \text{ pre-votes for 0} \\ 1 & \text{if there are } n-t \text{ pre-votes for 1} \\ \texttt{abstain} & \text{if there are pre-votes for 0 and 1.} \end{cases}$$

Produce a signature share on the message $(\texttt{main-vote}, r, v)$
and the corresponding justification, then send all parties a message of the form

$$(\texttt{main-vote}, r, v, \textit{justification}, \textit{signature share}).$$

3. Check for Decision. Collect $n - t$ justified main-votes of round r. If all these are main-votes for $v \in \{0, 1\}$, then decide on v, and continue for one more round.Otherwise simply proceed.

4. Coin. Generate a *coin share* of the coin for round r and send all parties a message of the form

$$(r, \textit{coin share}).$$

Collect $n - t$ shares of the coin for round r and combine to get the value of $F(r) \in \{0, 1\}$.

Fig. 3. Asynchronous binary Byzantine agreement protocol ABBA [11]

The adversary can make one honest party believe that the initial vote of a corrupted party is 0, while another honest party believes it is 1, since these messages do not require justification. However, since all the remaining votes need justification, the adversary cannot just make up the pre-votes and main-votes of the corrupted parties. For example, if in round r there are at least $n-t$ pre-votes for 0 and between 1 and $n-t-1$ pre-votes for 1 (all of which carry proper justification), then there is justification in round r for both a main-vote for 0 and for `abstain`, but not for 1. Thus, the adversary can make one honest party believe a corrupted party has a main-vote for 0 in round r, while another honest party believes that the same corrupted party has a main-vote for `abstain`.

Assumptions. Recall that, to verify the ABBA protocol correct, we need to establish the properties of ***Validity***, ***Agreement*** and ***Fast Convergence***. A number of assumptions were needed in order to perform the verification. These include the correctness of the cryptographic primitives; for example, we assume the following properties of the threshold coin-tossing scheme:

Robustness For any round r it is computationally infeasible for an adversary to produce $n-t$ valid shares of the coin for round r such that the output of the share combining algorithm is not $F(r)$.

Unpredictability An adversary's advantage in the following game is negligible. The adversary interacts with the honest parties and receives less than $n-2t$ shares of the coin for round r from honest parties, then at the end of the interaction outputs a bit $v \in \{0,1\}$. The adversaries advantage is defined as the distance from 1/2 of the probability that $F(r)=v$.

These assumptions are implicit in the models we construct, in that they restrict the power of the adversary. For example, the adversary cannot forge messages or make up any of the votes of the corrupted parties which require justification.

The remaining assumptions concern fairness statements which correspond to the fact that the adversary must eventually send all messages (which ensure that parties eventually cast votes). For example, we assume that

Proposition 1. *For any party that enters round $r+1$:*

(a) *if the party does not decide by round r, then the coin for round r is tossed;*
(b) *if the party does not decide by round $r+1$, then the coin in round $r+1$ is tossed.*

5.2 Agreement and Validity

Both these arguments are independent of the actual probability values, and hence can be verified by conventional model checking methods. Below we give a brief outline of the arguments based on two lemmas.

Lemma 1. *If in round r there are main-votes for v, then there are none for $\neg v$.*

Lemma 2. *If party i decides on v in round r, then there are less than $n-2t$ main-votes for* `abstain` *in round r from honest parties.*

***Validity*:** We prove that if all honest parties have the same initial preference, then all honest parties decide on this value in the initial round. Suppose all honest parties have the same initial value v, then in round 1 the pre-votes of all parties will be v, since all will see a majority of pre-processing votes for v (a majority of pre-processing votes requires at least $t+1$ votes, that is, at least one vote from an honest party). It then follows that all parties will have a main-vote for v in round 1, and hence all decide on v in the first round.

***Agreement*:** We prove that if the first honest party to decide decides on v in round r, then all honest parties decide on v either in round r or round $r+1$. Therefore, suppose party i is the first honest party to decide and it decides on v in round r. Then i must have received an honest main-vote for v, and hence, by Lemma 1, there are no main-votes for $\neg v$ in round r. Therefore, any party that decides in round r must decide on v. Now, by Lemma 2, there are less than $n-2t$ honest main-votes for `abstain`, and since a party reads at least $n-2t$ honest main-votes, a party must receive an honest main-vote for something other than `abstain` in round r and Lemma 1 implies this must be for v. Putting this together, all honest parties receive a main-vote for v and none for $\neg v$ in round r, thus all have a pre-vote for v in round $r+1$. It follows that all will have a main-vote for v in round $r+1$, and hence all will decide on v in round $r+1$.

In [83, 53] fully automated proofs of these properties, for all values of n, have been given using the Cadence SMV [72] a proof assistant.

5.3 Fast Convergence: High Level Proof

Before we give an outline of the proof of ***Fast Convergence*** we need to introduce the following notation. Let $\rho_r \in \{0,1\}$ be the value (when it exists) which at least $n-2t$ honest parties cast pre-votes for in round r. Since $2(n-2t) > n-t$ (see Lemma 1), in any round, there cannot be $n-2t$ honest pre-votes for v and for $\neg v$. Using this notation we introduce the following lemma.

Lemma 3. *If an honest party receives a justified main-vote for* `abstain` *in round $r+1$, then $\rho_r = 1 - F(r)$.*

Proof. Suppose an honest party receives a justified main-vote for `abstain` in round $r+1$. Then the justification must include both a pre-vote for 0 and for 1 in round $r+1$. However, as in any round there cannot be main-votes for 0 and 1 (see Lemma 1), one of these pre-votes must be justified by the value of the coin ($F(r)$) and the other by a main-vote (which must be for $1-F(r)$) in round r. We also have that if there is a main-vote in round r for v, then there are at least $n-t$ pre-votes for v in round r, and hence at least $n-2t$ honest pre-votes for v in round r. That is, we have $v = \rho_r$. Putting this together we have $\rho_r = 1-F(r)$ as required. □

Lemma 3 implies that if ρ_r can be calculated before the coin $F(r)$ "is tossed", then all parties will accept only main-votes for ρ_r in round $r+1$ with probability

1/2 (the probability that $\rho_r = F(r)$). However, if ρ_r is not determined until after the coin is tossed, then the adversary may be able to set $\rho_r = 1 - F(r)$, and hence stop parties from agreeing. Note that, if ρ_r is undefined, then all parties will vote for the value of the coin in round $r + 1$ and agreement is reached.

Consider the state where the $(n - 2t)$th honest party is about to reveal its share of the coin $F(r)$ and suppose that $\mathcal{S}$ is the set of parties which have finished round r. All the parties in $\mathcal{S}$ have already collected main-votes for round r, and hence their pre-votes for round $r + 1$ are already determined. We have two cases to consider:

– There exists a party in $\mathcal{S}$ which is going to vote for a determined $v \in \{0, 1\}$ (not for the coin). Then this party has received at least one main-vote for v in round r. This means that ρ_r is determined and equals v, and hence the value of ρ_r is determined before the coin $F(r)$ is revealed.
– All parties in $\mathcal{S}$ will base their pre-vote in round $r + 1$ on the value of the coin $F(r)$. Now, since there are at least $n - 2t$ honest parties in $\mathcal{S}$, there will be at least $n - 2t$ pre-votes for $F(r)$ in round $r + 1$. In this case, the only possible value for ρ_{r+1} is $F(r)$, and therefore ρ_{r+1} is determined before the coin $F(r + 1)$ is revealed.

Therefore, the probability of agreement within two rounds from any $r > 1$ is at least 1/2, and hence the probability that an honest party advances by more than $2r + 1$ rounds is bounded above by 2^{-r}.

The proof of ***Fast Convergence*** clearly depends on the probability values, and hence cannot be verified using conventional model checking methods. However, from the proof we see that establishing this property reduces to analysing the probabilistic aspects of the protocol over two arbitrary rounds. That is, the complexity of having possibly unboundedly many rounds is removed from the verification. In Section 5.4 we describe how we formulate an abstraction by considering only two arbitrary rounds, verify the correctness of the abstraction, and prove ***Fast Convergence*** for finite configurations using the probabilistic model checker PRISM [54, 83]. In Section 5.5 we give an alternative proof of ***Fast Convergence*** for an arbitrary number of parties, which is automated except for one high-level inductive argument involving probabilistic reasoning.

5.4 Fast Convergence: Automated Verification

In this section we use PRISM, to verify ***Fast Convergence*** for finite configurations ($n = 4, \ldots, 20$). Based on the high level proof of ***Fast Convergence***, for a fixed round $r > 1$ we construct an abstract protocol considering *only* the main-votes for $r - 1$, all votes for round r and the pre-votes for $r + 1$. To initialise the abstract protocol we consider any possible combination of main-votes for round $r - 1$ which satisfies the condition: there cannot be a main-vote for 0 and a main-vote for 1. This restriction holds for the full protocol (see Lemma 1). Furthermore, we suppose that no party has decided in round r, that is, all parties take part in round r and round $r + 1$ (if a party has decided in an earlier round $r' < r$, then by ***Agreement*** it follows that all honest parties will decide by round $r' + 1 \leq r$). We only explicitly define the main-votes and pre-votes of the honest parties, and express the votes of the corrupted parties in terms of the honest parties votes using the assumptions we have made.

The times at which the coins of rounds $r-1$ and r are flipped, that is, when the $(n-t)$th share of the coin for rounds $r-1$ and r are released, are also abstracted. We suppose that this can happen any time after at least $n-2t$ honest parties have collected the main-votes for round $r-1$ that they require to cast their pre-vote in round r. The fact that this condition is sufficient for the coin in round $r-1$ follows from the fact that $n-t$ parties must give their share of the coin to work out the value of the coin, and honest parties do not work out their share until they have collected these main-votes from round $r-1$. Since clearly the coin in round r cannot be tossed before the coin in round $r-1$, requiring the same condition for the coin in round r is also sufficient. Note that, although this means that the coins for round $r-1$ and r may be tossed earlier than would be possible in the actual protocol, and hence give the adversary more power, these requirements are sufficient for proving ***Fast Convergence***.

Abstract Protocol. We now introduce the abstract protocol using the PRISM description language which is a variant of reactive modules [4]. The basic components of the language are *modules* and *variables*. A system is constructed as a number of modules which can interact with each other. A module contains a number of variables which express the state of the module, and its behaviour is given by a set of guarded commands of the form:

```
[] <guard> → <command>;
```

The guard is a predicate over the variables of the system and the command describes a transition which the module can make if the guard is true (using primed variables to denote the next values of variables). If a transition is probabilistic, then the command is specified as:

```
<prob> : <command> + ··· + <prob> : <command>
```

To construct the abstract protocol, we define a module for the adversary which decides on the main-votes in round $r-1$, modules for the coins in rounds $r-1$ and r, and modules for each honest party. The abstract protocol is then defined as the asynchronous parallel composition of these modules:

$$\textit{adversary} \;|||\; \textit{coin}_{r-1} \;|||\; \textit{coin}_r \;|||\; \textit{party}_1 \;|||\; \cdots \;|||\; \textit{party}_{n-t}.$$

We let $N = n-t$, the number of honest parties, and $M = n-2t$, the minimum number of main-votes (pre-votes) from honest parties required before a pre-vote (main-vote) can be made. Also, in the construction of the protocol, we define certain variables which can be updated by any module, which for example count the number of parties that have made a certain pre-vote. We achieve this by using *global* variables.

Module for the Adversary. The adversary decides on the main-votes in round $r-1$ and the only restriction we impose is that there cannot be votes for both

0 and 1. We suppose that there are always main-votes for abstain, and honest parties can decide on pre-votes after reading any. Instead boolean variables are used to denote whether there are main-votes for 0 and 1. The adversary has the following structure:

```
module adversary

    $m_0^{r-1}$ : [0..1]; // main-vote for 0 in round $r-1$
    $m_1^{r-1}$ : [0..1]; // main-vote for 1 in round $r-1$

    [] $(m_0^{r-1}{=}0) \wedge (m_1^{r-1}{=}0) \rightarrow (m_0^{r-1'}{=}1)$; // choose 0
    [] $(m_0^{r-1}{=}0) \wedge (m_1^{r-1}{=}0) \rightarrow (m_1^{r-1'}{=}1)$; // choose 1

endmodule
```

Note that, before the adversary makes this choice, we suppose there are only abstain votes in round $r-1$.

Modules for the Coins. The coin for round $r-1$ can be be tossed once $n-2t$ honest parties have decided on their pre-vote for round r and use the (global) variable n_r with range $[0..M]$ to count the number of parties who have decided on their pre-vote in round r (n_r is updated by honest parties when they have decided on their pre-vote for round r).

```
module $coin_{r-1}$

  $c_{r-1}$ : [0..1]; // local state of the coin (0 not tossed and 1 tossed)
  $v_{r-1}$ : [0..1]; // value of the coin

  // wait until $n_r \geq M$ before tossing the coin
  [] $(c_{r-1}{=}0) \wedge (n_r{\geq}M) \rightarrow 0.5 : (v'_{r-1}{=}0) \wedge (c'_{r-1}{=}1) + 0.5 : (v'_{r-1}{=}1) \wedge (c'_{r-1}{=}1)$;

endmodule
```

The coin for round r is similar and is constructed by renaming $coin_{r-1}$ as follows:

```
module $coin_r = coin_{r-1}[c_{r-1} = c_r, v_{r-1} = v_r]$ endmodule
```

Modules for the Parties. The local state of this party i is represented by the variable $s_i \in \{0, \ldots, 9\}$ with following interpretation:

0 - read main-votes in round $r-1$ and decide on a pre-vote for round r;
1 - cast pre-vote for 0 in round r;
2 - cast pre-vote for 1 in round r;
3 - cast pre-vote for coin in round r;
4 - read pre-votes and cast main-vote in round r;
5 - read main-votes in round r and decide on a pre-vote for round $r+1$;
6 - cast pre-vote for coin in round $r+1$;
7 - cast pre-vote for 0 in round $r+1$;

8 - cast pre-vote for 1 in round $r+1$;
9 - finished.

The module of party i has the form:

```
module $party_i$

  $s_i : [0..9]$; // local state of party i

  [] ...
  ⋮  ⋮

endmodule
```

where the transitions are dependent upon the local state of the party, what votes have been cast and the values of the coins. We now consider each state of the party in turn.

Read main-votes in round r–1 and decide on a pre-vote for round r: Recall that, before the adversary has chosen which of 0 and 1 is a possible main-vote for round $r-1$, there are only main votes for abstain, and hence the only pre-vote the party can have is for the coin. However, once the adversary has chosen between 0 and 1 there can be pre-votes for either this value or the coin (since we suppose there are always main-votes for abstain). Since we do not restrict the number of main-votes the party must read before choosing its pre-vote in round r, the transition rules for reading main-votes in round $r-1$ are given by:

```
[] $(s_i{=}0) \wedge (m_0^{r-1}{=}1) \rightarrow (s_i'{=}1) \wedge (n_r'{=}\min(M, n_r{+}1))$ // pre-vote for 0
[] $(s_i{=}0) \wedge (m_1^{r-1}{=}1) \rightarrow (s_i'{=}2) \wedge (n_r'{=}\min(M, n_r{+}1))$ // pre-vote for 1
[] $(s_i{=}0) \rightarrow (s_i'{=}3) \wedge (n_r'{=}\min(M, n_r{+}1))$ // pre-vote for coin
```

Note that, the party increments the variable n_r once it has finished reading main-votes in round $r-1$ and decided on a pre-vote for round r.

Cast pre-votes in round r: In this state a party has either already decided on its pre-vote, or it will be based on the value of the coin, and hence it must wait for the coin to be tossed. We introduce the variables (which will be needed for parties to decide on their main-votes in round r) p_v^r, for $v = 0, 1$, which count the number of pre-vote for v in round r. The transition rules of the party in this state are given by:

```
[] $(s_i{=}1) \rightarrow (s_i'{=}5) \rightarrow (p_0^{r'}{=}p_0^r{+}1)$; // cast pre-vote for 0
[] $(s_i{=}2) \rightarrow (s_i'{=}5) \rightarrow (p_1^{r'}{=}p_1^r{+}1)$; // cast pre-vote for 1
// cast pre-vote for the coin
[] $(s_i{=}3) \wedge (c_{r-1}{=}1) \wedge (v_{r-1}{=}0) \rightarrow (s_i'{=}5) \wedge (p_0^{r'}{=}p_0^r{+}1)$;
[] $(s_i{=}3) \wedge (c_{r-1}{=}1) \wedge (v_{r-1}{=}1) \rightarrow (s_i'{=}5) \wedge (p_1^{r'}{=}p_1^r{+}1)$;
```

Note that the global variables p_0^r and p_1^r are incremented when the party casts its vote.

Read pre-votes and cast main-vote in round r: A party must wait until sufficiently many $(n-t)$ pre-votes in round r have been cast, and hence until $n-2t$ honest parties have cast their pre-vote, that is, $p_0^r+p_1^r \geq M$. For the party to cast a main-vote for abstain, it must receive a pre-vote for 0 and for 1 which can be from either an honest or corrupted party. To receive an honest vote for $v \in \{0,1\}$, an honest party must have voted for this value, that is, $p_v^r > 0$. On the other hand, to receive a corrupted vote for v, either this is the value of the coin, or the corrupted party received a main-vote in the previous round for v, that is, $v_v = 0$ or $m_v^{r-1} = 1$. To cast a main-vote for $v \in \{0,1\}$, the party must at least have received at least $n-2t$ honest pre-votes for v. Before we give the transition rules we need the following boolean global variables: m_v^r for $v \in \{0,1,\mathit{abstain}\}$ to indicate what main-votes have been made. Note that, again, we only record if there is a main-vote for a value as opposed to the total number of votes. The transition rules are then given by:

$$
\begin{array}{l}
// \textit{ main-vote for abstain}\\
[]\ (s_i{=}4) \wedge (p_0^r{+}p_1^r \geq M) \wedge ((p_0^r > 0) \vee (v_1{=}0) \vee (m_0^{r-1}{=}1)) \wedge\\
\qquad\qquad ((p_1^r > 0) \vee (v_1{=}1) \vee (m_1^{r-1}{=}1)) \rightarrow (s_i'{=}5) \wedge (m_{abs}^{r'}{=}1)\\
// \textit{ main-vote for 0}\\
[]\ (s_i{=}4) \wedge (p_0^r{+}p_1^r \geq M) \wedge (p_0^r \geq M) \rightarrow (s_i'{=}5) \wedge (m_0^{r'}{=}1)\\
// \textit{ main-vote for 1}\\
[]\ (s_i{=}4) \wedge (p_0^r{+}p_1^r \geq M) \wedge (p_1^r \geq M) \rightarrow (s_i'{=}5) \wedge (m_1^{r'}{=}1)
\end{array}
$$

The global variables m_0^r, m_1^r and m_{abs}^r are updated when the party decides.

Read main-votes in round r and decide on a pre-vote for round $r+1$: To vote for the coin, the party must have received abstain votes from an honest party (again this is a requirement but is not sufficient in the actual protocol). To vote for $v \in \{0,1\}$ the party needs at least one vote for v from either an honest or corrupted party. To get such a vote from a corrupted party there needs to be at least $n-2t$ honest main-votes for v in round r. The transition rules for reading the main-votes are therefore given by:

$$
\begin{array}{l}
[]\ (s_i{=}5) \wedge (m_{abs}^r{=}1) \wedge (m_0^r{=}0) \wedge (m_1^r{=}0) \rightarrow (s_i'{=}6)\ // \textit{ pre-vote for coin}\\
[]\ (s_i{=}5) \wedge ((m_0^r{=}1) \vee (p_0^r \geq M)) \rightarrow (s_i'{=}7)\ // \textit{ pre-vote for 0}\\
[]\ (s_i{=}5) \wedge ((m_1^r{=}1) \vee (p_1^r \geq M)) \rightarrow (s_i'{=}8)\ // \textit{ pre-vote for 1}
\end{array}
$$

Cast pre-votes in round $r+1$: Since we are only concerned with finding whether all pre-votes are the same or not, we introduce just (global) boolean variables indicating that a pre-vote for this value has been cast or not, that is p_v^{r+1} for $v = 0,1$. The transition rules then follow similarly the cases for $s_i = 1,2,3$ above:

$$
\begin{array}{l}
[]\ (s_i{=}7) \wedge (c_r{=}1) \rightarrow (s_i'{=}9) \wedge (p_0^{r+1'}{=}1);\ // \textit{ cast pre-vote for 0}\\
[]\ (s_i{=}8) \wedge (c_r{=}1) \rightarrow (s_i'{=}9) \wedge (p_1^{r+1'}{=}1);\ // \textit{ cast pre-vote for 1}\\
// \textit{ cast pre-vote for the coin}\\
[]\ (s_i{=}6) \wedge (c_r{=}1) \wedge (v_r{=}0) \rightarrow (s_i'{=}9) \wedge (p_0^{r+1'}{=}1);\\
[]\ (s_i{=}6) \wedge (c_r{=}1) \wedge (v_r{=}1) \rightarrow (s_i'{=}9) \wedge (p_1^{r+1'}{=}1);
\end{array}
$$

This completes the possible transitions of party i. To construct further parties we use renaming, for example:

$$\texttt{module}\ \mathit{party}_j = \mathit{party}_i[s_i = s_j]\ \texttt{endmodule}$$

Correctness of the Abstract Protocol. To prove the correctness of the abstract model constructed in PRISM, we follow the method presented in [58] for timed probabilistic systems. This method reduces the verification of the correctness of the abstraction to constructing non-probabilistic variants of the abstract and concrete models and checking *trace refinement* between these systems. The method is reliant on encoding the probabilistic information and choices of the adversary in *actions* during model construction. Since the Cadence SMV language does not support actions, we use the process algebra CSP [88] and the model checker FDR [31].

More formally, we hand-translate both the abstract protocol and the full protocol (restricted to two arbitrary rounds) into CSP, encoding both the probabilistic choices and the possible non-deterministic choices of the adversary into the actions of the CSP processes. Using the tool FDR we were then able to show that the concrete protocol is a trace refinement of the abstract protocol, and hence the correctness of our abstraction. Note that we were only able to do this for finite configurations. For further details and the FDR code see the PRISM web page [83].

Model Checking Results. The property we wish to verify is that from the initial state, with probability at least 0.5, all honest parties have the same pre-vote in round $r + 1$, and hence decide by round $r + 1$. This property can be expressed by the PCTL formula:

$$\mathcal{P}_{\geq 0.5}\left[\texttt{true}\,\mathcal{U}\left(\bigwedge_{i=1}^{N} s_i{=}9\right) \wedge \left((p_0^{r+1}{=}1 \wedge p_1^{r+1}{=}0) \vee (p_0^{r+1}{=}0 \wedge p_1^{r+1}{=}1)\right)\right].$$

On all models constructed this property does indeed hold. A summary of the model checking results obtained for the abstract protocol in PRISM is included in Figure 4, where all experiments were run on a 440 MHz SUN Ultra 10 workstation with 512 Mb memory under the Solaris 2.7 operating system. Further details of the experiments can be found at the PRISM web page [83].

5.5 Fast Convergence: Parametric Verification

In this section we give a proof of ***Fast Convergence*** for any number of parties. The proof is fully automated except for one high-level inductive argument involving probabilistic reasoning. The proof demonstrates how to separate the probabilistic and nondeterministic behaviour and isolate the probabilistic arguments in the proof of correctness.

The high-level probabilistic argument is based on a number of properties (**P1** – **P6**) that can be proved by non-probabilistic reasoning, and have been proved

n	t	number of states	construction time (sec)	model checking time (sec)	**minimum probability**
4	1	16,468	3.00	1.49	**0.5**
5	1	99,772	5.41	4.86	**0.5**
6	1	567,632	8.53	7.81	**0.5**
7	2	1,303,136	10.9	16.0	**0.5**
8	2	8,197,138	20.0	24.4	**0.5**
9	2	5.002e+7	27.0	36.5	**0.5**
10	3	9.820e+7	33.8	58.9	**0.5**
11	3	6.403e+8	62.4	85.1	**0.5**
12	3	4.089e+9	75.4	114	**0.5**
13	4	7.247e+9	98.3	167	**0.5**
14	4	4.856e+10	157	282	**0.5**
15	4	3.199e+11	194	470	**0.5**
16	5	5.273e+11	241	651	**0.5**
17	5	3.605e+12	363	987	**0.5**
18	5	2.429e+13	610	1,318	**0.5**
19	6	3.792e+13	694	1,805	**0.5**
20	6	2.632e+14	1,079	2,726	**0.5**

Fig. 4. Model checking results for PRISM

in the Cadence SMV proof assistant for an arbitrary number of parties. Here we state them in English; for the corresponding formal statements and Cadence SMV proofs see [83]. First we proved that, for any party that enters round $r+1$ and does not decide in round r:

P1 If before the coin in round r is tossed there is a concrete pre-vote (i.e. a vote *not* based on the value of the coin) for v in round $r+1$ and after the coin in round r is tossed it equals v, then the party decides in round $r+1$.

P2 If before the coin in round r is tossed there are no concrete pre-votes in round $r+1$, then either the party decides in round $r+1$, or if after the coins in round r and round $r+1$ are tossed they are equal, then the party decides in round $r+2$.

In addition, we proved that the following properties hold.

P3 If the coin in round r has not been tossed, then neither has the coin in round $r+1$.

P4 In any round r there cannot be concrete pre-votes for 0 and 1.

P5 In any round r, if there is a concrete pre-vote for $v \in \{0,1\}$, then in all future states there is a concrete pre-vote for v.

P6 Each coin is only tossed once.

We complete the proof of ***Fast Convergence*** with a simple manual proof based on the following classification of protocol states:

– let *Undec*(r) be the set of states in which the coin in round $r-1$ is not tossed and there are no concrete pre-votes in round r;

- for $v \in \{0,1\}$, let *Pre-vote*(r, v) be the set of states where the coin in round $r-1$ is not tossed and there is a concrete pre-vote for v in round r.

It follows from **P4** that these sets are pairwise disjoint and *any* state where the coin in round $r-1$ is not tossed is a member of one of these sets. The following proposition is crucial to establishing the efficiency of the protocol.

Proposition 2. *In an idealised system, where the values of the coins in rounds* $1, 2, \ldots, 2r-1$ *are truly random, the probability of a party advancing by more than* $2r+1$ *rounds is bounded above by* 2^{-r}.

Proof. We prove the proposition by induction on $r \in \mathbb{N}$. The case when $r = 0$ is trivial since the probability bound is 1. Now suppose that the proposition holds for some $r \in \mathbb{N}$ and suppose a party enters round $2r+1$. If a party decides in round $2r$, then by ***Agreement*** all parties will decide by round $2r+1$, and hence the probability that a party enters round $2r+3$ given a party enters round $2r+1$ is bounded above by 0. On the other hand, if no party decides in round $2r$, then by Proposition 1(a) the coin for round $2r$ is tossed. For any state s reached just before the coin is tossed we have two cases to consider:

- $s \in$ *Pre-vote*$(2r+1, v)$ for some $v \in \{0,1\}$: by **P1**, if the coin in round $2r$ equals v, any party which enters round $2r+1$ decides in round $2r+1$, and hence using **P6** it follows that the probability of a party advancing more than $2r+3$ rounds given that a party advances more than $2r+1$ rounds is bounded above by 1/2.
- $s \in$ *Undec*$(2r+1)$: using **P5**, there are no concrete pre-votes in round $2r+1$ before the coin in round $2r+1$ is tossed, and hence by **P2** any party either decides in round $2r+1$, or, if the coins in round $2r$ and $2r+1$ are equal, it decides in round $2r+2$. Now, since in s the coin for round $2r$ has not been tossed, by **P3** neither has the coin for round $2r+1$. Therefore, using Proposition 1 and **P6** it follows that the probability of a party advancing more than $2r+3$ rounds given that a party advances more than $2r+1$ rounds is bounded above by 1/2.

Putting this together and since $\mathbf{P}(A \cap B) = \mathbf{P}(A|B) \cdot \mathbf{P}(B)$, we have

$$\begin{aligned}
&\mathbf{P}(\text{a party advances} > 2r{+}3 \text{ rounds}) \\
&\quad = \mathbf{P}(\text{a party advances} > 2r{+}3 \text{ rounds and a party advances} > 2r{+}1 \text{ rounds}) \\
&\quad = \mathbf{P}(\text{a party advances} > 2r{+}3 \text{ rounds} \mid \text{a party advances} > 2r{+}1 \text{ rounds}) \cdot \\
&\quad\quad \mathbf{P}(\text{a party advances} > 2r{+}1 \text{ rounds}) \\
&\quad \leq 1/2 \cdot 2^{-r} = 2^{-(r+1)}
\end{aligned}$$

as required. □

It can be argued that in a real system the probability of a party advancing by more than $2r+1$ rounds is bounded above by $2^{-r} + \varepsilon$, where ε is negligible. This follows from the **Unpredictability** property of the coin tossing scheme and **P6**; for more details see [10].

In addition to ***Fast Convergence*** we can directly prove that the protocol guarantees ***Probabilistic Termination*** in a constant expected number of rounds by using probabilistic complexity statements. As in [82], the complexity measure of interest corresponds to the increase in the maximum round number among all the parties. We now sketch the argument for proving that the protocol guarantees ***Probabilistic Termination*** in a constant expected number of rounds using the probabilistic complexity statements. First, let $\phi_{MaxRound}$ be the complexity measure that corresponds to the increase in the maximum round number among all the parties, and define the following sets of states:

– $\mathcal{R}$, the set of reachable states of the protocol;
– $\mathcal{D}$, the set of reachable states of the protocol in which all parties have decided;
– *Undec*, the set of states in which the coin in round $r_{\max} - 1$ is not tossed and there are no concrete pre-votes in round $r_{\max}$, where $r_{\max}$ is the current maximum round number among all parties;
– *Pre-vote*(v), the set of states where the coin in round $r_{\max} - 1$ is not tossed and there is a concrete pre-vote for v in round $r_{\max}$, where $r_{\max}$ is the current maximum round number among all parties.

Next we require the following property (which is straightforward to prove in Cadence SMV): from any state (under any fair scheduling of the non-determinism) the maximum round increases by at most one before we reach a state where either all parties have decided or the coin in the maximum round has not been tossed, which can be expressed as the following probabilistic complexity statement:

$$\mathcal{R} \xrightarrow{\phi_{MaxRound} \leq 1}_{1} \mathcal{D} \cup \mathit{Undec} \cup \mathit{Pre\text{-}vote}(0) \cup \mathit{Pre\text{-}vote}(1) .$$

Applying similar arguments to those given in Proposition 2 we can show that the following probabilistic complexity statements hold:

$$\mathit{Undec} \xrightarrow{\phi_{MaxRound} \leq 2}_{\frac{1}{2}} \mathcal{D} \quad \text{and} \quad \mathit{Pre\text{-}vote}(v) \xrightarrow{\phi_{MaxRound} \leq 2}_{\frac{1}{2}} \mathcal{D} \quad \text{for } v \in \{0, 1\}.$$

Alternatively, one could use *coin lemmas* [92, 34] to validate these probabilistic complexity statements. Then, using the compositionality result of complexity statements [91] and the fact that the sets *Undec*, *Pre-vote*(0) and *Pre-vote*(1) are disjoint, the above complexity statements can be combined to give:

$$\mathcal{R} \xrightarrow{\phi_{MaxRound} \leq 2+1}_{1 \cdot \frac{1}{2}} \mathcal{D} ,$$

that is, from any state of the protocol the probability of reaching a state where all parties have decided while the maximum round increases by at most 3 is at least 1/2. Finally, again using results presented in [91], it follows that from any state of the protocol all parties decide within at most $O(1)$ rounds.

6 Discussion and Conclusions

We have presented a number of techniques that can be applied to the analysis of randomized distributed algorithms. The main problem in verifying such algorithms is correctly dealing with the interplay between probability and nondeterminism. A number of approaches exist however, the lesson to be learnt when dealing with complex randomized distributed algorithms is to first try and separate the probabilistic reasoning to a small isolated part of the protocol. This then simplifies the probabilistic arguments and allows one to use standard non-probabilistic techniques in the majority of the verification.

The differing techniques have been used to verify a number of different randomized distributed algorithms and it can be seen that both the structure of the protocol under study and the type of property being verified has influence over the applicability of each approaches and how easy it is to apply.

It may be beneficial to consider new methods for verification for example based on the techniques developed in the areas of performance analysis and the extensive literature concerning dynamic programming and Markov Decision Processes. In particular, it would be useful to examine how methods using *rewards* can be incorporated in these approaches, for example to compute the expected number of rounds.

With regards to computer-aided verification, state-of-the-art probabilistic model checking tools are applicable to only complete, finite state models. On the other hand, there exist non-probabilistic model checkers which can deal with parametric and infinite state programs, however they do not support probabilistic reasoning. A fully automated proof of correctness of randomized distributed algorithms could feasibly be derived using a theorem prover e.g. [43]. An alternative goal is to develop proof techniques for probabilistic systems in the style of for example Cadence SMV, incorporating both those used in Cadence SMV and the proof rules for probabilistic complexity statements following [91]. Given an implementation of such rules as a layer on top of, for example, the PRISM model checking tool, one may be able to *fully automate* the proof of correctness of complex randomized distributed algorithms.

References

1. S. Aggarwal and S. Kutten. Time-optimal self stabilizing spanning tree algorithms. In R. Shyamasundar, editor, *Proc. Foundations of Software Technology and Theoretical Computer Science*, volume 761 of *LNCS*, pages 15–17. Springer, 1993.
2. R. Alur, C. Courcoubetis, and D. Dill. Model-checking for probabilistic real-time systems. In J. Albert, B. Monien, and M. Rodrf8guez-Artalejo, editors, *Proc. Int. Col. Automata, Languages and Programming (ICALP'91)*, volume 510 of *LNCS*, pages 115–136. Springer, 1991.
3. R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.
4. R. Alur and T. Henzinger. Reactive modules. *Formal Methods in System Design*, 15:7–48, 1999.
5. J. Aspnes and M. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 11(3):441–460, 1990.

6. C. Baier, M. Huth, M. Kwiatkowska, and M. Ryan, editors. *Proc. Int. Workshop Probabilistic Methods in Verification (PROBMIV'98)*, volume 22 of *ENTCS*. Elsevier Science, 1998.
7. C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11:125–155, 1998.
8. M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proc. Symp. Principles of Distributed Computing (PODC'83)*, pages 27–30. ACM Press, 1983.
9. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. Thiagarajan, editor, *Proc. Foundations of Software Technology and Theoretical Computer Science (FSTTCS'95)*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.
10. C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols (extended abstract). In J. Kilian, editor, *Proc. Advances in Cryptology - CRYPTO 2001*, volume 2139 of *LNCS*, pages 524–541. Springer, 2001.
11. C. Cachin, K. Kursawe, and V. Shoup. Random oracles in Constantinople: practical asynchronous Byzantine agreement using cryptography (extended abstract). In *Proc. Symp. Principles of Distributed Computing (PODC'00)*, pages 123–132. ACM Press, 2000.
12. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
13. R. Cleaveland, S. Smolka, and A. Zwarico. Testing preorders for probabilistic processes. In W. Kuich, editor, *Proc. Int. Col. Automata, Languages and Programming (ICALP'92)*, volume 623 of *LNCS*, pages 708–719. Springer, 1992.
14. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.
15. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. Symp. Principles of Programming Languages (POPL'77)*, pages 238–252. ACM Press, 1977.
16. S. Creese and A. Roscoe. Data independent induction over structured networks. In H. Arabnia, editor, *Proc. Int. Conf. Parallel and Distributed Processing Techniques and Applications (PDPTA'00)*, volume II. CSREA Press, 2000.
17. P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In de Alfaro and Gilmore [24], pages 39–56.
18. P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reduction and refinement strategies for probabilistic anylsis. In Hermanns and Segala [42], pages 57–76.
19. C. Daws, M. Kwiatkowska, and G. Norman. Automatic verification of the IEEE 1394 root contention protocol with KRONOS and PRISM. In *Proc. Int. Workshop Formal Methods for Industrial Critical Systems (FMICS'02)*, volume 66(2) of *ENTCS*. Elsevier Science, 2002.
20. L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.
21. L. de Alfaro. Temporal logics for the specification of performance and reliability. In R. Reischuk and M. Morvan, editors, *Proc. Symp. Theoretical Aspects of Computer Science (STACS'97)*, volume 1200 of *LNCS*, pages 165–176. Springer, 1997.
22. L. de Alfaro. From fairness to chance. In Baier et al. [6].

23. L. de Alfaro. How to specify and verify the long-run average behaviour of probabilistic systems. In *Proc. Symp. Logic in Computer Science (LICS'98)*, pages 454–465. IEEE Computer Society Press, 1998.
24. L. de Alfaro and S. Gilmore, editors. *Proc. Int. Workshop Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'01)*, volume 2165 of *LNCS*. Springer, 2001.
25. L. de Alfaro, T. Henzinger, and R. Jhala. Compositional methods for probabilistic systems. In Larsen and Nielsen [60], pages 351–365.
26. C. Derman. *Finite-State Markovian Decision Processes.* New York: Academic Press, 1970.
27. E. Dijkstra. *A Discipine of Programming.* Prenticec Hall International, 1976.
28. S. Dolev, A. Israeli, and S. Moran. Analyzing expected time by scheduler-luck games. *IEEE Transactions on Software Engineering*, 21(5):429–439, 1995.
29. M. Duflot, L. Fribourg, and C. Picaronny. Randomized finite-state distributed algorithms as Markov chains. In J. Welch, editor, *Proc. Distributed Computing (DISC'2001)*, volume 2180 of *LNCS*, pages 240–254. Springer, 2001.
30. M. Duflot, L. Fribourg, and C. Picaronny. Randomized dining philosophers without fairness assumption. In *Proc. IFIP Int. Conf. Theoretical Computer Science (TCS'02)*, volume 223 of *IFIP Conference Proceedings*, pages 169–180. Kluwer Academic, 2002.
31. Failures divergence refinement (FDR2). Formal Systems (Europe) Limited, http://www.formal.demon.co.uk/FDR2.html.
32. W. Feller. *An Introduction to Probability Theory and its Applications*, volume 1. John Wiley & Sons, 1950.
33. M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(5):374–382, 1985.
34. K. Folegati and R. Segala. Coin lemmas with random variables. In de Alfaro and Gilmore [24], pages 71–86.
35. H. Hansson. *Time and Probability in Formal Design of Distributed Systems.* Elsevier, 1994.
36. H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proc. Real-Time Systems Symposium*, pages 278–287. IEEE Computer Society Press, 1990.
37. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(4):512–535, 1994.
38. S. Hart, M. Sharir, and A. Pnueli. Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages and Systems*, 5(3):356–380, 1983. A preliminary version appeared in Proc. ACM Symp. Principles of Programming Languages, pages 1–6, 1982.
39. T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Proc. Symp. Logic in Computer Science (LICS'98)*, pages 394–406. IEEE Computer Society Press, 1992.
40. T. Henzinger, S. Qadeer, and S. Rajamani. You assume, we guarantee: Methodology and case studies. In A. Hu and M. Vardi, editors, *Proc. Computer-aided Verification (CAV'98)*, volume 1427 of *LNCS*, pages 440–451. Springer, 1998.
41. T. Herman. Probabilistic self stabilisation. *Information Processing Letters*, 35(2):63–67, 1990.
42. H. Hermanns and R. Segala, editors. *Proc. Int. Workshop Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02)*, volume 2399 of *LNCS*. Springer, 2002.

43. Joe Hurd. Verification of the Miller-Rabin probabilistic primality test. In R. Boulton and P. Jackson, editors, *TPHOLs 2001: Supplemental Proceedings*, number EDI-INF-RR-0046 in Informatics Report Series, pages 223–238. Division of Informatics, University of Edinburgh, 2001.
44. Joe Hurd. *Formal Verification of Probabilistic Algorithms.* PhD thesis, University of Cambridge, 2002.
45. Michael Huth and Marta Kwiatkowska. Quantitative analysis and model checking. In *Proc. Symp. Logic in Computer Science (LICS'97)*, pages 111–122. IEEE Computer Society Press, 1997.
46. IEEE Computer Society. IEEE Standard for a High Performance Serial Bus. Std 1394–1995. August 1996.
47. A. Israeli and M. Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *Proc. Symp. Principles of Distributed Computing (PODC'90)*, pages 119–131. ACM Press, 1990.
48. B. Jonnsson and K.G. Larsen. Specification and refinement of probabilistic processes. In *Proc. Symp. Logic in Computer Science (LICS'91)*, pages 266–277. IEEE Computer Society Press, 1991.
49. B. Jonsson and J. Parrow, editors. *Proc. Int. Conf. Concurrency Theory (CONCUR'94)*, volume 836 of *LNCS*. Springer, 1994.
50. Y. Kesten and A. Pnueli. Control and data abstraction: the cornerstone of practical formal verification. *Software Tools for Technology Transfer*, 4(2):328–342, 2000.
51. Y. Kesten, A. Pnueli, E. Shahar, and L. Zuck. Network invariants in action. In L. Brim, P. Jancar, M. Kretinsky, and A. Kucera, editors, *Proc. CONCUR'02 – Concurrency Theory*, volume 2421 of *LNCS*, pages 101–115. Springer, 2002.
52. E. Kushilevitz and M. Rabin. Randomized mutual exclusion algorithm revisited. In PODC92 [80], pages 275–284.
53. M. Kwiatkowska and G. Norman. Verifying randomized Byzantine agreement. In D. Peled and M. Vardi, editors, *Proc. Formal Techniques for Networked and Distributed Systems (FORTE'02)*, volume 2529 of *LNCS*, pages 194–209. Springer, 2002.
54. M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In T. Field, P. Harrison, J. Bradley, and U. Harder, editors, *Proc. Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, volume 2324 of *LNCS*, pages 200–204. Springer, April 2002.
55. M. Kwiatkowska, G. Norman, and R. Segala. Automated verification of a randomized distributed consensus protocol using Cadence SMV and PRISM. In G. Berry, H. Comon, and A. Finkel, editors, *Proc. Int. Conf. Computer Aided Verification (CAV'01)*, volume 2102 of *LNCS*, pages 194–206. Springer, 2001.
56. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282:101–150, 2002.
57. M. Kwiatkowska, G. Norman, and J. Sproston. Symbolic computation of maximal probabilistic reachability. In Larsen and Nielsen [60], pages 169–183.
58. M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. *Special Issue of Formal Aspects of Computing*, 2002. To appear.
59. M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In Hermanns and Segala [42], pages 169–187.

60. K. Larsen and M. Nielsen, editors. *Proc. CONCUR'01: Concurrency Theory*, volume 2154 of *LNCS*. Springer, 2001.
61. K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991. Preliminary version of this paper appeared in Proc. 16th Annual ACM Symposium on Principles of Programming Languages, pages 134-352, 1989.
62. R. Lassaigne and S. Peyronnet. Approximate verification of probabilistic systems. In Hermanns and Segala [42], pages 213–214.
63. D. Lehmann and M. Rabin. On the advantage of free choice: A symmetric and fully distributed solution to the dining philosophers problem (extended abstract). In *Proc. Symp. on Principles of Programming Languages (POPL'81)*, pages 133–138. ACM Press, 1981.
64. G. Lowe. *Probabilities and Priorities in Timed CSP.* PhD thesis, Oxford University Computing Laboratory, 1993.
65. G. Lowe. Representing nondeterministic and probabilistic behaviour in reactive processes. Technical Report PRG-TR-11-93, Oxford University Computing Laboratory, 1993.
66. N. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.
67. N. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *Proc. Symp. Principles of Distributed Computing (PODC'94)*, pages 314–323. ACM Press, 1994.
68. A. McIver. Quantitative program logic and expected time bounds in probabilistic distributed algorithms. *Theoretical Computer Science*, 282:191–219, 2002.
69. A. McIver and C. Morgan. An expectation-based model for probabilistic temporal logic. *Logic Journal of the IGPL*, 7(6):779–804, 1999.
70. A. McIver, C. Morgan, and E. Troubitsyna. The probabilistic steam boiler: a case study in probabilistic data refinement. In J. Grundy, M. Schwenke, and T. Vickers, editors, *Proc. Int. Refinement Workshop and Formal Methods Pacific 1998*, Discrete Mathematics and Theoretical Computer Science, pages 250–265. Springer, 1998.
71. K. McMillan. Verification of infinite state systems by compositional model checking. In L. Pierre and T. Kropf, editors, *Proc. Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'99)*, volume 1703 of *LNCS*, pages 219–233. Springer, 1999.
72. K. McMillan. A methodology for hardware verification using compositional model checking. *Science of Computer Programming*, 37(1–3):279–309, 2000.
73. R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.
74. C. Morgan and A. McIver. *pGL*: Formal reasoning for randomized distributed algorithms. *South African Computer Journal*, pages 14–27, 1999.
75. C. Morgan, A. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, 1996.
76. C. Morgan, A. McIver, K. Seidel, and J. Sanders. Refinement-oriented probability for CSP. *Formal Aspects of Computing*, 8(6):617–647, 1996.
77. R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.
78. A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.
79. A. Pnueli and L. Zuck. Probabilistic verification. *Information and Computation*, 103:1–29, 1993.
80. *Proc. Symp. Principles of Distributed Computing (PODC'92).* ACM Press, 1992.

81. A. Pogosyants and R. Segala. Formal verification of timed properties of randomized distributed algorithms. In *Proc. Symp. Principles of Distributed Computing (PODC'95)*, pages 174–183. ACM Press, 1995.
82. A. Pogosyants, R. Segala, and N. Lynch. Verification of the randomized consensus algorithm of Aspnes and Herlihy: a case study. *Distributed Computing*, 13(3):155–186, 2000.
83. PRISM web page. http://www.cs.bham.ac.uk/~dxp/prism/.
84. M. Rabin. N-process mutual exclusion with bounded waiting by $4 \log_2 N$-valued shared variable. *Journal of Computer and System Sciences*, 25(1):66–75, 1982.
85. M. O. Rabin. Probabilistic algorithms. In J. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 21–39. Academic Press, New York, 1976.
86. RAPTURE web page. http://www.irisa.fr/prive/bjeannet/prob/prob.html.
87. M. Reiter and A. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security (TISSEC)*, 1(1):66–92, 1998.
88. A. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall, 1997.
89. A. Saias. *Randomness versus Non-Determinism in Distributed Computing.* PhD thesis, Massachusetts Institute of Technology, 1994.
90. I. Saias. Proving probabilistic correctness statements: the case of Rabin's algorithm for mutual exclusion. In PODC92 [80], pages 263–274.
91. R. Segala. *Modelling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, Massachusetts Institute of Technology, 1995.
92. R. Segala. The essence of coin lemmas. In Baier et al. [6].
93. R. Segala. Verification of randomized distributed algorithms. In E. Brinksma, H. Hermanns, and J.-P. Katoen, editors, *Lectures on Formal Methods and Performance Analysis (First EEF/Euro Summer School on Trends in Computer Science)*, volume 2090 of *LNCS*, pages 232–260. Springer, 2001.
94. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In Jonsson and Parrow [49], pages 481–496.
95. V. Shmatikov. Probabilistic analysis of anonymity. In *Proc. Computer Security Foundations Workshop (CSFW'02)*, pages 119–128. IEEE Computer Society Press, 2002.
96. V. Shoup. Practical threshold signatures. In B. Preneel, editor, *Proc. Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220. Springer, 2000.
97. T. Speed. Probabilistic risk assessment in the nuclear industry : WASH-1400 and beyond. In L. LeCam and R. Olshen, editors, *Proc. Berkeley Conference in honour of Jerzy Neyman and Jack Kiefer.* Wadsworth Inc., 1985.
98. E. Stark and G. Pemmasani. Implementation of a compositional performance analysis algorithm for probabilistic I/O automata. In J. Hillston and M. Silva, editors, *Proc. Int. Workshop Process Algebra and Performance Modelling (PAPM'99)*, pages 3–24. Prensas Universitarias de Zaragoza, 1999.
99. E. Stark and S. Smolka. Compositional analysis of expected delays in networks of probabilistic I/O automata. In *Proc. Symp. Logic in Computer Science (LICS'98)*, pages 466–477. IEEE Computer Society Press, 1988.
100. M. Stoelinga and F. Vaandrager. Root contention in IEEE 1394. In J.-P. Katoen, editor, *Proc. AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'99)*, volume 1601 of *LNCS*, pages 53–74. Springer, 1999.
101. M. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proc. Symp. Foundations of Computer Science (FOCS'85)*, pages 327–338. IEEE Computer Society Press, 1985.

102. M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. Symp. Logic in Computer Science (LICS'86)*, pages 332–344. IEEE Computer Society Press, 1986.
103. S-H. Wu, S.A. Smolka, and E.W. Stark. Composition and behaviour of probabilistic I/O automata. In Jonsson and Parrow [49], pages 513–528.
104. Wang Yi and K.G. Larsen. Testing probabilistic and non-deterministic processes. In R. Linn Jr. and M. Ümit Uyar, editors, *Protocol Specification, Testing and Verification*, volume C-8 of *IFIP Transactions*, pages 47–61. North-Holland, 1992.
105. L. Zuck, A. Pnueli, and Y. Kesten. Automatic verification of probabilistic free choice. In A. Cortesi, editor, *Proc. Verification, Model Checking, and Abstract Interpretation, Third International Workshop (VMCAI 2002)*, volume 2294 of *LNCS*, pages 208–224. Springer, 2002.