# Verification of Randomized Distributed Algorithms

Roberto Segala

Department of Computer Science
University of Bologna

**Abstract.** We describe modular verification techniques for randomized distributed algorithms as extensions of techniques for ordinary, non randomized, distributed algorithms. The main difficulty to overcome arises from the subtle interplay between probability and nondeterminism, where probability is due to the random choices that occur within an algorithm, and nondeterminism is due to the unknown speeds and scheduling policies of the processes. The techniques that we introduce are based on separation of probability from nondeterminism.
When the nondeterminism is factored out, the analysis of an algorithm has several pieces that are in common with the area of performance evaluation. Thus, the techniques that we describe are likely to constitute a bridge to export typical performance evaluation techniques to the area of concurrent nondeterministic systems and, vice versa, to understand alternative ways for handling nondeterminism when it arises.

## 1 Introduction

Since the pioneering paper of Rabin [81], randomization has turned out to be a fundamental tool for the efficient solution of problems, and in particular, in the field of distributed computation, a fundamental tool for the solution of problems that are otherwise unsolvable. Examples are algorithms for mutual exclusion with few shared variables [82, 57], algorithms for consensus [21, 12, 13], shown otherwise unsolvable in [38], and algorithms for symmetric solution to the dining philosophers problem [65].

Unfortunately, randomization is very complicated to deal with since it is combined with the intrinsic nondeterminism of concurrent systems that derives from the unknown scheduling policies and relative speeds of the processes. As opposed to the standard approach in performance analysis, we cannot simply assume some approximate form of resolution of the nondeterminism; rather we are interested in the worst case scenario. Thus, our main interest is to study the functional correctness and performance of an algorithm under any resolution of the nondeterminism, possibly with some fairness restrictions.

The main difficulty in the analysis of a randomized distributed algorithm is that, although the designer of an algorithm may believe that a specific stochastic process takes place during execution, the resolution of the nondeterminism may affect such stochastic process at the point of compromising its outcome. Several times the resolutor of the nondeterminism, which is usually called an *adversary*, has the ability not to schedule some of the elementary experiments that are part of a stochastic process, thus creating dependencies between events that are supposed to be independent. The result is that several algorithms proposed in the literature turned out to be incorrect. As an example, the original algorithm for mutual exclusion of Rabin [82] was shown to have problems in [84] and later fixed in [57].

Our objective is to study techniques for the analysis of randomized distributed algorithms, possibly extending known techniques for the analysis of ordinary, non randomized, distributed systems. In particular, we extend labeled transition systems [56] to account for randomization, and we extend the typical modular verification techniques based on analysis of computations [68] and simulation relations [71, 51, 69]. Modular analysis is important for scalability issues, so that the analysis of a large system can be decomposed into the analysis of several smaller systems.

The probabilistic extension of labeled transition systems that we obtain, which we call *probabilistic automata*, turn out to be an extension of Markov Decision Processes [32], a model used for the study of optimal control *policies* in a randomized environment; furthermore, once the nondeterminism is resolved, the objects that we obtain are infinite state Markov Processes, or more precisely semi-Markov Processes, and these semi-Markov processes, which we call *probabilistic executions*, are the objects on which functional correctness and performance is studied.

Large probabilistic automata are built by composing smaller probabilistic automata and letting them interact together by means of a mechanism that resembles the product of ordinary automata. An important result (cf. Theorem 1) is that each probabilistic execution of the composition of two probabilistic automata can be seen as the result of two probabilistic executions of the two components, respectively. This means that a property of the composition of two probabilistic automata can be derived from the properties of the components, i.e., it is possible to reason in a modular way.

We introduce two abstract verification techniques based on the extension of the idea of language inclusion [49], also called *trace inclusion*, and of simulation relation [51, 69]. Both techniques induce a preorder relation on probabilistic automata that is preserved by composition, thus leading again to modularity. Roughly speaking, the language of a probabilistic automaton describes all the possible communication patterns that the automaton can engage in. Thus, if the language of a probabilistic automaton $\mathcal{A}$ is a subset of the language of another probabilistic automaton $\mathcal{B}$, then $\mathcal{A}$ cannot engage in any communication pattern that $\mathcal{B}$ cannot engage in, and because of compositionality, $\mathcal{A}$ cannot engage in any communication pattern that $\mathcal{B}$ cannot engage in independently of the context in which $\mathcal{A}$ and $\mathcal{B}$ are embedded. Suppose now that $\mathcal{B}$ always responds to the requests it receives. Then there are no traces in the language of $\mathcal{B}$ where a request does not have any response. Because of trace inclusion, there are no traces in the language of $\mathcal{A}$ where a request does not have any response, and thus $\mathcal{A}$ responds to all the requests it receives. Language inclusion can be used as an implementation relation. Simulation relations are used as a sound proof technique for language inclusion.

We also describe two techniques for the separation of probability from nondeterminism, since it is the interaction between probability and nondeterminism that causes most of the problems. The first technique consists of a collection of tools, called *coin lemmas*, that provide us with a rule to associate an event with each probabilistic execution of a probabilistic automaton and a minimum probability $p$ that each event is guaranteed to have. Thus it is sufficient to show that all the elements in the events given by the rule ensure the correct termination of the algorithm under examination to conclude that the algorithm terminates correctly with probability at least $p$. The advantage of coin lemmas is that they take care explicitly of the interaction between probability and nondeterminism (cf. Sections 3 and 5), and reduce the analysis of a probabilistic property to the analysis of a property that involves only nondeterminism.

The second technique for the separation of probability and nondeterminism originates from arguments that are typical of random variables. Consider a probabilistic execution (a semi-Markov process), and consider a function $\phi$ that assigns a non-negative real number with each path (which we call an execution) in the probabilistic execution. We call $\phi$ a *complexity function*. Consider a reachability property that holds with probability 1. Function $\phi$ applied to the minimum paths where the reachability property is satisfied is a random variable that expresses the complexity required to satisfy the reachability property. Furthermore, the expected value of $\phi$ is the expected complexity to satisfy the property. In distributed algorithms it is typical to study the relationship between different complexity measures to derive the complexity of the algorithm under examination. Such relations can be lifted to expectations using the results that hold for random variables. Once again, results about expectations are proved by analyzing properties that involve only nondeterminism.

Throughout the paper we give highlights of a large case study [79] where all the techniques that we present are used. We do not work out all the details of the example and we refer the interested reader to the paper that documents the whole case study. Other simpler case studies appear in [2, 67, 78, 86].

The fact that in our analysis we have to deal with objects that are similar to Markov Decision Processes and with Markov processes in general suggests that the analysis of randomized distributed algorithms has several elements in common with classical performance analysis. It is our hope that the experience we have gained can constitute one of the bridges for exchanging techniques between concurrency theory, specifically the study of nondeterminism in concurrency theory, and performance analysis. Our presentation is focused on algorithms; yet people familiar with other problems that include randomization and that need an explicit treatment of nondeterminism may find useful suggestions.

The rest of the paper is structured as follows: Section 2 gives an overview of related work in the areas of probabilistic models with nondeterminism and of analysis of randomized distributed algorithms; Section 3 illustrates some examples of typical problems that arise in the analysis of randomized distributed algorithms; Section 4 introduces probabilistic automata and the related modular verification techniques; Section 5 introduces coin lemmas; Section 6 shows how to apply the techniques of this paper by giving highlights on a specific example that is studied in more detail in [79]; Section 7 studies techniques for performance analysis, and Section 8 shows how to analyze the performance of the algorithm of Section 6; finally, Section 9 gives some concluding remarks.

## 2 Related Work

The study of probability in concurrency theory dates back to the 80's [62], where the nondeterministic + operator was replaced by a probabilistic + operator. Since then, several models were proposed, and a first classification attempt came in [41, 40], where probabilistic processes were classified into reactive, generative, and stratified. Typical reactive models are in [62, 24, 63, 64, 27] and typical generative models are in [39, 54, 92, 14, 31, 11]. A generative process is like a Markov process and does not include any nondeterminism, while a reactive process includes some form of nondeterminism. A mixed model was introduced in [94, 95]. Denotational models for probabilistic systems were studied in [50, 91]. Models with real nondeterminism and probability were introduced in [45, 46, 96, 89].

Several concepts from concurrency theory are extended to the probabilistic case. Among these are simulation relations [52, 89], trace based semantics [39, 86], and testing based semantics [26, 96, 27, 73, 87]. In the area of probabilistic bisimulation there is a lot of interest in decision algorithms [18, 75, 20] and in variants of bisimulation that lead to more efficient algorithms [20].

Other studies that combine concurrency theory and performance analysis are stochastic process algebras [48, 42, 22]. The reader interested in this area may find more specific information in this volume [25, 47]. Typically stochastic process algebras include little nondeterminism and are more alike generative processes.

Nondeterminism was studied also in the area of model checking [36], where the validity of a formula expressed in some temporal logic is verified mechanically on a model expressed in terms of automata. The pioneering paper was in 1985 [93], where a simple model checking algorithm for *Concurrent Markov Chains* was introduced. Concurrent Markov Chains are very much alike probabilistic automata if we do not dive too deeply into semantic issues. Other model checking algorithms for properties that hold with probability 0 or 1 are studied in [29, 30], and extensions with real time are studied in [9, 10]. The first model checking algorithm for properties that hold with any probability is presented in [43, 44], and more efficient algorithms are studied in [23]. Fairness is studied in [19], and symbolic model checking algorithms are studied in [17, 8]. Quantitative properties in the presence of real time are studied in [59, 60].

Other relevant work in the analysis of systems with probability and nondeterminism is in [4, 6, 5, 7]. Here, some automatic techniques for the analysis of long average behavior properties of systems are introduced. The models and techniques are inspired by former studies on Markov Decision Processes [32], a model that in essence coincides with reactive processes and that was introduced to study optimal control policies for systems that include randomization.

Finally, objects like probabilistic automata were introduced long ago in [80] for the study of generalizations of the class of regular languages.

In the area of verification of randomized distributed algorithms there is work on extending temporal logic based techniques in [76, 77, 83, 70]. These techniques can be used to study events that hold either with probability 1 or probability 0 and are based on transforming the 0/1 probability of an event into a topological property of the underlying transition system. The main underlying concepts are called extreme fairness and $\alpha$-fairness. The same techniques were used for the study of performance [76], but the results are not very tight.

Other logic-based techniques are introduced in [70], where the reasoning style of probabilistic predicate transformers [72] is applied to the analysis of a distributed algorithm. Predicate transformers are a generalization of the weakest precondition style of reasoning; their semantics is a real number rather than a boolean, and the real number is an expectation. Thus, the performance of an algorithm is the semantics of the formula that expresses its complexity.

In recent work [35] a new idea of scheduler-luck games was introduced. Given a randomized distributed algorithm we set up a game between a player *scheduler* and a player *luck*. The player scheduler resolves the nondeterminism trying to compromise the correctness of the algorithm, while the player luck fixes the outcome of some of the coin flips trying to ensure the correctness of the algorithm. We say that luck has a $k$ winning strategy if it can ensure correctness against any scheduler by fixing the value of at most $k$ coins. In such case we can conclude that the algorithm is correct with probability at least $1/2^k$. Scheduler-luck games can be seen as an instance of the coin lemmas that we introduce in Section 5; however in some cases they provide us with a very elegant verification technique.

## 3 The Problems with Randomization

We can view a distributed system as a collection of tasks that evolve on separate processors and that communicate through messages. A distributed algorithm is a description of the tasks that each processor should perform, and a distributed algorithm is called randomized if some of the operations performed by the processors involve some randomization. The evolution of a randomized distributed system can be described as a stochastic process; however, the stochastic process changes depending on the relative speeds of the processors and/or the scheduling policy between the processors. Thus, since processor speeds and scheduling policies are generally unknown, there is no way to know exactly the stochastic process that will take place. Stating that an algorithm is correct means stating that no matter what stochastic process takes place the algorithm achieves its goal.

Showing the correctness of a randomized distributed algorithm is not easy at all. To cite some authoritative comments, "intuition often fails to grasp the full intricacy of the algorithm" [76], and "proofs of correctness for probabilistic distributed systems are extremely slippery" [65]. Moreover, the difficulty of the problem is supported by the existence of are several flawed algorithms.

One problem in the analysis of an algorithm is that it is difficult to define precisely the probability space on which to reason, and the lack of a well defined probability space forces the designer of an algorithm to reason informally. This is the problem at the base of the error in the analysis of the the first randomized algorithm of Rabin for mutual exclusion [82]: one of the informal steps in the proof relates two probabilities that are not defined in the same probability space.

Another problem derives from the fact that sometimes the stochastic process that a designer anticipates may not take place in the algorithm. We illustrate the problem by referring to a case study [2]. In [3] a randomized algorithm for computing a spanning tree in a network is presented. In the algorithm each node tries to detect what tree to belong to and in particular each node decides whether it is a root of its tree by using randomization to break symmetry. Roughly, when two or more nodes are in conflict because they all want to be a root of the same tree, the nodes throw a random number, called an id, according to the Afek-Matias schema [1], i.e., number $i$ is chosen with probability $1/2^i$. Then the root is the process that draws the highest id. It is shown in [1] that there is a constant $c$ such that, whenever $k$ processes draw a number according to the Afek-Matias distribution ($k$ is any fixed number), with probability at least $c$ there is a unique maximum. Then we can conclude that there is a unique root with probability at least $c$, which is sufficient to show the correctness of the algorithm.

Unfortunately, a closer look at the argument above shows that there is a serious flaw. Specifically, the result of [1] is stated in terms of conditional probabilities, in the sense that if we first fix $k$ and then we draw $k$ numbers, then there is a unique maximum with probability at least $c$. However, if we choose how many id's to draw based on some partial outcome of the process, then we could continue to draw id's until we have two maximums. In such case the probability of a unique maximum would be very low. Thus, in order to show the correctness of the algorithm of [3], we need to make sure that it is not possible to schedule processes so that the number of id's to draw can be determined based on some partial outcome of the drawing process. Indeed, the problem was present in a previous unpublished version of the algorithm and it was discovered only when one of the authors tried to apply coin lemmas to verify its correctness. We will return to this problem when talking about coin lemmas.

# 4 Probabilistic Automata

In this section we describe probabilistic automata [86], the model that we use for our analysis. All our definitions are given in the discrete case; generalizations to non-discrete systems appear in [34, 33].

## 4.1 Probabilistic Automata

A *probabilistic automaton* $\mathcal{A}$ is a tuple $(S, \bar{s}, \Sigma, \mathcal{D})$ where $S$ is a set of *states*, $\bar{s} \in S$ is a *start state*, $\Sigma = (E, H)$ is a pair of disjoint sets of *external* and *internal* *(hidden) actions*, respectively, called an *interface*, and $\mathcal{D} \subseteq S \times \Sigma \times Disc(S)$ is a *transition relation*, where $Disc(S)$ denotes the set of discrete probability measures over $S$.

States are ranged over by $s, q$; actions are ranged over by $a, b, c$; internal actions are ranged over by $\tau$; discrete distributions are ranged over by $\mu$. We call an element of $\mathcal{D}$ a *transition*, and we denote a generic transition $(s, a, \mu)$ alternatively by $s \xrightarrow{a} \mu$. We say that a transition $(s, a, \mu)$ is *enabled* from $s$ and that it is *labeled* by $a$. We call a transition $(s, a, \mu)$ *Dirac* if $\mu$ is a Dirac measure, i.e., a measure that assigns probability 1 to a single element. We call a state $s$ *Dirac* if all the transitions enabled from $s$ are Dirac; we call $s$ *deterministic* if it enables at most one transition for each action; we call $s$ *single* if it enables at most one transition. We say that a probabilistic automaton is *deterministic* if each state is deterministic.

For notational convenience, we denote the elements of a generic probabilistic automaton $\mathcal{A}$ by $S, \bar{s}, \Sigma, E, H, \mathcal{D}$, and we propagate primes and indicies. Thus, the elements of $\mathcal{A}'_i$ are denoted by $S'_i, \bar{s}'_i, \Sigma'_i, E'_i, H'_i, \mathcal{D}'_i$. Sometimes we abuse of notation and write $\Sigma$ for the set $E \cup H$ rather than the pair $(E, H)$. Finally, for a state $s$ we denote by $\mathcal{D}(s)$ the set of transitions of $\mathcal{D}$ that are enabled from $s$.

If we think of a Markov process, for each state there is a unique probability distribution over states that describes the probability distribution of the next state in the stochastic process. In probabilistic automata the distribution is not unique; rather there are several distributions. From the formal point of view, in a Markov process $\mathcal{D}$ is a function rather than a relation. In the description of a distributed algorithm nondeterminism arises from the fact that from each state several processes are ready to perform a computational step, but only one of them will be the first one to take its step. Each computational step is described by a transition; thus, for each state there are several transitions enabled. Probabilistic automata are equipped also with a set of actions that take an active role in the transitions. Actions are used to describe potential communication between automata when they interact (cf. Section 4.3).

An ordinary labeled transition system (automaton) [56] is essentially a probabilistic automaton where all states are Dirac. The probabilistic automata of [86] are more general than the probabilistic automata defined here in that $\mathcal{D} \subseteq S \times Disc(\Sigma \times S \cup \perp)$, where $\perp$ is a special symbol that denotes the possibility of not moving from $s$ forever. The probabilistic automata of this paper coincide with the simple probabilistic automata of [86]. The reactive systems of [62] are deterministic probabilistic automata. The probabilistic automata of [80] are essentially our probabilistic automata. Markov Decision Processes [32] are essentially deterministic probabilistic automata. The *alternating model* [93] can be seen as a probabilistic automaton where each state is either Dirac or single.

### 4.2 Probabilistic Executions

An *execution* of a probabilistic automaton $\mathcal{A}$ is an alternate sequence of states and actions, starting with a state and, if the sequence is finite, ending with a state, $\alpha = s_0 a_1 s_2, \ldots$, such that for each $i$ there exists a transition $(s_i, a_{i+1}, \mu)$ in

$\mathcal{D}$ such that $\mu(s_{i+1}) > 0$. An execution describes one of the possible evolutions of a probabilistic automaton. It is the result of resolving the nondeterminism and the probability as well.

Denote by $\mathit{fstate}(\alpha)$ the first state of an execution $\alpha$ and, if $\alpha$ is finite, denote by $\mathit{lstate}(\alpha)$ the last state of $\alpha$. A finite execution $\alpha_1 = s_0 a_1 s_1 \cdots a_n s_n$ of $A$ and an execution $\alpha_2 = s_n a_{n+1} s_{n+1} \cdots$ of $A$ can be *concatenated*. The concatenation, written $\alpha_1 \frown \alpha_2$, is the execution $s_0 a_1 s_1 \cdots a_n s_n a_{n+1} s_{n+1} \cdots$. An execution $\alpha_1$ of $\mathcal{A}$ is a *prefix* of an execution $\alpha_2$ of $\mathcal{A}$, written $\alpha_1 \leq \alpha_2$, iff either $\alpha_1 = \alpha_2$ or $\alpha_1$ is finite and there exists an execution $\alpha_1'$ of $\mathcal{A}$ such that $\alpha_2 = \alpha_1 \frown \alpha_1'$.

To study the probabilistic behavior of a probabilistic automaton we need to study the objects that result from the resolution of the nondeterminism only, which are semi-Markov processes. We define a function $\sigma$, called a *scheduler*, that resolves the nondeterminism in a probabilistic automaton based on the past history. That is, $\sigma$ is a function that takes a finite execution and returns a sub-probability distribution over the transitions that are enabled from the last state of its argument. Formally, $\sigma(\alpha) \in \mathit{SubDisc}(\mathcal{D}(\mathit{lstate}(\alpha)))$. If in $\sigma(\alpha)$ the measure of $\mathcal{D}(\mathit{lstate}(\alpha))$ is not 1, then with probability $1 - \sigma(\alpha)(\mathcal{D}(\mathit{lstate}(\alpha)))$ no transition is scheduled. In the area of distributed algorithms schedulers are usually called *adversaries*, since they are seen as entities that try to degrade the performance of an algorithm as much as possible, while in the area of Markov Decision Processes schedulers are called *policies*, since the main objective is to find the scheduler (controller) that achieves the best performance.

If we fix a starting state $s$ and a scheduler $\sigma$, then we have determined a semi-Markov process with start state $s$. Indeed, for each finite execution $\alpha$ that starts with $s$, $\sigma(\alpha)$ determines the distribution over the next action and state in the process. We can then compute the probability of each finite execution and define a probability distribution over executions using the classical cone construction. Specifically, we need to build a collection of sets of executions, called a $\sigma$-field, that includes the empty set and is closed under complement and countable union. The elements of the $\sigma$-field are the objects that we can measure. According to the cone construction, a cone is a set of the form $C_\alpha = \{\alpha' \mid \alpha \leq \alpha'\}$, where $\alpha$ is a finite execution that starts with $s$. We consider the $\sigma$-field generated by the set of cones, that is, the smallest $\sigma$-field that includes all the cones, and we define the probability of a cone $C_\alpha$ as the probability of $\alpha$. Standard measure theoretic arguments show that the measure defined on the cones extends uniquely to a measure defined over the whole $\sigma$-field. We denote such measure by $\mu_{\sigma,s}$.

Observe that a cone $C_\alpha$ contains exactly all those executions where $\alpha$ occurs, and thus it describes the occurrence of $\alpha$. For this reason it is reasonable to define $\mu_{\sigma,s}(C_\alpha)$ as the probability of $\alpha$. To be convinced that the $\sigma$-field generated by the cones is sufficiently expressive, simply observe that any reachability property (e.g., the occurrence of some action $a$ or the reachability of some state $s$) can be expressed as a union of cones (the cones of execution where the reachability property is successful, e.g., $a$ occurs or $s$ is reached) and that the set of cones with non-zero probability is countable.

At this point the typical statements in the analysis of an algorithm are of the form "for each probabilistic execution the probability of an event $E$ is at least $p$". Of course the event $E$ must be a measurable set under any scheduler. An event $E$ could be the set of execution where each request has a response.

### 4.3 Parallel composition

We said in several occasions that modular reasoning is essential. That is, we said that we want to analyze the properties of a large system by analyzing its components separately. Parallel composition is the mechanism that allows us to compose small components to lead to a large system. In this section we introduce the parallel composition operator and its main property that enables modular reasoning.

Parallel composition is also the operator that illustrates how two or more probabilistic automata communicate. Each probabilistic automaton has an interface, which is the set of communications that the probabilistic automaton can engage in. Following the approach of CSP [49], if an action is in common between two probabilistic automata, then we force synchronization on such action: in the composition a probabilistic automaton performs the common action iff also the other probabilistic automaton performs the same action. In such case a communication takes place. The reader interested in action-based communication is referred to the extensive literature on proces algebras [49, 71, 16].

We now turn to the formal definitions. We say that two probabilistic automata $\mathcal{A}_0$ and $\mathcal{A}_1$ are *compatible* is $E_0 \cap H_1 = H_0 \cap E_1 = \emptyset$. The parallel composition of two compatible probabilistic automata $\mathcal{A}_0, \mathcal{A}_1$, denoted by $\mathcal{A}_0 \| \mathcal{A}_1$, is a probabilistic automaton $\mathcal{A}$ defined as follows:

- $S = S_0 \times S_1$;
- $\bar{s} = (\bar{s}_0, \bar{s}_1)$;
- $E = E_0 \cup E_1$ and $H = H_0 \cup H_1$;
- $\mathcal{D}$ is the set of triplets $((s_0, s_1), a, \mu_1 \times \mu_2)$, where, for $i \in \{0, 1\}$, $s_i \in S_i$, and either
  - $(s_i, a, \mu_i) \in \mathcal{D}_i$, or
  - $a \notin \Sigma_i$ and $\mu_i(s_i) = 1$.

The definition of $\mathcal{D}$ needs some more words of explanation. The measure $\mu_0 \times \mu_1$ considers the experiments described by $\mu_0$ and $\mu_1$ as independent; thus, $\mu_0 \times \mu_1(s_0, s_1)$ is defined to be $\mu_0(s_0)\mu_1(s_1)$. The automata $\mathcal{A}_0$ and $\mathcal{A}_1$ synchronize on their common actions and evolve independently on the other actions. The second item in the definition of $\mathcal{D}$ states that an automaton $\mathcal{A}_i$ idles whenever $\mathcal{A}$ performs a transition labeled by an action not in $\Sigma_i$. In other words, $\mathcal{A}_{1-i}$ evolves independently with such action. Observe that from the compatibility condition two probabilistic automata cannot engage in synchronizations through their internal actions.

The main mechanism to derive properties of a probabilistic automaton from the properties of its components is the concept of *projection*. That is, we can define the projections of a probabilistic execution onto the component automata and prove that each projection is indeed a probabilistic execution of the component. Later in Section 6.4 we can see this result at work.

Let $\mathcal{A}$ be $\mathcal{A}_0\|\mathcal{A}_1$, and let $\alpha$ be an execution of $\mathcal{A}$. For $i \in \{0,1\}$, define $\pi_i(\alpha)$, the projection of $\alpha$ onto $\mathcal{A}_i$, to be the sequence obtained from $\alpha$ by projecting each state onto its $i^{\text{th}}$ component and by removing all the actions not in $\Sigma_i$ together with their following state. It is immediate to show that $\pi_i(\alpha)$ is an execution of $\mathcal{A}_i$.

If we apply the projection operator to a probabilistic execution $\mu_{\sigma,s}$, then it is possible to show that the projection is a measurable function from the $\sigma$-field relative to $s$ and the $\sigma$-field relative to $\pi_i(s)$. Thus, the measure induced by $\pi_i$, which we denote by $\pi_i(\mu_{\sigma,s})$, is a well defined probability measure. Recall from probability theory that the induced measure $\pi_i(\mu_{\sigma,s})$ is defined as $\pi_i(\mu_{\sigma,s})[E] = \mu_{\sigma,s}[\pi_i^{-1}(E)]$ The main result is then the following.

**Theorem 1.** *Let $\mathcal{A}$ be $\mathcal{A}_0\|\mathcal{A}_1$, and let $\mu_{\sigma,s}$ be a probabilistic execution of $\mathcal{A}$. Let $i \in \{0,1\}$. Then there is a scheduler $\sigma_i$ for $\mathcal{A}_i$ such that $\pi_i(\mu_{\sigma,s}) = \mu_{\sigma_i,\pi_i(s)}$.*

### 4.4 Trace Distributions

The trace of an execution $\alpha$ is the sub-sequence of $\alpha$ that consists of the external actions. That is, a trace extracts from an execution the sequence of externally visible communication events that take place. Several properties can be studied simply by looking at the communication events of a system. As an example, if in a system requests and responses occur through communication events, then we can see whether each request obtains a response by observing traces.

The function that extracts the trace of an execution is a measurable function from the $\sigma$-field associated with a probabilistic execution and the $\sigma$-field over traces generated by cones of traces. Thus, a probabilistic execution can be associated with its corresponding probability distribution over traces, which we call a *trace distribution.*

Two probabilistic automata can be compared based on their trace distribution sets in a similar way as two ordinary automata are compared based on their languages. We say that $\mathcal{A} \sqsubseteq_D \mathcal{A}'$ if each trace distribution of $\mathcal{A}$ is also a trace distribution of $\mathcal{A}'$. Usually we say that $\mathcal{A}$ *implements* $\mathcal{A}'$.

As an example consider the property stating that a system responds to its requests with probability at least $1/2$. If requests and responses are external actions, then in $\mathcal{A}'$ each request obtains a response with probability at least $1/2$ if there is no trace distribution of $\mathcal{A}'$ where a request obtains a response with probability lower than $1/2$. Since $\mathcal{A} \sqsubseteq_D \mathcal{A}'$, then also in $\mathcal{A}$ there is no trace distribution where a request obtains a response with probability lower than $1/2$. Thus, also in $\mathcal{A}$ each request obtains a response with probability at least $1/2$.

Trace distribution inclusion is not preserved by parallel composition, which is a problem if our objective is to reason in a modular way. Roughly speaking, two probabilistic automata that have the same traces but resolve the nondeterminism in different points (say $\mathcal{A}_0$ resolves the nondeterminism before $\mathcal{A}_1$) can be distinguished by a probabilistic automaton flips a coin and publishes the result after $\mathcal{A}_0$ resolves the nondeterminism and before $\mathcal{A}_1$ resolves the nondeterminism. In this case $\mathcal{A}_1$ can resolve its nondeterminism based on the outcome of the coin flip of the context, while $\mathcal{A}_0$ cannot. See [85] for some detailed examples. For this reason we introduce a new relation called *trace distribution precongruence*, denoted by $\sqsubseteq_C$, which is the coarsest precongruence included in the trace distribution inclusion.

Studying the trace distribution precongruence directly is complicated; however we can use simulations as illustrated in the next section as a sufficient condition for trace distribution precongruence. This kind of approach is typical in the analysis of distributed systems and will be illustrated later in this paper (cf. Section 6.6).

### 4.5 Simulation Relations

Proving trace distribution inclusion is difficult in general. However, if we can show that each move of a probabilistic automaton $\mathcal{A}_0$ can be simulated by a probabilistic automaton $\mathcal{A}_1$, then we can conclude that each trace distribution of $\mathcal{A}_0$ is also a trace distribution of $\mathcal{A}_1$. This idea is known as the simulation method [69] and constitutes a very useful technique for the analysis of distributed systems [66]. In this section we describe one of the probabilistic extensions of the notion of simulation and state its properties with respect of modularity and trace distribution inclusion. We refer the interested reader to [90, 86].

To define formally the notion of simulation we need to define the notion of *weak transition*. We say that there is a weak transition from a state $s$ labeled by $a$ to distribution $\mu$, denoted by $s \stackrel{a}{\Longrightarrow} \mu$, if there exists a scheduler $\sigma$ such that in $\mu_{\sigma,s}$ the following holds:

- the support of $\mu_{\sigma,s}$ is included in the set of finite executions that start with $s$ and have trace $a$;
- for each state $s'$, $\mu(s') = \sum_{\alpha \mid lstate(\alpha)=s'} \mu_{\sigma,s}(\alpha)$.

In other words there is a weak transition from $s$ labeled by $a$ to $\mu$ if it is possible to schedule transitions in such a way that in the resulting semi Markov process we terminate with probability 1 after performing several internal action and a single external action $a$, and the distribution over the terminal states is $\mu$.

A simulation relation from a probabilistic automaton $\mathcal{A}_0$ to a probabilistic automaton $\mathcal{A}_1$ is a relation $\mathcal{R} \subseteq S_0 \times S_1$ such that

- $\bar{s}_0 \ \mathcal{R} \ \bar{s}_1$, and
- if $s_0 \ \mathcal{R} \ s_1$ and $s_0 \xrightarrow{a} \mu_0$, then there exists a distribution $\mu_1$ such that $s_1 \stackrel{a}{\Longrightarrow} \mu_1$ and $\mu_0 \ \mathcal{R} \ \mu_1$.

We write $\mathcal{A}_0 \sqsubseteq \mathcal{A}_1$ if there is a simulation from $\mathcal{A}_0$ to $\mathcal{A}_1$.

We have not said yet what it means that $\mu_0 \ \mathcal{R} \ \mu_1$. If $\mu_0$ and $\mu_1$ are Dirac distributions, then we can simply say that the states that have probability 1 are related. In general the definition is slightly more complicated since a state in the support of $\mu_0$ could be related to several states in the support of $\mu_1$ and vice versa. What counts is that each state $s_0$ in the support of $\mu_0$ is related to states in the support of $\mu_1$ that have a cumulative probability equal to $\mu_0(s_0)$ and vice versa. Formally, $\mu_0 \mathcal{R} \mu_1$ if there is a function $w : S_0 \times S_1 \to [0,1]$ such that

- if $w(s_0, s_1) > 0$ then $s_0 \mathcal{R} s_1$;
- for each $s_0 \in S_0$, $\sum_{s_1 \in S_1} w(s_0, s_1) = \mu_0(s_0)$;
- for each $s_1 \in S_1$, $\sum_{s_0 \in S_0} w(s_0, s_1) = \mu_1(s_1)$.

The two main properties of simulation relations are the fact that simulation relations are preserved by parallel composition and that they imply trace distribution inclusion. Combining the two facts, simulation relations imply trace distribution precongruence.

**Theorem 2.** *Let $\mathcal{A}_0, \mathcal{A}_1$ be two probabilistic automata such that $\mathcal{A}_0 \sqsubseteq \mathcal{A}_1$, and let $\mathcal{A}$ be a probabilistic automaton compatible with both $\mathcal{A}_0$ and $\mathcal{A}_1$. Then,*

- *$\mathcal{A}_0 \sqsubseteq_C \mathcal{A}_1$, and*
- *$\mathcal{A}_0 \| \mathcal{A} \sqsubseteq \mathcal{A}_1 \| \mathcal{A}$.*

## 5 Coin Lemmas

Stating that a property $\phi$ holds with some minimum probability $p$ no matter what the adversary does means that for each probabilistic execution the event that expresses $\phi$ has probability at least $p$. The property $\phi$ could state that an algorithm terminates successfully. Unfortunately, it is usually the case that analyzing $\phi$ directly is very difficult. Thus, rather than identifying the event that expresses $\phi$, we can identify a sub-event that is guaranteed to have probability at least $p$. It is even better and less error prone if we can identify a simple rule to associate an event of probability at least $p$ with each probabilistic execution.

A possible rule consists of identifying a set of executions $\theta$ and associate each probabilistic execution $\mu$ with the event $\Omega_\mu \cap \theta$, where $\Omega_\mu$ denotes the sample space associated with $\mu$. With such a rule, once we know that each execution of $\theta$ ensures $\phi$, we can conclude that $\phi$ holds with probability at least $p$, where $p$ is the minimum probability ensured by the rule. The main advantage of this approach is that the analysis of a property that involves probability is reduced to a property that does not involve probability. The problem that we have to solve is how to choose a rule that would guarantee the minimum probability $p$. Coin lemmas [86, 67, 88] serve this purpose.

The choice of a rule is not as simple as it might appear. As a simple example, consider an algorithm where at some time two processes may flip one coin each, and suppose that the algorithm terminates successfully whenever the two coins give the same result. Something along these lines occurs in the randomized consensus algorithm of Ben Or [21]. We would like to say that the algorithm terminates successfully with probability at least 1/2. However, there may be executions of the algorithm where either no coin is flipped or exactly one coin is flipped. In order to state that the algorithm is successful with probability at least 1/2 we have to show either that two coins are always flipped, or that the algorithm terminates successfully whenever less than two coins are flipped. Problems like this one are very easy to overlook. Alternatively, we can formulate a coin lemma stating that, if we define $\theta$ to be the set of executions where either less than two coins are flipped or two coins are flipped and they give the same result, then for each probabilistic execution $\mu$ the probability of $\Omega_\mu \cap \theta$ is at least 1/2. Then it is sufficient to consider all the executions of $\theta$ without worrying about probabilities. Observe that from the definition of $\theta$ we are forced to consider explicitly the possibility of flipping less than two coins.

We now give a formal statement of the coin lemma cited above, where we use actions to identify the coins that are flipped.

**Lemma 1.** *Let $\mathcal{A}$ be a probabilistic automaton, and let $(a_1, U_1), (a_2, U_2)$ be pairs consisting of an action and a set of states such that $a_1 \neq a_2$. Let $p_1, p_2$ be two numbers in $[0, 1]$ such that, for $i \in \{1, 2\}$, for each transition $(s, a_i, \mu)$ of $\mathcal{A}$, $\mu[U_i] \geq p_i$.*

*For each probabilistic execution $\mu$ let $FIRST((a_i, u_i))(\mu)$ be the set of executions $\alpha$ of $\Omega_\mu$ such that either $a_i$ does not occur in $\alpha$, or $a_i$ occurs in $\alpha$, and its first occurrence leads to a state of $U_i$.*

*Then, for each probabilistic execution $\mu$ of $\mathcal{A}$,*

- $P_\mu[FIRST((a_i, U_i))(\mu)] \geq p_i$.
- $P_\mu[FIRST((a_1, U_1))(\mu) \cap FIRST((a_2, U_2))(\mu)] \geq p_1 p_2$.

The main idea behind the formulation of a coin lemma is the following: we fix a collection of elementary experiments and we fix a collection of successful outcomes. Let $p$ be the probability of the successful outcomes. Then we map each elementary experiment to some random draws in a probabilistic execution in such a way that no two distinct elementary experiment map to the same random draw. If in an execution not all the elementary experiments take place, then the execution is considered by the rule if and only if it is possible to fix the outcome of the elementary experiments that do not take place in such a way that we obtain a successful outcome for the whole collection of elementary experiments. Then we are guaranteed that the chosen set of executions always has probability at least $p$.

Returning to the algorithm of [3] discussed in Section 3, the coin lemma for the experiment of throwing $k$ id's would consider the set $\theta$ of executions where either less than $k$ id's are drawn, or exactly $k$ id's are drawn and there is a unique maximum. Then the probability of $\theta$ is at least $c$, where $c$ is the probability identified in Section 3. A scheduler that draws new id's until there are two maximums would stop sometimes after drawing less than $k$ id's, and in the corresponding execution, which is considered in the coin lemma since we can fix the values of the unflipped id's so that there is a unique maximum, we would discover that there is more than one winner, thus revealing the flaw in the algorithm.

The reader interested in the analysis of the algorithm of [3] is referred to [2]; the reader interested in the analysis of the algorithm of Ben Or is referred to [86]; the reader interested in details about coin lemmas is referred to [86, 88].

## 6 An Example: Randomized Consensus

In this section we give highlights of a case study [79] where the randomized consensus algorithm of Aspnes and Herlihy [12] is proved to be correct and terminate within expected cubic time. We do not work out the analysis in full detail; rather we describe the parts of the proof that show the interplay between probability and nondeterminism and how modularity simplifies the analysis. We refer to [79] the reader interested in all the details.

### 6.1 The Consensus Problem

The consensus problem consists of making $n$ asynchronous processes decide on the same value (either 0 or 1) in the presence of stopping faults, given that each process starts with its own initial value. The initial value is provided by the environment during initialization. We say that an algorithm solves the consensus problem if it satisfies the following properties.

**Validity:** If a process decides on a value within an execution of the algorithm, then this value is the initial value of some process.
**Agreement:** Any two processes that decide within an execution of the algorithm decide on the same value.
**Wait-free termination:** All initialized and non-failed processes eventually decide.

It is known from [38] that there is no deterministic algorithm for asynchronous processes that solves consensus and guarantees termination even in the presence of at most one single faulty process. However, the problem becomes solvable using randomization if we relax the termination condition and we replace it with the following condition.

**Probabilistic wait-free termination:** With probability 1, all initialized and non-failed processes eventually decide.

### 6.2 Description of the Algorithm

The algorithm of Aspnes and Herlihy proceeds in rounds. Every process maintains a variable with two fields, *value* and *round*, that contain the process' current preferred value ($0, 1$ or $\perp$) and current round (a non-negative integer), respectively. We say that a process is at round $r$ if its *round* field is equal to $r$. Note that, due to asynchrony, different processes could be at different rounds at some point of an execution. The variables $(value, round)$ are multiple-reader single-writer. Each process starts with its *round* field initialized to 0 and its *value* field initialized to $\perp$.

After receiving the initial value to agree on, each process $i$ executes the following loop. It first reads the $(value, round)$ variables of all other processes in its local memory. We say that process $i$ is a *leader* if according to its readings its own round is greater than or equal to the rounds of all other processes. We also say that a process $i$ *observed* that another process $j$ is a leader if according to $i$'s readings the round of $j$ is greater than or equal to the rounds of all other processes. If process $i$ at round $r$ discovers that it is a leader, and that according to its readings all processes that are at rounds $r$ and $r-1$ have the same value as $i$, then $i$ breaks out of the loop and decides on its value. Otherwise, if all processes that $i$ observed to be leaders have the same value $v$ , then $i$ sets its value to $v$, increments its round and proceeds to the next iteration of the loop. In the remaining case (leaders that $i$ observed do not agree), $i$ sets its value to $\perp$ and scans the other processes again. If once again the leaders observed by $i$ do not agree, then $i$ determines its new preferred value for the next round by invoking a coin flipping protocol. There is a separate coin flipping protocol for each round. Figure 1 gives a high level view of the algorithm. The left box is the main algorithm which is subdivided into processes; the right boxes are the coin flipping protocols which interact with the main algorithm through some invocation and response messages.

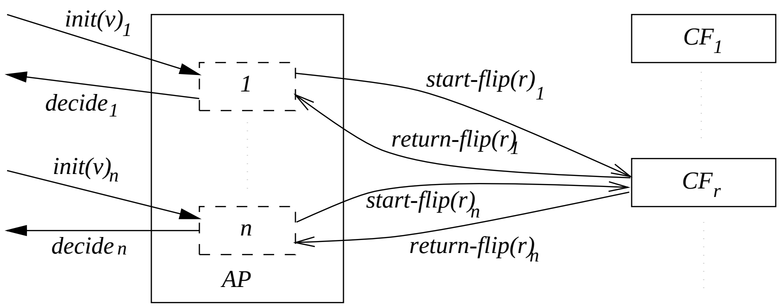


**Fig. 1.** The interaction diagram of the algorithm of Aspnes and Herlihy.

We represent the main part of the algorithm as an automaton $AP$ (Agreement Protocol), and the coin flipping protocols as probabilistic automata $CF_r$ (Coin Flipper), one for each round $r$. With this decomposition we can prove several important properties of the algorithm as properties of $AP$ using ordinary techniques for non-probabilistic systems.

### 6.3 Informal Analysis of the Algorithm

It is easy to show that the algorithm satisfies validity since if all processes start all with the same value $v$, then no process will ever observe disagreement among the leaders and no process will ever propose a value different from $v$.

It is more difficult to show that the algorithm satisfies agreement. The first important observation is that agreement does not rely on probability, but rather on the fact that the processes at the two highest rounds all agree when a process decides. The very strict condition on the decision action ensures that no process will ever be able to compromise a decision that was taken already. If a process decides $v$ at round $r$, then all processes at round $r$ agree on $v$ and no process at round $r-1$ can observe leaders with values different from $v$. More precisely, suppose for the sake of contradiction that the decision is taken by process $P$ and that there is a process $Q$ at round $r-1$ that is up to proposing a value different from $v$ for round $r$ or up to flipping a coin for the value to propose at round $r$. Let $Q$ be the first such process. This means that all processes at round $r$ or higher agree on $v$. We distinguish two exhaustive cases.

1. Process $Q$ observed that the leaders agree on a value different from $v$.
   In this case, since all processes at round $r$ prefer $v$, process $Q$ observed that the leaders are at round $r-1$. Thus, since $Q$ is at round $r-1$, $Q$ itself prefers a value different from $v$ at round $r-1$. Consider the last observation that $P$ made of $Q$. If $P$ observed $Q$ at round $r-1$, then the value preferred by $Q$ at round $r-1$ must be $v$, a contradiction (it is possible to show that a process cannot switch its preferred value within a round); if $P$ did not observe $Q$ at round $r-1$, then $P$ was already at round $r$ when $Q$ moved to round $r-1$, which means that $Q$ observed at least one process at round $r$ during its last scan, again a contradiction.
2. Process $Q$ observed that the leaders do not agree on $v$.
   Since during the second scan of process $Q$ the value proposed by $Q$ is $\perp$, process $P$ observed $Q$ either at a round lower than $r-1$ or while process $Q$ was scanning the other processes for the first time. In both cases during the second scan of round $r-1$ process $Q$ sees that process $P$ is at round $r$, and thus that all leaders agree on $v$. This is a contradiction.

The agreement property is quite intricate to analyze, and the analysis above may look incomplete since each statement relies on the understanding of several subtle interactions between processes. However, assuming that all the statements are correct, the informal analysis above provides the main ideas behind the correctness of the algorithm of Aspnes and Herlihy. In the formal proof [79] all the informal analysis above is embedded into an invariant property that can be proved easily by induction. No probability involved in the proof.

The termination property (eventually some process will decide) relies strongly on the properties of the coin flipping protocol. If at a certain round the coin flipping protocol behaves like a *global coin flip*, i.e., like the flip of a unique coin the result of which is returned to each process, then termination occurs within a few rounds. Informally, all the processes that do not flip coins to select the value for the next round will select the same value, and all the processes that flip obtain the same value. The key problem is how to define a coin flipper that behaves like a global coin flipper with high probability. We postpone the discussion to Section 6.5.

### 6.4 Using Modularity for the Analysis of Termination

We now show how modularity plays a crucial role in the analysis of termination and concentrate on showing that in the algorithm of Aspnes and Herlihy some decision is reached within some expected number of rounds. This property depends on the probabilistic properties of the coin flipping protocols. However, there are several progress properties of the algorithm that do not depend on any probabilistic assumption. We first study such properties, which are properties of $AP$ only, and then we use modularity to combine these properties with the probabilistic behavior of the coin flippers.

For each round $r$, let $CF_r$ be a coin flipping protocol, that is, a probabilistic automaton with the interface of a coin flipper of Figure 1. Define $AH$ (Aspnes-Herlihy) to be $AP \parallel CF$, where $CF$ is $(\parallel_{r\geq 1} CF_r)$.

Define a function $\phi_{MaxRound}$ that takes a finite execution $\alpha$ and gives

$$\phi_{MaxRound}(\alpha) \triangleq \mathit{lstate}(\alpha).\mathit{max\text{-}round} - \mathit{fstate}(\alpha).\mathit{max\text{-}round},$$

where the *max-round* component of a state is the maximum round number among all the processes. Define the following sets of states.

- $\mathcal{R}$ the set of reachable states of $AH$ such that some process is involved in the consensus protocol;
- $\mathcal{D}$ the set of reachable states of $AH$ such that no process is involved in the consensus protocol (each process either has decided or has not started yet).

We call the states of $\mathcal{R}$ *active*, since they represent situations where some process is participating actively in the consensus protocol. We want to show that, under some special conditions on the coin flipping protocols, starting from any state of $\mathcal{R}$, a state from $\mathcal{D}$ is reached within some bounded number of rounds. It turns out that it is easier to split the problem in two parts: first we show a simple property that, unless the algorithm terminates, the system reaches a point where one process has just moved to a new maximum round ($\mathcal{F}_0$ and $\mathcal{F}_1$ below, where the subscript corresponds to the value preferred by the process at the maximum round); then, we show that from such an intermediate point, under some special conditions on the coin flipping protocols, the algorithm terminates. Formally, define the following sets of states.

- $\mathcal{F}_0$ the set of states of $\mathcal{R}$ where there exists a round $r$ and a process $l$ such that $l$ just reached round $r$ preferring value 0, and no other process has reached round $r$ yet;
- $\mathcal{F}_1$ the set of states of $\mathcal{R}$ where there exists a round $r$ and a process $l$ such that $l$ just reached round $r$ preferring value 1, and no other process has reached round $r$ yet.

We show two properties, the first of which is almost trivial:

- **M1** If $AH$ is in a state $s$ of $\mathcal{R}$ and all invocations to the coin flippers on non-failing ports get a response, then a state from $\mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{D}$ is reached within one round.
- **M2** If $AH$ is in a state $s$ of $\mathcal{F}_v$, all invocations to the coin flippers on non-failing ports get a response, and all invocations to $CF_{s.max\text{-}round}$ get only response $v$, then a state from $\mathcal{D}$ is reached within two rounds.

Informally, "all invocations to the coin flippers on non-failing ports get responses" means that whenever a process invokes a coin flipper, either it fails or it gets a response, and "all invocations to $CF_r$ get only response $v$" means that whenever $CF_r$ returns a result, the result is $v$. We are not interested in the formal statements for the purpose of this paper. The proofs of **M1** and **M2** follow standard arguments for nondeterministic systems based on invariants.

In Section 6.5 we show how to build a distributed implementation of the coin flippers that satisfies the following properties.

**C1** For each probabilistic execution of $CF_r$ that starts with a reachable state of $CF_r$, the probability of the executions where all invocations to the coin flippers on non-failing ports get responses is 1

**C2** For each fair probabilistic execution of $CF_r$, and each value $v \in \{0, 1\}$, the probability of the executions where all invocations to $CF_r$ get only response $v$ is at least $p$, where $p$ is a real number in $[0, 1]$.

We use modularity to combine **M1**, **M2**, **C1** and **C2** and show that $AH$ guarantees progress within expected $O(1/p)$ rounds. That is, we prove the following proposition.

**Proposition 1.** *If each coin flipping protocol $CF_r$ satisfies properties* **C1** *and* **C2***, then in $AH$, starting from any state of $\mathcal{R}$ and under any scheduler, a state from $\mathcal{D}$ is reached within at most $O(1/p)$ expected rounds.*

The main statement that we use in the proof is

$$\mathcal{R} \overset{\phi_{MaxRound} \leq 3}{\underset{p}{\longrightarrow}} \mathcal{D}. \tag{1}$$

whose meaning is that under any scheduler, starting from a state of $\mathcal{R}$, with probability at least $p$ a state from $\mathcal{D}$ is reached within 3 rounds. Then we simply have to iterate the experiment of (1) until the experiment is successful, visiting at most $3/p$ expected rounds. To prove Statement (1) we prove two intermediate statements:

$$\mathcal{R} \overset{\phi_{MaxRound} \leq 1}{\underset{1}{\longrightarrow}} \mathcal{F}_0 \cup \mathcal{F}_1 \cup \mathcal{D}, \tag{2}$$

and for each $v \in \{0, 1\}$,

$$\mathcal{F}_v \overset{\phi_{MaxRound} \leq 2}{\underset{p}{\longrightarrow}} \mathcal{D}. \tag{3}$$

The proofs of Statements (2) and (3) rely on the properties **M1**, **M2**, **C1**, and **C2** and on Theorem 1.

**Proposition 2.** *Assuming that the coin flippers in $AH$ satisfy* **C1***,*

$$\mathcal{R} \overset{\phi_{MaxRound} \leq 1}{\underset{1}{\longrightarrow}} \mathcal{F}_1 \cup \mathcal{F}_0 \cup \mathcal{D}. \tag{4}$$

*Proof.* Let $\mu$ be a probabilistic execution of $AH$ that starts from a state of $\mathcal{R}$. Consider the inverse image $\theta$ of the event in $\pi_{CF}(\mu)$ stating that each invocation on a non failing port gets an answer. Then, the probability of $\theta$ is 1 by **C1** and Theorem 1. Observe that each execution of $\pi_{AP}(\theta)$ satisfies the premise of **M1**. Thus, in each execution of $\theta$ a state from $\mathcal{F}_1 \cup \mathcal{F}_0 \cup \mathcal{D}$ is reached within 1 round.

**Proposition 3.** *Assuming that the coin flippers in $AH$ satisfy* **C1** *and* **C2**,

$$\mathcal{F}_v \xrightarrow[p]{\phi_{MaxRound} \leq 2} \mathcal{D}. \tag{5}$$

*Proof.* Let $\mu$ be a probabilistic execution of $AH$ that starts from a state $s_0$ of $\mathcal{F}_v$, and let $r = s_0.\mathit{max\text{-}round}$. Consider the inverse image $\theta$ of the event in $\pi_{CF}(\mu)$ stating that each invocation on a non failing port gets an answer and each response of $CF_r$ is $v$. Then, the probability of $\theta$ is at least $p$ by **C1**, **C2** and Theorem 1. Observe that $\pi_{AP}(\theta)$ satisfies the premises of **M1** and **M2**. Thus, in each execution of $\theta$ a state from $\mathcal{D}$ is reached within 2 rounds.

### 6.5 The Coin Flippers

We are left to show how to build a distributed coin flipping protocol with the properties **C1** and **C2**. In this section we build an almost distributed version of the coin flipping protocol where processes interact through a multiple-writer multiple-reader shared register; in Section 6.6 we refine the protocol of this section to yield a distributed protocol. The protocol is based on random walks and satisfies properties **C1** and **C2** with a sufficiently high probability $p$ that is independent of $n$.

We represent the coin flipping protocol by letting an automaton $DCN_r$ (Distributed CoiN) interact with a centralized counter $CT_r$ (CounTer), that is, $CF_r = \mathit{Hide}_I(DCN_r \parallel CT_r)$, where $I$ is the set of actions used for the interaction between $DCN_r$ and $CT_r$, and $\mathit{Hide}_I$ is an operator that transforms the actions of $I$ from external to internal. Figure 2 shows the structure of the coin flipping protocol. In this section, $DCN_r$ is distributed while $CT_r$ is composed of $n$ processes that receive requests from $DCN_r$ and read/update a single shared variable: the details of the distributed implementation of a shared counter are not necessary for any properties of the coin flipping protocol. The algorithm is simple. To flip a coin a process reads the value of the counter and returns 0 if the counter is below $-Kn$, $K$ being a constant, 1 if the counter is above $Kn$. If the counter is between $-Kn$ and $Kn$, then the process flips a coin to decide whether to increment or decrement the shared counter and then starts from the beginning.

Since the protocols for $DCN_r$ and $CT_r$ are the same for any round $r$, we drop the subscript $r$ from our notation.

The idea behind the coin flipping protocol is very simple: the difference between the heads and tails obtained in the elementary coin flips form a stochastic process which is known in the literature under the name of random walk [37]. The value of the shared counter and the actual difference between heads and tails may differ by at most $n$ since in the worst case each process may be trying to update the counter. Finally, if the difference between heads and tails is greater than or equal to $(K+1)n$, then no process will ever observe a value below $Kn$. This last property requires a careful analysis, but the idea is that the processes that have to update the counter will not flip any more and the other processes will flip at most once.

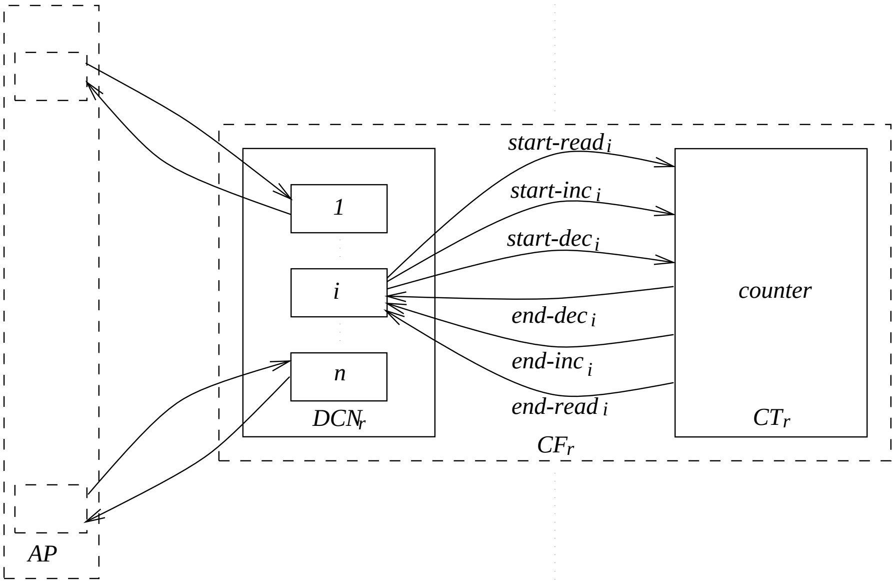


**Fig. 2.** The structure of the coin flipping protocol.

Based on the properties above, if the difference between heads and tails ends above $(K+1)n$ before ending below $-(K-1)n$, then all processes will return head: no process will ever observe a value below $-Kn$ since the value of the counter and the difference between heads and tails differ by at most $n$; furthermore, after hitting $(K+1)n$ all processes will observe a value above $Kn$. A symmetric argument holds for tail. From random walk theory, the barrier $(K+1)n$ is reached before $-(K-1)n$ with probability $(K-1)/2K$.

Most of the properties described in this informal analysis do not rely on any probabilistic assumption, and thus will not be described in detail in this paper. The only thing that is relevant here is the kind of coin lemma that we need to use for the random walk. Essentially, in the spirit of coin lemmas, we consider the set $\theta$ of those executions where either there are finitely many coin flips and the barrier $-(K-1)n$ is not reached (i.e., it is possible to fix the values of the unflipped coins so that the barrier $(K+1)n$ is reached before $-(K-1)n$), or the barrier $(K+1)n$ is reached before $-(K-1)n$. Then we are guaranteed that in each probabilistic execution the set $\Theta$ has probability at least $(K-1)/2K$. In the analysis of the coin flippers the case where the barriers are not reached does occur whenever all the processes that are flipping fail. However, such event does not compromise termination.

### 6.6 Implementing the Shared Counter

In this section we describe how to obtain an implementation of $CT$ that can replace the abstract automaton $CT$ in $CF$ without compromising properties **C1** and **C2** with $p = (K-1)/2K$. In this way, using the coin flipping protocol with the new counter, we obtain a protocol for consensus that uses only single-writer multiple-reader shared variables.

The implementation of $CT$, which we denote by $DCT$ (Distributed CounTer), is an adaptation of an algorithm proposed by Lamport [61] for read/write registers. For our purpose the details of $DCT$ are completely irrelevant. The important fact is that using standard results about atomic objects [66], we can find a simulation relation from $DCT$ to $CT$. Since simulation relations are preserved by composition (cf. Theorem 2), we have a simulation relation between the whole algorithm of Aspnes and Herlihy with the distributed counter and the whole algorithm with the centralized counter, which implies trace distribution inclusion as well.

Since the termination property is a property expressible by means of trace distributions (simply perform some visible action whenever a decision is taken), trace distribution inclusion is sufficient to show that the algorithm of Aspnes and Herlihy with the distributed counter works correctly within an expected constant number of rounds.

## 7 Performance Evaluation with Complexity Functions

Although it is not possible to estimate the running time of a distributed asynchronous algorithm (processor speeds are unknown), it is possible to give an estimate of the running time under some assumptions on the minimum speed of the processors. Typically we assume that each processor completes a computational step within 1 time unit once the computational step can be performed; any other time limit can be studied simply by scaling.

In the randomized case things are complicated by the fact that our performance measures are expectations and that such expectations change depending on how the nondeterminism is resolved. Furthermore, it is easier to compute expected rounds rather than expected time. Once again, the main idea is to work in the absence of probabilities and then lift the results to expectations. We start with the notion of a complexity function.

### 7.1 Complexity Functions

Fix a probabilistic automaton $\mathcal{A}$. A *complexity function* is a function from executions of $\mathcal{A}$ to $\Re^{\geq 0}$. A *complexity measure* is a complexity function $\phi$ such that, for each pair $\alpha_1$ and $\alpha_2$ of executions that can be concatenated, $max(\phi(\alpha_1), \phi(\alpha_2)) \leq \phi(\alpha_1 \frown \alpha_2) \leq \phi(\alpha_1) + \phi(\alpha_2)$.

Informally, a complexity measure is a function that determines the complexity of an execution. A complexity measure satisfies two natural requirements: the complexity of two tasks performed sequentially should not exceed the complexity of performing the two tasks separately and should be at least as large as the complexity of the more complex task: it should not be possible to accomplish more by working less. The restrictions imposed on complexity measures are required for the iterative argument of Section 6.4.

### 7.2 Complexity Functions and Random Variables

Consider a probabilistic execution $\mu$ of $\mathcal{A}$ and a set of executions $\Theta$ such that $\mu[\cup_{q\in\Theta} C_q] = 1$. The set $\Theta$ is said to be a full cut of $\mu$ and the event denoted by $\Theta$ is said to be *finitely satisfiable*. Informally, the elements of $\Theta$ represent the points where the property denoted by $\Theta$ is satisfied. Let $\phi$ be a complexity function. Then the restriction of $\phi$ to $\Theta$ is a random variable from the measurable space $(\Theta, 2^{\Theta})$. Define the measure on $\Theta$ as $P[q] = \mu[C_q]$. We call such random variable the random variable induced by $\Theta$. We can define the expected complexity $\phi$ to reach $\Theta$ in $\mu$ as follows:

$$E_{\mu,\Theta}[\phi] \triangleq \sum_{q\in\Theta} \phi(q)\mu[C_q],$$

i.e., the expected complexity $\phi$ to reach $\Theta$ in $\mu$ is the expected value of the random variable induced by $\Theta$.

### 7.3 Linear Combination of Complexity Functions

Given the close relationship between complexity functions and random variables in the presence of full cuts, typical results for random variables can be used. In particular, if several complexity measures are related by a linear inequality, then their expected values over a full cut are related by the same linear inequality. We use this property for the time analysis of the protocol of Aspnes and Herlihy. That is, we express the time complexity of the protocol in terms of two other complexity measures (rounds and elementary coin flips), and then we use Proposition 4 below to derive an upper bound on the expected time for termination based on upper bounds on the expected values of the other two complexity measures. The analysis of the other two complexity measures is simpler, and the relationship between time and the other two complexity measures can be studied using known methods for ordinary nondeterministic systems, with no probability involved.

**Proposition 4.** *Let $\mu$ be a probabilistic execution of some probabilistic automaton $\mathcal{A}$, and let $\Theta$ be a full cut of $\mu$. Let $\phi, \phi_1, \phi_2$ be complexity functions, and $c_1, c_2$ be two constants such that, for each $\alpha \in \Theta$, $\phi(\alpha) \leq c_1\phi_1(\alpha) + c_2\phi_2(\alpha)$. Then $E_{\mu,\Theta}[\phi] \leq c_1 E_{\mu,\Theta}[\phi_1] + c_2 E_{\mu,\Theta}[\phi_2]$.* ∎

### 7.4 Modular Analysis with Complexity Functions

To verify properties in a modular way it is useful to derive complexity properties of a composite systems based on complexity properties of the single components. Proposition 5 helps in doing this. Informally, suppose that we have a complexity function $\phi$ for $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$ and a complexity function $\phi_1$ for $\mathcal{A}_1$ such that $\phi$ and $\phi_1$ coincide up to projection. In other words $\phi$ measures in $\mathcal{A}$ the property of $\mathcal{A}_1$ that is measured by $\phi_1$. Furthermore, suppose that we know an upper bound on the expected value of $\phi_1$ that is independent of how the nondeterminism is resolved in $\mathcal{A}_1$. Then, the same upper bound is valid for $\phi$ as well. In other words, the property that we know about $\mathcal{A}_1$ can be lifted to $\mathcal{A}$.

**Proposition 5.** *Let $\mathcal{A}$ be $\mathcal{A}_1 \parallel \mathcal{A}_2$, and let $i \in \{1,2\}$. Let $\phi$ be a complexity function for $\mathcal{A}$, and let $\phi_i$ be a complexity function for $\mathcal{A}_i$. Suppose that for each finite execution $\alpha$ of $\mathcal{A}$, $\phi(\alpha) = \phi_i(\alpha\lceil\mathcal{A}_i)$. Let $c$ be a constant. Suppose that for each probabilistic execution $\mu$ of $\mathcal{A}_i$ and each full cut $\Theta$ of $\mu$, $E_{\mu,\Theta}[\phi_i] \leq c$. Then, for each probabilistic execution $\mu$ of $\mathcal{A}$ and each full cut $\Theta$ of $\mu$, $E_{\mu,\Theta}[\phi] \leq c$.* ■

## 8 Performance of the Randomized Consensus Algorithm

For the analysis of the algorithm of Aspnes and Herlihy we define several complexity functions and prove linear inequalities about them. Then we lift the results to expectations using the property of reaching $\mathcal{D}$ as our full cut. The complexity measures are the following:

- $\phi_t$: Time elapsed.
- $\phi_{flip,r}$: Elementary coin flips at round $r$.
- $\phi_{id,r}$: Updates to the counter at round $r$.
- $\phi_{id}$: Updates to the counters.

Denote by $DAH$ the algorithm of Aspnes and Herlihy with the distributed shared counter, and denote by $DCF_r$ the coin flipper for round $r$ with the distributed shared counter. A property that can be proved by means of ordinary nondeterministic analysis is the following.

**Lemma 2.** *Let $\alpha$ be an execution of $DAH$ where each computational step is taken within 1 time unit, and let $R = fstate(\alpha).max\text{-}round$. Suppose that all the states of $\alpha$, with the possible exception of $lstate(\alpha)$ are active, that is, are states of $\mathcal{R}$. Then, $\phi_t(\alpha) \leq d_1 n^2(\phi_{MaxRound}(\alpha)+R)+d_2 n\phi_{id}(\alpha)+d_3 n^2$ for some constants $d_1, d_2$, and $d_3$.*

That is, during the consensus protocol we perform quadratic work for each round and linear work for each update to the shared counters. We know already from Section 6.4 that the expected number of new rounds is constant. We need to derive an upper bound on $\phi_{id}$.

Since within a coin flipper the number of updates to the shared counter differs from the number of flipped coins at most by $n$, we can state that for each $r > 0$, $\phi_{flip,r} \leq \phi_{id,r} + n$. Using a coin lemma like argument and the relative properties of random walks, we can show that the expected value of $\phi_{flip,r}$ is quadratic in $n$. Thus, by Proposition5, for each probabilistic execution $\mu$ and each full cut $\Theta$, $E_{\mu,\Theta}[\phi_{id,r}] = O(n^2)$.

Since the expected number of visited rounds is $R$ plus a constant number, we obtain that $E_{\mu,\Theta}[\phi_{id}] = O(Rn^2)$. By replacing in the expression of Lemma 2, we obtain that the expected time for termination is $O(Rn^3)$. In particular, if initially $R = 0$, the expected time for termination is $O(n^3)$.

Once again we refer to [79] the reader interested in the details of the proof.

## 9 Concluding Remarks

In this paper we have illustrated some of the techniques that can be used for the analysis of randomized distributed algorithms. We addressed the main problem of handling probability in the presence of nondeterminism, an intrinsic characteristic of concurrent systems. All the techniques that we have illustrated originate from concurrency theory and from the experience that we have gained through the analysis of concurrent systems that do not involve probability.

We believe that this area constitutes one of the possible bridges between concurrency theory and the area of performance analysis. The results of this paper should serve two purposes: migrating some of the typical techniques of concurrency theory to performance evaluation, a goal that is achieved also by stochastic process algebras, and exposing people familiar with performance evaluation to the typical problems of concurrent systems, so that the converse migration can take place.

It would be interesting to identify other areas of application that go beyond randomized distributed algorithms to see whether the techniques of this paper, possibly extended and generalized, can be applied. It would be useful as well to see how techniques based on rewards would extend and or adapt to coin lemma based arguments.

Finally, computer aided verification is getting increasing attention. We are currently involved in a project where we use a model checker and a theorem prover to get mechanized proofs for randomized distributed algorithms [58]. In this area it is very important to find the right balance between the amount of human intervention (the problem is undecidable in general and in any case it has a high complexity) and the parts of an analysis that can be carried out via model checking and/or theorem proving. Some information about related topics can be found in this volume [55].

## References


1. Y. Afek and Y. Matias. Simple and efficient election algorithms for anonymous networks. In *3rd International Workshop on Distributed Algorithms,* Nice, France, 1989.
2. S. Aggarwal. Time optimal self-stabilizing spanning tree algorithms. Technical Report MIT/LCS/TR-632, MIT Laboratory for Computer Science, 1994. Master's thesis.
3. S. Aggarwal and S. Kutten. Time optimal self stabilizing spanning tree algorithms. In R.K. Shyamasundar, editor, *13th International Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 761 of *Lecture Notes in Computer Science*, pages 400–410, Bombay, India., December 1993. Springer-Verlag.
4. L. de Alfaro. *Formal Verification of Probabilistic Systems.* PhD thesis, Stanford University, 1997. Available as Technical report STAN-CS-TR-98-1601.
5. L. de Alfaro. How to specify and verify the long-run average behavior of probabilistic systems. In *Proceedings* $13^{th}$ *Annual Symposium on Logic in Computer Science*, pages 454–465. IEEE Computer Society Press, 1998.

6. L. de Alfaro. Stochastic transition systems. In *Proceedings of CONCUR 98,* Nice, France, volume 1466 of *Lecture Notes in Computer Science*, pages 423–438. Springer-Verlag, 1998.
7. L. de Alfaro. Computing minimum and maximum reachability times in probabilistic systems. In L. Baeten and S. Mauw, editors, *Proceedings of CONCUR 99,* Eindhoven, The Netherlands, volume 1664 of *Lecture Notes in Computer Science*, pages 66–81. Springer-Verlag, 1999.
8. L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic processes using mtbdds and the kronecker representation. In *Proceedings of TACAS'2000*, volume 1785 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.
9. R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking for probabilistic real-time systems. In J. Leach Albert, B. Monien, and M. Rodríguez, editors, *Proceedings $18^{th}$ ICALP,* Madrid, volume 510 of *Lecture Notes in Computer Science*, pages 115–136. Springer-Verlag, 1991.
10. R. Alur, C. Courcoubetis, and D.L. Dill. Verifying automata specifications of probabilistic real-time systems. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Proceedings of the REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lecture Notes in Computer Science*, pages 28–44. Springer-Verlag, 1991.
11. S. Andova. Process algebra with probabilistic choice. In J.P. Katoen, editor, *Formal Methods for Real-Time and Probabilistic Systems*, number 1601 in Lecture Notes in Computer Science, pages 111–129. Springer-Verlag, 1999.
12. J. Aspnes and M.P. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 15(1):441–460, September 1990.
13. James Aspnes and Orli Waarts. Randomized consensus in expected $O(n \log^2 n)$ operations per processor. In *33rd Annual Symposium on Foundations of Computer Science*, pages 137–146, Pittsburgh, Pennsylvania, 24–27 October 1992. IEEE.
14. J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. *Information and Computation*, 122:234–255, 1995.
15. J.C.M. Baeten and J.W. Klop, editors. *Proceedings of CONCUR 90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.
16. J.C.M. Baeten and W.P Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, 1990.
17. C. Baier. On algorithmic verification methods for probabilistic systems, 1998. Habilitation thesis, University of Mannheim.
18. C. Baier and H. Hermanns. Weak bisimulation for fully probabilistic processes. Technical Report 99/12, Centre for Telematics and Information Technology, University of Twente, 1999.
19. C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11, 1998.
20. C. Baier and M. Stoelinga. Norm functions for bisimulations with delays. In *Proceedings of FOSSACS*. Springer-Verlag, 2000.
21. M. Ben-Or. Another advantage of free choice: completely asynchronous agreement protocols. In *Proceedings of the $2^{nd}$ Annual ACM Symposium on Principles of Distributed Computing,* Montreal, Quebec, Canada, August 1983.
22. M. Bernardo, L. Donatiello, and R. Gorrieri. Modeling and analyzing concurrent systems with MPA. In U. Herzog and M. Rettelbach, editors, *Proceedings of the Second Workshop on Process Algebras and Performance Modelling* (PAPM), Erlangen, Germany, pages 175–189, 1994.

23. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proceedings Conferenco on Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513, 1995.
24. B. Bloom and A. Meyer. A remark on bisimulation between probabilistic processes. In *Proceedings of the Symposium on Logical Foundations of Computer Science*, volume 363 of *Lecture Notes in Computer Science*, pages 26–40, 1989.
25. E. Brinksma and H. Hermanns. Process algebra and stochastic process algebra. This volume.
26. I. Christoff. *Testing Equivalences for Probabilistic Processes.* PhD thesis, Department of Computer Science, Uppsala University, 1990.
27. R. Cleaveland, S.A. Smolka, and A. Zwarico. Testing preorders for probabilistic processes (extended abstract). In *Proceedings* $19^{th}$ *ICALP,* Madrid, volume 623 of *Lecture Notes in Computer Science*, pages 708–719. Springer-Verlag, 1992.
28. W.R. Cleaveland, editor. *Proceedings of CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.
29. C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In $29^{th}$ *Annual Symposium on Foundations of Computer Science*, pages 338–345, 1988.
30. C. Courcoubetis and M. Yannakakis. Markov decision procedures and regular events. In M. Paterson, editor, *Proceedings* $17^{th}$ *ICALP,* Warwick, volume 443 of *Lecture Notes in Computer Science*, pages 336–349. Springer-Verlag, July 1990.
31. P. D'Argenio, H. Hermanns, and J.P. Katoen. On asynchronous generative parallel composition. In *Proceedings of PROBMIV'98*, volume 22 of *Electronic Notes in Theoretical Computer Science*, 1999.
32. C. Derman. *Finite State Markovian Decision Processes.* Acedemic Press, 1970.
33. J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Approximating continuous Markov processes. In *Proceedings* $15^{th}$ *Annual Symposium on Logic in Computer Science,* Santa Barbara, California, pages 95–106. IEEE Computer Society Press, 2000.
34. J. Desharnais, V. Gupta, and P. Panangaden. Bisimulation for labelled Markov processes. *Information and Computation*, 1999.
35. S. Dolev, A. Israeli, and S. Moran. Analyzing expected time by scheduler-luck games. *IEEE Transactions on Parallel and Distributed Systems*, 8(4):424–440, April 1997.
36. E.A. Emerson and E.C. Clarke. Using branching time temporal logic to synthesize synchronous skeletons. *Science of Computer Programming*, 2:241–266, 1982.
37. W. Feller. *An Introduction to Probability Theory and its Applications. Volume 1.* Jokn Wiley & Sons, Inc., 1950.
38. M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with a family of faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.
39. A. Giacalone, C.C Jou, and S.A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of the Working Conference on Programming Concepts and Methods (IFIP TC2),* Sea of Galilee, Israel, 1990.
40. R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1996.
41. R.J. van Glabbeek, S.A. Smolka, B. Steffen, and C.M.N. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *Proceedings* $5^{th}$ *Annual Symposium on Logic in Computer Science,* Philadelphia, USA, pages 130–141. IEEE Computer Society Press, 1990.

42. N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and distributed system design: the integration of functional specification and performance analysis using stochastic process algebras. In L. Donatiello and R. Nelson, editors, *Performance Evaluation of Computer and Communication Systems. Joint Tutorial Papers of Performance '93 and Sigmetrics '93*, volume 729 of *Lecture Notes in Computer Science*, pages 121–146. Springer-Verlag, 1993.
43. H. Hansson. *Time and Probability in Formal Design of Distributed Systems.* PhD thesis, Department of Computer Science, Uppsala University, 1991.
44. H. Hansson. *Time and Probability in Formal Design of Distributed Systems*, volume 1 of *Real-Time Safety Critical Systems.* Elsevier, 1994.
45. H. Hansson and B. Jonsson. A framework for reasoning about time and reliability. In *Proceedings of the* $10^{\mathrm{th}}$ *IEEE Symposium on Real-Time Systems,* Santa Monica, Ca., 1989.
46. H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proceedings of the* $11^{\mathrm{th}}$ *IEEE Symposium on Real-Time Systems,* Orlando, Fl., 1990.
47. J. Hillston. Exploiting structure in solution: decomposing compositional models. This volume.
48. J. Hillston. PEPA: Performance enhanced process algebra. Technical Report CSR-24-93, Department of Computer Science, University of Edimburgh (UK), 1993.
49. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, Englewood Cliffs, 1985.
50. C. Jones and G. Plotkin. A probabilistic powerdomain of evaluations. In *Proceedings* $4^{th}$ *Annual Symposium on Logic in Computer Science,* Asilomar, California, pages 186–195. IEEE Computer Society Press, 1989.
51. B. Jonsson. Simulations between specifications of distributed systems. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings of CONCUR 91,* Amsterdam, volume 527 of *Lecture Notes in Computer Science*, pages 346–360. Springer-Verlag, 1991.
52. B. Jonsson and K.G. Larsen. Specification and refinement of probabilistic processes. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, pages 266–277, Amsterdam, July 1991.
53. B. Jonsson and J. Parrow, editors. *Proceedings of CONCUR 94,* Uppsala, Sweden, volume 836 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.
54. C.C. Jou and S.A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In Baeten and Klop [15], pages 367–383.
55. J.P. Katoen and P. D'Argenio. General distributions in process algebra. This volume.
56. R. Keller. Formal verification of parallel programs. *Communications of the ACM*, 7(19):561–572, 1976.
57. E. Kushilevitz and M. Rabin. Randomized mutual exclusion algorithms revisited. In *Proceedings of the* $11^{th}$ *Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, pages 275–284, 1992.
58. M. Kwiatkowska, G. Norman, and R. Segala. Automated verification of a randomized distributed consensus protocol using cadence smv and prism. In *Proceedings of the* $13^{\mathrm{th}}$ *Workshop on Computer Aided Verification,* Aalborg, Denmark, July 2001, Lecture Notes in Computer Science. Springer-Verlag, 2001.
59. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Verifying quantitative properties of continuous probabilistic real-time graphs. In Palamidessi [74], pages 132–137.

60. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 286, 2001.
61. L. Lamport. Concurrent reading and writing. *Communications of the ACM*, 20(11):806–811, 1977.
62. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. In *Conference Record of the* $16^{th}$ *ACM Symposium on Principles of Programming Languages,* Austin, Texas, pages 344–352, 1989.
63. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.
64. K.G. Larsen and A. Skou. Compositional verification of probabilistic processes. In Cleaveland [28], pages 456–471.
65. D. Lehmann and M. Rabin. On the advantage of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In *Proceedings of the* $8^{\text{th}}$ *Annual ACM Symposium on Principles of Programming Languages*, pages 133–138, January 1981.
66. N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., 1996.
67. N.A. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *Proceedings of the* $13^{th}$ *Annual ACM Symposium on Principles of Distributed Computing,* Los Angeles, CA, pages 314–323, 1994.
68. N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the* $6^{th}$ *Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, Vancouver, Canada, August 1987. A full version is available as MIT Technical Report MIT/LCS/TR-387.
69. Nancy Lynch and Frits Vaandrager. Forward and backward simulations – Part I: Untimed systems. *Information and Computation*, 121(2):214–233, September 1995.
70. A. McIver. Reasoning about efficiency within a probabilistic mu-calculus. In *Proceedings of PROBMIV'98*, volume 22 of *Electronic Notes in Theoretical Computer Science*, 1999.
71. R. Milner. *Communication and Concurrency*. Prentice-Hall International, Englewood Cliffs, 1989.
72. Carroll Morgan, Annabelle McIver, and Karen Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, May 1996.
73. M. Nunez and D. de Frutos. Testing semantics for probabilistic LOTOS. In *Proceedings of Formal Description Techniques VIII*, pages 365–380, 1995.
74. C. Palamidessi, editor. *Proceedings of CONCUR 2000,* University Park, PA, USA, volume 1877 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.
75. A. Philippou, I. Lee, and O. Sokolsky. Weak bisimulation for probabilistic systems. In Palamidessi [74], pages 334–349.
76. A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.
77. A. Pnueli and L. Zuck. Probabilistic verification. *Information and Computation*, 1(1):1–29, 1993.
78. A. Pogosyants and R. Segala. Formal verification of timed properties of randomized distributed algorithms. In *Proceedings of the* $14^{th}$ *Annual ACM Symposium on Principles of Distributed Computing,* Ottawa, Ontario, Canada, pages 174–183, August 1995.

79. A. Pogosyants, R. Segala, and N. Lynch. Verification of the randomized consensus algorithm of Aspnes and Herlihy: a case study. *Distributed Computing*, 13:155–186, July 2000.
80. M.O. Rabin. Probabilistic automata. *Information and Control*, 6:230–245, 1963.
81. M.O. Rabin. Probabilistic algorithms. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Results*, pages 21–39. Academic Press, 1976.
82. M.O. Rabin. $N$-process mutual exclusion with bounded waiting by $4 \log N$ shared variables. *Journal of Computer and System Sciences*, 25:66–75, 1982.
83. J.R. Rao. Reasoning about probabilistic algorithms. In *Proceedings of the* $9^{th}$ *Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, August 1990.
84. I. Saias. Proving probabilistic correctness: the case of Rabin's algorithm for mutual exclusion. In *Proceedings of the* $11^{th}$ *Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, August 1992.
85. R. Segala. A compositional trace-based semantics for probabilistic automata. In I. Lee and S.A. Smolka, editors, *Proceedings of CONCUR 95,* Philadelphia, PA, USA, volume 962 of *Lecture Notes in Computer Science*, pages 234–248. Springer-Verlag, 1995.
86. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, MIT, Dept. of Electrical Engineering and Computer Science, 1995. Also appears as technical report MIT/LCS/TR-676.
87. R. Segala. Testing probabilistic automata. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR 95,* Pisa, Italy, volume 1119 of *Lecture Notes in Computer Science*, pages 299–314. Springer-Verlag, 1996.
88. R. Segala. The essence of coin lemmas. In *Proceedings of PROBMIV'98*, volume 22 of *Electronic Notes in Theoretical Computer Science*, 1999.
89. R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. In Jonsson and Parrow [53], pages 481–496.
90. R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.
91. K. Seidel. Probabilistic communicating processes. Technical Report PRG-102, Ph.D. Thesis, Programming Research Group, Oxford University Computing Laboratory, 1992.
92. C. Tofts. A synchronous calculus of relative frequencies. In Baeten and Klop [15].
93. M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of 26th IEEE Symposium on Foundations of Computer Science*, pages 327–338, Portland, OR, 1985.
94. S.H. Wu, S. Smolka, and E.W. Stark. Composition and behaviors of probabilistic I/O automata. In Jonsson and Parrow [53].
95. S.H. Wu, S. Smolka, and E.W. Stark. Composition and behaviors of probabilistic I/O automata. *Theoretical Computer Science*, 176(1-2):1–38, 1999.
96. W. Yi and K.G. Larsen. Testing probabilistic and nondeterministic processes. In *Protocol Specification, Testing and Verification XII*, pages 47–61, 1992.