Termination of Probabilistic Concurrent Programs

SERGIU HART and MICHA SHARIR
Tel Aviv University
and
AMIR PNUELI
The Weizmann Institute of Science

The asynchronous execution behavior of several concurrent processes, which may use randomization, is studied. Viewing each process as a discrete Markov chain over the set of common execution states, necessary and sufficient conditions are given for the processes to converge almost surely to a given set of goal states under any fair, but otherwise arbitrary, schedule, provided that the state space is finite. (These conditions can be checked mechanically.) An interesting feature of the proof method is that it depends only on the topology of the transitions and not on the actual values of the probabilities. It is also shown that in this model synchronization protocols that use randomization are in certain cases no more powerful than deterministic protocols. This is demonstrated by (1) establishing lower bounds, similar to those known for deterministic protocols, on the size of a shared variable necessary to ensure mutual exclusion and lockout-free behavior of a "randomized" protocol and (2) showing that no fully symmetric "randomized" protocol can ensure mutual exclusion and freedom from lockout.

Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Concurrent Programming; D.2.4 [**Software Engineering**]: Program Verification—*correctness proofs*; D.4.1. [**Operating Systems**]: Process Management—*concurrency*; *mutual exclusion*; *scheduling*; *synchronization*; G.3 [**Mathematics of Computing**]: Probability and Statistics—*probabilistic algorithms (including Monte Carlo)*

General Terms: Algorithms, Theory, Verification

Additional Key Words and Phrases: Program analysis, Markov chains, ergodic sets

1. INTRODUCTION

Randomization has proved to be a very useful tool in the construction of certain algorithms for concurrent processes, which behave "better" than their deterministic counterparts: They use less shared memory, they have relatively simpler

A preliminary version of this paper appeared in the Conference Record of the Ninth Annual Symposium on Principles of Programming Languages, January 1982 [5].
M. Sharir's work on this paper was partially supported by the Bat-Sheva Fund at Tel Aviv University and by Office of Naval Reseach grant N00014-75-C-0571 at the Courant Institute of Mathematical Sciences, New York University.
Authors' addresses: S. Hart, Department of Statistics, Tel Aviv University, Ramat-Aviv, Tel Aviv 69978, Israel; M. Sharir, Department of Computer Science, School of Mathematical Sciences, Tel Aviv University, Ramat-Aviv, Tel Aviv 69978, Israel; A. Pnueli, Department of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel.

structure, and in certain cases they accomplish goals that deterministic algorithms provably cannot accomplish. The (insignificant) price that one usually has to pay in using such a probabilistic algorithm is that certain properties that are required from the algorithm will happen with probability one but not necessarily with certainty (e.g., when tossing a fair coin, a "head" will eventually occur with probability one, but not with certainty!) For all practical purposes, however, this difference is meaningless. Recently published algorithms of this sort include synchronization protocols [9], choice coordination [10], a symmetric distributed solution to the dining philosophers' problem [8], a probabilistic implementation of a common resource allocation scheme [3], and various algorithms in symmetric distributed systems [6].

The problem that one encounters in designing such an algorithm is in showing that the algorithm is indeed correct. This is quite a tedious task, since one must consider all possible sequences of execution steps taken by the processes and show that, except on a set of zero probability, some desired event must occur. To quote Lehmann and Rabin in [8], "The realm of proofs of correctness for concurrent processes is not well known. As the reader will realize, proofs of correctness for probabilistic distributed systems are extremely slippery."

Our aim in this paper is to shed light on this unexplored realm. In particular, assuming that each process has only finitely many states,[1] we give necessary and sufficient conditions for certain properties of a parallel program to hold (with probability one). Moreover, these conditions can be tested algorithmically, given the "atomic" structure of each process. That is, we present a fully mechanical decision procedure for testing certain properties, including, for example, freedom from deadlock and from individual lockout.

We are also concerned with comparing the capabilities of probabilistic synchronization protocols with those of deterministic ones. We provide in Section 3 two instances in which the probabilistic approach is provably not more powerful than the deterministic approach. These instances are (1) achieving mutual exclusion and freedom from individual lockout by using a shared variable v on which indivisible "test-and-set" operations are allowed (here the size of v must be $\Omega(N)$ in both cases, where N is the number of processes), and (2) achieving these same properties by a fully symmetric protocol which uses a shared variable on which separate "read" and "write" operations are allowed (this is impossible to achieve under either approach).

It turns out that a major conceptual problem in the analysis of a probabilistic concurrent program is the choice of the right model for the possible execution sequences of the processes. It is a generally accepted paradigm that the processes can be viewed as executing in sequence (with no two processes ever executing simultaneously). At each step some process is chosen to perform the next "atomic" action. The decisions about which process will be chosen to operate next are thought of as being taken by some imaginary *scheduler*, and the main problem is this: What rules can the scheduler use in its decisions? The choice of the right model for the execution scheduling of the individual processes is further complicated by the use of randomization by these processes. This may cause the schedule to depend on the outcome of random draws made by the processes. A

[1] As we see later, our results apply to more general situations as well.

problem arises here: What degree of dependency of the schedule on the outcome of random draws are we willing to assume? Allowing less dependency of this sort leads to consideration of more restricted classes of schedules. Consequently, concurrent probabilistic algorithms may perform well under such restricted schedules but may not be robust enough to retain this performance if more general schedules are also allowed. For example, the admissible schedules considered by Rabin in [9] are more restricted than those considered by Lehmann and Rabin in [8] and by us here. We show, both as a consequence of our general theorems, and also directly, that Rabin's algorithm for synchronization given in [9] does not have the required properties if one allows general schedules of the sort considered in [8].

The paper is organized as follows: In Section 2 we introduce our model of process scheduling, which, in our opinion, is the most natural one and admits rather general schedules. Using this model, we prove our main result (Theorem 2.1) giving necessary and sufficient conditions for almost-sure convergence of concurrent probabilistic programs. We also present a decision procedure for mechanical checking of these conditions. In Section 3 we compare the performance of probabilistic and deterministic synchronization protocols. We analyze Rabin's protocol given in [9] and show that it may fail to provide lockout-free behavior under our general schedules. We then show that this fact actually follows from a more general result (Theorem 3.2) concerning lower bounds on the size of a "test-and-set" shared variable necessary to provide freedom from lockout. These bounds are similar to those established by [1] for deterministic protocols, thus showing that in this case no extra power is gained by using randomization. We conclude Section 3 with a simple argument that provides another situation in which randomization cannot strengthen protocol performance. This is the case of a fully symmetric protocol using a shared variable on which separate read and write operations are allowed. Here it is impossible to provide mutual exclusion and freedom from lockout using either a deterministic or a randomized algorithm.

Our analysis is based on viewing each process participating in a concurrent program as a discrete Markov chain on the common execution states. This continues a former study by these authors [12], where Markov chains were used in the verification of sequential probabilistic programs. There the standard theory of Markov chains was quite adequate to obtain the required verification techniques. Here, however, we have to deal with certain forms of "cooperation" among several Markov chains, and this topic has been much less studied. Dubins and Savage [2] study such an interleaving under quite a different model (in particular, "fairness" is not an issue in [2], whereas in this paper it is a crucial property of the cooperations that we study).

2. CHARACTERIZATION OF PROBABILISTIC CONCURRENT TERMINATION

The model that we use for representing a probabilistic concurrent program is the following: A finite set $K = \{1, 2, \ldots, \kappa\}$ of *processes* acting on a *state space* I is given. A state $i \in I$ is identified by the location in each process and by the values of all shared and local variables. With each process $k \in K$ we associate a transition probability matrix P^k describing the possible state transitions that can occur as a result of a single atomic action of k; namely, P^k_{ij} is the probability of reaching state j from state i in a single execution step of process k. We refer to ordered pairs (i, j) with $P^k_{ij} > 0$ as *k-transitions* (from i to j). Note that, for all $i, j \in I$, $P^k_{ij} \geq 0$, and $\sum_{j \in I} P^k_{ij} = 1$. As an abbreviation for process 1, ..., process k, ... we write p1, ..., pk, ...

Let i_0 be a fixed *starting state* (or *initial state*). The interleaved scheduling of the processes in K will be modeled by an *execution tree* τ starting at i_0, defined as follows. Each node in the tree is labeled by a pair (i, k), designating the state $i \in I$ of the program and the process $k \in K$ to be scheduled next. The root is labeled by (i_0, k_0) where i_0 is the initial state and k_0 is the first process to be scheduled. For each node ν, labeled (i, k), let $J \equiv J(i, k)$ be the set of all states $j \in I$ with $P^k_{ij} > 0$ (throughout this paper, the sets $J(i, k)$ that we encounter are always assumed to be finite). For each $j \in J$ there will correspond exactly one son ν' of ν, labeled (j, k') where $k' \in K$ may vary with ν'; these will be all the sons of ν.

Thus, to each (finite or infinite) path in the tree that starts at the root, there corresponds a sequence $i_0, i_1, \ldots$ of states "visited" along it. Moreover, this correspondence is easily seen to be one to one. We therefore identify rooted paths in τ with sequences of states in I and call them *execution paths*.

Now, given an execution tree τ, to each execution path $i_0, i_1, \ldots$ there corresponds a unique sequence $k_0, k_1, \ldots$ of processes scheduled along it. For every $n \geq 0$ and every finite execution path $(i_0, i_1, \ldots, i_n)$, define $\sigma(i_0, i_1, \ldots, i_n) = k_n$ to be that process $k_n \in K$ scheduled there next; thus, (i_n, k_n) is the label of the last node on the path. We have thus defined a partial mapping σ from the set $\bigcup_{n=0}^{\infty} (\{i_0\} \times I^n)$ into K. Such a function σ is called a *schedule starting at* i_0, and τ is its associated execution tree. Note that the domain[2] of a schedule σ starting at i_0 (which is defined inductively) is the set of all finite sequences $(i_0, i_1, \ldots, i_n)$ of states in I such that, for every $r = 0, 1, \ldots, n - 1$, one has $P^{k_r}_{i_r i_{r+1}} > 0$ where $k_r = \sigma(i_0, \ldots, i_r)$. The two descriptions of the interleaving of the processes—as an execution tree τ or as a schedule σ—are clearly equivalent.

Note that the schedule's "decisions" are deterministic; that is, at each tree noae a unique process is scheduled. One might also consider more general schedules by allowing the schedule to "draw" the process to be scheduled at each node using some probability distribution on K (which may depend on the particular tree node); these are called *random schedules*. However, as is shown below, there is no loss of generality in considering deterministic schedules only. Except where otherwise stated, all schedules are henceforth assumed to be deterministic.

With each schedule σ we associate a probability measure μ_σ defined on the measurable space (Ω, Σ) where Ω is the set of all infinite sequences in I and where Σ is the σ-field of subsets of Ω generated by the *cylindrical sets*

$$\Omega(i_0, i_1, \ldots, i_n) \equiv \{\pi \in \Omega \mid \pi_j = i_j,\ j = 0, \ldots, n\}$$

[2] One could define σ on the whole set $\bigcup_{n=0}^{\infty} (\{i_0\} \times I^n)$; however, its values outside the set of execution paths are irrelevant.

where $i_0, \ldots, i_n$ is some finite sequence of states in I. μ_σ is defined so that for each cylindrical set $\Omega(i_0, \ldots, i_n)$ we have

$$\mu_\sigma(\Omega(i_0, \ldots, i_n)) = \begin{cases} P^{k_0}_{i_0,i_1} \cdot P^{k_1}_{i_1,i_2} \cdot \cdots \cdot P^{k_{n-1}}_{i_{n-1}i_n} & \text{if } i_0, \ldots, i_n \text{ is an execution path of } \sigma, \\ 0 & \text{otherwise} \end{cases}$$

where $k_r = \sigma(i_0, \ldots, i_r)$, $r = 0, \ldots, n-1$.

To ensure proper functioning of the concurrent program, we require that any admissible schedule σ possess certain "fairness" properties.

Definition. An infinite execution path $\{i_n\}_{n=0}^{\infty}$ of a schedule σ is *fair* if each $k \in K$ appears infinitely often in the sequence $\{\sigma(i_0, \ldots, i_n)\}_{n=0}^{\infty}$. A schedule σ is *strictly fair* if all its execution paths are fair.

That is, in each execution of the program each process is scheduled infinitely often. This is a standard requirement (see [8], for example, where this is called "proper") in the analysis of parallel programs.

We also consider the following weaker notion of fairness.

Definition. A schedule σ is *fair* if μ_σ-almost every execution path of σ is fair.

As is shown below (Proposition 2.3), fair schedules are in some sense limit points of strictly fair schedules, so that with no loss of generality we could deal with either class of schedules and obtain essentially the same results.

The main problem that we are concerned with is to compute the probability of reachability of certain states from other states. For example, if i is a state in which a process k, participating in some synchronization protocol, is trying to enter its critical section, and if X is the set of all states in which k is at its critical section, then the probability of reaching X from i is the probability that k will not be locked out of entering its critical section from state i. Similarly, if X_1 is the set of states in which *some* process is at its critical section, then the probability of reaching X_1 from i is the probability that the system will not be deadlocked at state i.

Let therefore $X \subset I$ be a given set of "goal" states, let $i \in I$ be the initial state, and let σ be some fair schedule starting at i. Our aim is to compute

$$f^*_X(\sigma) \equiv \text{probability of ever reaching a state in } X \text{ from } i \text{ under } \sigma.$$

Note that this can also be expressed as

$$f^*_X(\sigma) = \sum_{n=0}^{\infty} f^{(n)}_X(\sigma)$$

where

$f^{(n)}_X(\sigma)$ = probability of reaching a state in X from i for the first time after exactly n steps under σ.

Note also that $f^*_X(\sigma)$ can also be written as $\mu_\sigma(A(X))$ where $A(X)$ is the event that a state in X is ever reached.

Since we are interested in showing that $f^*_X(\sigma) = 1$ for every admissible schedule

σ, we define

$$h^*_{i,X} = \inf\{f^*_X(\sigma) : \sigma \text{ a fair schedule starting at } i\},$$

and we seek conditions under which $h^*_{i,X} = 1$.

Let us first explain our reasoning intuitively. One way in which an adversary schedule can prevent the system from ever entering X is to iterate forever through a set of states E disjoint from X, scheduling at each $i \in E$ some process k that transfers i only to states in E, and using all processes in this manner (so that it remains fair). We show that, in particular, if the state space I is finite, then the nonexistence of such a set E is a necessary and sufficient condition for $h^*_{i,X}$ to be 1 for all $i \in I$.

We begin with a precise definition of the above notion:

Definition. A set $E \subset I$ is called *K-ergodic* if the following holds: For each $i \in E$ define

$$K_E(i) \equiv \{k \in K : P^k_{i,E} \equiv \sum_{j \in E} P^k_{ij} = 1\}$$

(this is the set of processes that cannot lead from i to a state outside E). Then we require that

(i) $$\bigcup_{i \in E} K_E(i) = K;$$

that is, for every process $k \in K$ there is a state $i \in E$ such that all k transitions from i leave the system in E.

(ii) For every $i \neq j \in E$ there exists a chain $i_0, i_1, \ldots, i_r \in E$ with $i_0 = i$, $i_r = j$, and a chain $k_0, k_1, \ldots, k_{r-1} \in K$ of processes such that for each $s = 0, 1, \ldots, r - 1$

$$k_s \in K_E(i_s)$$

and

$$P^{k_s}_{i_s i_{s+1}} > 0.$$

(A set E satisfying (ii) is called *communicating*.)

Let $\hat{\imath} \in I$ be the initial state and $X \subset I$ be the goal set with $\hat{\imath} \notin X$. Define $\hat{I}$ as the set of all states that can be reached (with positive probability) from $\hat{\imath}$ before a state in X is reached, using any finite sequence of processes. $\hat{I}$ includes $\hat{\imath}$ and is disjoint from X. Our main result is

THEOREM 2.1 *Let $\hat{\imath}$, X, $\hat{I}$ be as above, and assume that $\hat{I}$ is finite. Then the following conditions are equivalent*:

(1) $h^*_{\hat{\imath},X} = 1$.
(2) $h^*_{i,X} = 1$ *for each* $i \in \hat{I}$.
(3) $\hat{I}$ *does not contain any K-ergodic set.*
(4) *There exists a decomposition of $\hat{I}$ into disjoint sets $I_1, \ldots, I_n$ such that, if we put $J_m = \bigcup_{r=0}^{m} I_r$, $m = 0, 1, \ldots, n$, with $I_0 = X$, then for each $m = 1, 2, \ldots, n$*

we have the following:

(*a*) *For each* $i \in I_m$, $k \in K$, *if* $P^k_{i,J_{m-1}} = 0$, *then* $P^k_{i,I_m} = 1$.
(*b*) *There exists* $k \equiv k(m) \in K$ *such that, for each* $i \in I_m$, $P^k_{i,J_{m-1}} > 0$.

(Condition (4a) says that, if process k can transfer the system from a state in I_m to a state outside I_m, then some k-transitions (with nonzero probabilities) move the system "down" the chain $\{I_r\}$, toward the goal I_0; condition (4b) ensures the existence of at least one process that would do this for all states in I_m.)

Remarks. (1) We can regard the partition of $\hat{I} \cup X$ into $I_0, \ldots, I_n$ as the assignment of an element of the well-founded set $\{0, \ldots, n\}$ to each state $i \in I$. That is, each $i \in I_m$, $0 \leq m \leq n$, is assigned the value m. Denote this assignment by $\rho: I \to \{0, \ldots, n\}$.

Condition (4a) then states that, for every i and k, if there is no k-transition leading to i' such that $\rho(i) > \rho(i')$, then *all* k-transitions must lead to states i' such that $\rho(i) = \rho(i')$.

Condition (4b) states that for every m, $1 \leq m \leq n$, there exists a $k = k(m)$ such that for every $i \in I_m$ there is a k-transition from i to i' with $\rho(i') < \rho(i) = m$.

This view shows our method to be an extension of the proof method given by Lehmann, Pnueli, and Stavi in [7] for the just termination of deterministic concurrent programs. Indeed, if we restrict ourselves to deterministic processes (i.e., assume that for any $i \in I$, $k \in K$, $P^k_{ij} = 1$ for exactly one $j \in I$), then checking the validity of our condition (4) reduces to the proof method given in [7]. Recently, the first two authors have generalized conditions (3) and (4) to processes with infinite state sets, in a manner that also generalizes the proof method of [7] for deterministic or nondeterministic processes; see [4].

(2) An immediate corollary of the preceding theorem is that the convergence is independent of the particular values of nonzero transition probabilities, but depends only on the "topology" of these transitions.

Before presenting the proof of Theorem 2.1, we state and prove a "zero-one law" for concurrent probabilistic convergence, which is needed in that proof and is also of independent interest.

THEOREM 2.2 (ZERO-ONE LAW). *Let* $\hat{\imath}$, X, $\hat{I}$ *be as above, and assume that* $\hat{I}$ *is finite. Then* $\min_{i \in \hat{I}} h^*_{i,X}$ *is either* 0 *or* 1.

PROOF. It suffices to show that, if $h^*_{i,X} > 0$ for each $i \in \hat{I}$, then $h^*_{i,X} = 1$ for each $i \in \hat{I}$.

For this, put $\beta = \min_{i \in \hat{I}} h^*_{i,X} > 0$, and let $i \in \hat{I}$ for which $\beta = h^*_{i,X}$. Suppose $\beta < 1$, and let $\varepsilon = \beta(1 - \beta)/2 > 0$. Then there exists a fair schedule σ such that $\beta \leq f^*_X(\sigma) < \beta + \varepsilon$. Now recall that

$$f^*_X(\sigma) = \sum_{n=0}^{\infty} f^{(n)}_X(\sigma);$$

hence, there exists N large enough such that

$$\gamma_N \equiv \sum_{n=0}^{N} f^{(n)}_X(\sigma) > \frac{\beta}{2}$$

(γ_N is the probability of getting from i to X under σ in no more than N steps). Hence we can write

$$f_X^*(\sigma) = \gamma_N + \sum_{j \in \hat{I}} q_j \cdot r_j$$

where q_j is the probability of getting from i to j under σ in N steps without going through X, and where r_j is the probability of getting from j to X under the remainder of σ from the Nth step onward. By our assumptions we have $r_j \geq h_{j,X}^* \geq \beta$, because the remainder of σ is also fair. Also,

$$\sum_{j \in \hat{I}} q_j = 1 - \gamma_N.$$

Hence

$$\beta + \varepsilon > f_X^*(\sigma) \geq \gamma_N + (1 - \gamma_N)\beta = \beta + \gamma_N(1 - \beta) \geq \beta + \frac{\beta}{2}(1 - \beta),$$

so that

$$\frac{\beta}{2}(1 - \beta) = \varepsilon > \frac{\beta}{2}(1 - \beta),$$

a contradiction which proves the theorem. Q.E.D.

Proof of Theorem 2.1

(3) $\Rightarrow$ (4). Assume $\hat{I}$ does not contain any K-ergodic set. We claim that for any set $H \subset \hat{I}$ there exists a nonempty subset $G \subset H$ such that

(a) for each $i \in G$, $k \in K$, $P_{i,H}^k = 1$ implies $P_{i,G}^k = 1$;
(b) there exists $k \in K$ such that $P_{i,G}^k < 1$ for each $i \in G$ (and hence, by (a), $P_{i,H}^k < 1$ and $P_{i,H^c}^k > 0$).

Suppose that this claim has been established. Assume that $I_0 = X, I_1, \ldots, I_{m-1}$ have already been defined, and let $J_{m-1} = \bigcup_{k=0}^{m-1} I_k$, $H = I \backslash J_{m-1}$. Then, if we apply the above claim to this set H, the resulting set G will be taken as the set I_m, and the properties (a) and (b) are then precisely the required properties (4a) and (4b). Continuing inductively in this manner, we obtain the required decomposition (4).

To prove the claim, we define on H a relation $\rightarrow$ as follows: For $i, j \in H$, $i \rightarrow j$ if there exists $k \in K$ such that $P_{i,H}^k = 1$ and $P_{ij}^k > 0$ (i.e., if j can be reached from i in a single transition by a process that cannot take us out of H). Let $\Rightarrow$ denote the reflexive and transitive closure of $\rightarrow$, and define $i \sim j$ iff $i \Rightarrow j$ and $j \Rightarrow i$. The last relation is an equivalence relation, and $\Rightarrow$ induces on the set of equivalence classes a partial order relation. Let $G \subset H$ be an equivalence class having no successors under this relation (recall that H is a finite set). We claim that G has the required properties. Indeed, let $i \in G$ and $k \in K$ be such that $P_{i,H}^k = 1$. Let $j \in H$ be such that $P_{ij}^k > 0$. Then $i \rightarrow j$, and, since G is "last" under $\Rightarrow$, j must belong to G. Hence $P_{i,G}^k = 1$, and (a) is established. Hence, if $i, j \in G$, $i \rightarrow j$ if and only if there exists $k \in K$ such that $P_{i,G}^k = 1$ (i.e., $k \in K_G(i)$) and $P_{ij}^k > 0$. Since G is an equivalence class, it satisfies property (ii) in the definition of K-ergodicity.

But G cannot be K-ergodic by assumption, so that it cannot satisfy property (i). Hence there exists $k \in K$ such that $P^k_{i,G} < 1$ for all $i \in G$, and this proves (b).

(4) $\Rightarrow$ (2). We show that $f^*_X(\sigma) = 1$ for each $i \in I$ and each fair schedule σ starting at i. By Theorem 2.2, this follows from the assertion

$$f^*_X(\sigma) \geq \alpha^m, \quad i \in I_m, \qquad m = 0, 1, \ldots, n$$

where $\{I_m\}^n_{m=1}$ is the decomposition assumed in (4) and where α is

$$\alpha \equiv \min(\{P^k_{ij}: k \in K,\, i \in \hat{I},\, j \in \hat{I} \text{ such that } P^k_{ij} > 0\} \cup \{P^k_{i,X}: k \in K,\, i \in \hat{I} \text{ such that } P^k_{i,X} > 0\}) > 0$$

(due to the finiteness of $\hat{I}$; if I itself is finite, one could also take $\alpha = \min\{P^k_{ij}: P^k_{ij} > 0\}$). The claim holds trivially for $m = 0$. Assume that it holds for all $i \in J_{m-1}$ for some m. Let $i \in I_m$, and let σ be a fair schedule starting at i. Let $k \equiv k(m)$ be given by property (4b).

Since σ is fair, we claim that, with probability one, we eventually reach a node (j, k) in the execution tree of σ with $P^k_{j,J_{m-1}} > 0$ (otherwise, in particular, $k(m)$ will not be scheduled and σ will not be fair).

At the first such occurrence (on almost every path), there will be a probability of at least α of next being in J_{m-1}. Therefore, for any fair σ, starting at any $i \in I_m$, the probability of ever reaching J_{m-1} is $\geq \alpha$; from J_{m-1}, by the induction hypothesis, there is a probability of at least α^{m-1} of eventually getting to X. Hence $f^*_X(\sigma) \geq \alpha^{m-1} \cdot \alpha = \alpha^m$, and that proves our claim.

(2) $\Rightarrow$ (1). Trivial.

(1) $\Rightarrow$ (3). Let $E \subset \hat{I}$ be a K-ergodic set. We construct a fair schedule σ starting at $\hat{\imath}$ such that $f^*_X(\sigma) < 1$. The basic idea is to construct σ so that, once E is entered, it will never be left subsequently (almost surely). The main difficulty is to ensure that σ is fair. This is done by attempting to schedule processes in cyclic order. However, if we want to schedule some $\bar{k} \in K$, we may have to postpone this scheduling until we reach a state $i \in E$ for which $\bar{k} \in K_E(i)$. If we have not yet reached such a state, we want to schedule other processes in such a way that with probability one such a state will eventually be reached. This is done as follows:

Let $\bar{k} \in K$ be fixed. We partition E into a disjoint union $\{E_m\}^n_{m=1}$ as follows (we denote $E_1 \cup E_2 \cup \cdots \cup E_m$ by F_m):

$$E_1 = \{i \in E : \bar{k} \in K_E(i)\} \neq \emptyset \qquad \text{(by (i) of } K\text{-ergodicity);}$$

$$E_2 = \{i \in E \backslash F_1 : \exists k \in K_E(i),\, P^k_{i,E_1} > 0\};$$

$$\vdots$$

$$E_m = \{i \in E \backslash F_{m-1} : \exists k \in K_E(i),\, P^k_{i,E_{m-1}} > 0\};$$

$$\vdots$$

Since E is communicating (i.e., satisfies property (ii) of K-ergodicity), every $i \in E$ will belong to some E_m; the finiteness of $\hat{I}$ implies that there are only finitely many sets E_m. (This decomposition is *not* related to the one in (4).) Thus, in order to schedule $\bar{k}$, the following collection of partial schedules $\sigma_{\bar{k}}(i)$ may be used: At $i \in E_1$, schedule $\bar{k}$ and terminate $\sigma_{\bar{k}}(i)$. At $i \in E_m$, $m \geq 2$, schedule any $k \in K_E(i)$ such that $P^k_{i,E_{m-1}} > 0$. By induction, it follows that the probability of using $\bar{k}$ (i.e., of reaching E_1) if we start at a state $i \in E_m$ is at least α^{m-1} (where α is as defined earlier in the proof). Hence there is a probability of at least α^{n-1} of using $\bar{k}$ starting from any $i \in E$. We can then use an argument similar to that given in Theorem 2.2 to show that this probability is one for each $i \in E$. Indeed, let q_i denote the probability of using $\bar{k}$ under $\sigma_{\bar{k}}(i)$ from $i \in E$, put $\beta = \min_{i \in E} q_i > 0$, and let $i \in E$ be such that $\beta = q_i$. Hence there exists N large enough so that

$$\gamma_N \equiv (\text{probability of using } \bar{k} \text{ under } \sigma_{\bar{k}}(i) \text{ in no more than } N \text{ steps}) > \frac{\beta}{2} > 0,$$

and we can write

$$\beta = q_i = \gamma_N + \sum_{j \in E} s_j \cdot q_j$$

where s_j is the probability of reaching state j in N steps without using $\bar{k}$, and q_j is the probability of using $\bar{k}$ from j under $\sigma_{\bar{k}}(j)$ (by definition of $\sigma_{\bar{k}}$, the latter probability does not depend on N). Hence

$$\beta \geq \gamma_N + \sum_j s_j \cdot \beta = \gamma_N + (1 - \gamma_N)\beta = \beta + \gamma_N(1 - \beta),$$

and this is a contradiction unless $\beta = 1$.

The desired schedule σ is now constructed as follows: By definition of $\hat{I}$ and the fact that $E \subset \hat{I}$, there is a finite sequence of processes such that, if they are scheduled in this sequence, there is a positive probability $\gamma > 0$ of reaching a state in E from the initial state $\hat{\imath}$. The initial portion of σ is then constructed to yield at least this probability of reaching E by scheduling this sequence of processes. On other paths of σ we simply schedule the processes in a round-robin fashion until we reach a state in E, if at all.

Once having reached some state $i \in E$, we continue the construction of σ in phases, where the nth phase has a goal of scheduling process $n \pmod k + 1$, $n \geq 0$. Suppose we are in a phase that wishes to schedule $\bar{k}$. We then use $\sigma_{\bar{k}}$ as defined above until $\bar{k}$ is scheduled (with probability zero, $\sigma_{\bar{k}}$ may not terminate). Once $\bar{k}$ has been scheduled, we go on to the next phase. Since there are only countably many phases, the probability of not using all processes (while in E) infinitely often is zero, and outside E all processes will be scheduled infinitely often with certainty. Hence σ is fair. Moreover, σ will never take us out of E once having reached this set. Hence the probability of absorption at X (which is disjoint from E) is at most $1 - \gamma < 1$. This concludes the proof of the theorem. Q.E.D.

COROLLARY. *Let $\hat{\imath}$, X, and $\hat{I}$ be as above, and assume that $\hat{I}$ is finite. If $h^*_{\hat{\imath},X} < 1$, then there exist $i \in \hat{I}$ and a fair schedule σ starting at i such that $f^*_{\hat{X}}(\sigma) = 0$.*

PROOF. By Theorem 2.1 there exists a K-ergodic set $E \subset \hat{I}$. For each $i \in E$ we can construct a fair schedule σ starting at i, as in the last part of the proof of the theorem, for which $f^*_{\hat{X}}(\sigma) = 0$ (since the system never leaves E under σ). Q.E.D.

Thus, if $\min_{i\in\hat{I}} h^*_{i,x} = 0$, then this value is indeed attained by some fair schedule starting at some $i \in \hat{I}$. However, it is possible for this value to be zero, and yet have $f^*_X(\sigma) > 0$ for each $i \in \hat{I}$ and each *strictly fair* schedule σ. Thus there are cases in which the probability of reaching X is always nonzero under strictly fair schedules but may be zero for some fair schedules. Hence, in order to apply the zero-one law established above, it is necessary to check that $f^*_X(\sigma) > 0$ for all *fair* schedules and all $i \in \hat{I}$. To illustrate such a case consider the following example. Let $\hat{I} = \{1, 2\}$ and $X = \{0\}$, with two processes 1, 2 having the following transition matrices:

$$P^1 \quad \begin{array}{c|ccc} & 1 & 2 & 0 \\ \hline 1 & 0 & 0 & 1 \\ 2 & 0 & 1 & 0 \end{array} \qquad\qquad P^2 \quad \begin{array}{c|ccc} & 1 & 2 & 0 \\ \hline 1 & \frac{1}{2} & \frac{1}{2} & 0 \\ 2 & \frac{1}{2} & \frac{1}{2} & 0 \end{array}$$

That is, p2 repeatedly draws the next state, whereas p1 enters X at state 1 but waits at state 2. The following fair schedule σ will almost surely prevent p1 from entering X: In state 2 it schedules p1 and p2 alternately. In state 1 it always schedules p2. The probability that σ will schedule p1 infinitely often is one, since this will not happen only if p2 draws 1's repeatedly an infinite number of times, an event whose probability is zero. Hence σ is fair, and it locks out p1 from entering state 0 almost surely. On the other hand, for any strictly fair schedule σ, the probability of p1 entering X is positive. Indeed, consider an execution path in which only state 1 is visited, and where p2 repeatedly draws only 1's.

However, the difference between the two types of schedules—fair and strictly fair—is only marginal. Indeed, both Theorems 2.1 and 2.2 remain true if we restrict ourselves to strictly fair schedules only. This will be a consequence of the following proposition. (Recall that the *norm* $\|\mu\|$ of a real-valued measure μ over a σ-field Σ is defined as

$$\sup_{A,B\in\Sigma} |\mu(A) - \mu(B)| = \sup_{A\in\Sigma} \mu(A) - \inf_{B\in\Sigma} \mu(\mathrm{B}).)$$

PROPOSITION 2.3 *Let σ be a fair schedule, and let $\varepsilon > 0$ be given. Then there exists a strictly fair schedule σ_ε such that $\|\mu_{\sigma_\varepsilon} - \mu_\sigma\| < \varepsilon$.*

PROOF. Since σ is fair, p1 will be scheduled eventually with probability one. Hence there is n_1 large enough so that p1 will be scheduled by σ in no more than n_1 steps with probability greater than $1 - (\varepsilon/2)$. We change σ at step n_1 as follows: If p1 has not yet been scheduled, schedule it at step n_1 and use the processes in a round-robin fashion (i.e., cyclically) from there onward; otherwise, leave σ unchanged. This yields a schedule σ_1 for which $\|\mu_\sigma - \mu_{\sigma_1}\| < \varepsilon/2$. Next we pick p2 and repeat the same argument to the unchanged part of σ from step n_1 onward (i.e., to the finite collection of trees rooted at nodes of the execution tree of σ at the n_1 level). This will yield $n_2 > n_1$ at which the probability of having scheduled p2 before step n_2 is $1 - \varepsilon/4$. On the set where this did not happen we modify σ_1 by scheduling p2 at n_2 and then schedule all processes cyclically as before. We continue in this manner by picking up all processes $1, 2, \ldots, \kappa$ repeatedly in cyclic order, obtaining in the sth iteration a step n_s at which a perturbation of σ_{s-1} on a set having measure $\varepsilon/2^s$ is required to ensure that the current $\bar{k}$ will be scheduled in its turn with certainty.

The schedule σ_ε thus constructed is well defined and strictly fair, and

$$\|\mu_{\sigma_\varepsilon} - \mu_\sigma\| < \frac{\varepsilon}{2} + \frac{\varepsilon}{4} + \cdots = \varepsilon. \qquad \text{Q.E.D.}$$

COROLLARY. *Theorems* 2.1 *and* 2.2 *remain true with*

$$h^*_{i,X} = \mathit{inf}\{ f^*_X(\sigma) : \sigma \text{ } a \text{ strictly } \mathit{fair\ schedule\ starting\ at\ } i\}.$$

PROOF. For every schedule σ, $f^*_X(\sigma) = \mu_\sigma(A(X))$ where $A(X)$ is the set of execution paths reaching X; therefore $\|\mu_{\sigma_\varepsilon} - \mu_\sigma\| < \varepsilon$ implies $|f^*_X(\sigma_\varepsilon) - f^*_X(\sigma)| < \varepsilon$. Q.E.D.

Random Schedules. As mentioned above, the schedules that we have considered so far are deterministic, in the sense that they schedule a unique process at each node in the associated execution tree. Naturally, we are led to considering more complicated schedules that use randomizations in their scheduling decisions. Can such random fair schedules be worse than deterministic ones, that is, can such schedules prevent termination of a concurrent program that terminates almost surely under any fair deterministic schedule? There are two main ways of introducing randomizations into a schedule. The first consists of a random choice of the next process k at each node in the execution tree (i.e., instead of assigning some $k \in K$ to each node, the schedule assigns to each node a probability distribution over K). The second way consists of taking a probability distribution over the set of deterministic schedules (i.e., make all the random decisions a priori). (In game theory there are similar objects—namely, strategies; the deterministic ones are called "pure," and the two cases of random ones are called "behavior" and "mixed," respectively.)

We contend, however, that introducing random schedules into our analysis does not change the almost-sure termination properties of a concurrent program. It is easy to see that the first case (independent randomization at each decision node) is a special case of the second one (by doing all randomizations at the start!) Thus we need consider schedules of the second type only, that is, "mixed" schedules. To establish our assertion, recall that the quantity $f^*_X(\sigma)$, which we wish to estimate, can in fact be written as $\mu_\sigma(A(X))$ where μ_σ is the cylindrical measure induced by σ on the path space Ω and where $A(X)$ is the set of all execution paths starting at i and containing an element of X. Note also that the notion of a *fair* schedule depends only on the measure μ_σ and so can be easily extended to random schedules too. Now, if σ is a mixed fair schedule, that is, a probabilistic mixture of deterministic fair schedules, then μ_σ is simply a convex combination of $\mu_{\sigma'}$ for the corresponding deterministic fair schedules σ'. Hence, if $\mu_{\sigma'}(A(X)) = 1$ for each deterministic fair schedule σ', then also $\mu_\sigma(A(X)) = 1$. By the comment made above, the same is true also for "behavior" fair schedules σ. Hence we conclude that $f^*_X(\sigma) = 1$ for each fair deterministic schedule if and only if $f^*_X(\sigma) = 1$ for each fair random schedule, so that with no loss of generality we can, as we have actually done, consider only deterministic schedules in our analysis.

Next we present an algorithm that, given the transition probability matrices of the processes, the set X of goal states, a starting state $\hat{\imath}$, and the set $\hat{I}$ of intermediate states, either constructs a K-ergodic set $E \subset \hat{I}$, thereby showing that $h^*_{\hat{\imath},X} < 1$, or else builds up the decomposition $\{I_m\}$ as in statement (4) of Theorem

2.1, thereby showing that $h^*_{\hat{i},X} = 1$. The validity of the algorithm follows immediately from the proof of Theorem 2.1.

The algorithm proceeds as follows:

(1) Construct a transition graph G that is a directed labeled graph whose nodes are elements of $\hat{I}$, with one additional node designating X; for each $k \in K$ and each $i \in \hat{I}$ and $j \in \hat{I}$ (or $j = X$) such that $P^k_{ij} > 0$, we draw an edge e from i to j and label e by k.
(2) We begin to construct a decomposition $\{I_m\}$ by putting $I_0 = X$.
(3) Suppose that $I_0, \ldots, I_r$ have already been constructed. Delete from G all nodes belonging to the union $J_r = \bigcup_{m=0}^{r} I_m$ of these sets; also, for each $i \in \hat{I} \backslash J_r$ delete from G all edges e such that either (1) e leads from i to some $j \in J_r$ or (2) there exists another edge e' leading from i to some node in J_r, and e and e' are labeled by the same process $k \in K$. (In other words, we ignore states that have already been assigned a rank and transitions from a state i by processes that can, in a possibly alternative transition from i, move the system into J_r.) Let G_r denote the resulting graph.
(4) Partition G_r into strongly connected components, and let E be such a component having no successors (a terminal component). If each process $k \in K$ labels some internal edge of E, then E is a K-ergodic set, in which case the algorithm halts, having found such a set; otherwise, put $I_{(r+1)} = E$, define $k(r+1)$ to be a process k which does not label any internal edge of E, and repeat steps (3) and (4).
(5) If G finally empties out, then we have found the desired decomposition of $\hat{I}$.

We next illustrate these techniques with three simple examples.

Example 1. Consider the following two-process synchronization, where a shared variable c is used (we assume indivisible "test-and-set" operations on c). c has 3 values: 0, designating a neutral state; 1, a state in which process p1 will enter its critical section; and 2, where p2 will enter. The code for p1 (respectively, p2) is as follows:

```
Try: if c = 1 [respectively, 2] then go to Ex
     else
        if c = 0 then c := Random(1, 2) fi;     /* draw 1 or 2 with equal probabilities */
        go to Try
     fi
Ex:  —critical region—
     c := 0;
     go to Try
```

Here we have five states, each represented as (c, l_1, l_2) where c is the value of the shared variable and where l_1 (respectively, l_2) is the location in process p1 (respectively, p2), which can assume the values T (for being at the trying section) and X (for being in the critical section). We assume that both processes remain live indefinitely. The states are

$$i_0 = (0, \mathrm{T}, \mathrm{T});$$
$$i_1 = (1, \mathrm{T}, \mathrm{T});$$
$$i_2 = (2, \mathrm{T}, \mathrm{T});$$
$$i_3 = (2, \mathrm{T}, \mathrm{X});$$
$$\mathrm{X} = i_4 = (1, \mathrm{X}, \mathrm{T});$$

and the transition matrices are as shown in Figure 1.

	P^1					P^2				
	i_0	i_1	i_2	i_3	X	i_0	i_1	i_2	i_3	X
i_0		$\frac{1}{2}$	$\frac{1}{2}$				$\frac{1}{2}$	$\frac{1}{2}$		
i_1					1		1			
i_2			1						1	
i_3				1		1				

Figure 1

We now show that $\hat{I} = \{i_0, i_1, i_2, i_3\}$ does not contain a K-ergodic set by applying the algorithm described above. Begin by putting $I_0 = X = \{i_4\}$. The other sets are found as follows:

We first construct G as shown in Figure 2. (Each edge is labeled by the process that induces it.)

To find I_1, delete the 1-edge from i_1 to X to obtain G_0. A terminal strongly connected component in G_0 is $\{i_1\}$, and only p2 labels internal edges of this component. Hence, we put $I_1 = \{i_1\}$, $k(1) = 1$. To find I_2, delete from G_0 the node i_1 and all the edges going out of i_0 (some of these edges lead to I_1, while the other edges are siblings of such edges in the sense defined above). G_1 thus has the form shown in Figure 3, from which one easily obtains $I_2 = \{i_0\}$, $k(2) = 1$; $I_3 = \{i_3\}$, $k(3) = 2$; and $I_4 = \{i_4\}$, $k(4) = 2$.

Again, we stress the point that this construction and its consequences do not depend on the particular values of nonzero transition probabilities used in the randomization statements in the processes involved.

Example 2. Consider the following two-process synchronization protocol. Each process $i = 1, 2$ has a variable c_i that it can set and the other process can examine. The code for pi is

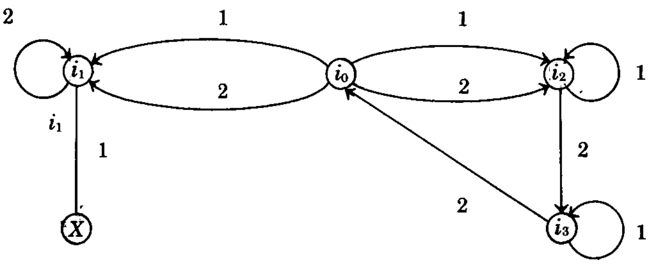
```
Try: $c_i$ := Random(0, 1);
     if $c_i > c_j$ then go to Ex        /* $j$ is the other process */
               else go to Try
     fi;
Ex:  —critical region—
     $c_i$ := 0; go to Try;
```

Denote a state by the quadruple (c_1, c_2, l_1, l_2) where l_i is the location in process $i = 1, 2$, and which can be either T (Try) or X (Ex). Let $\hat{\imath} = (0, 0, \mathrm{T}, \mathrm{T})$. Let X be the set of all states at which p1 is at its critical region. It is easily checked that $\hat{I}$ contains the following states:

$$\hat{\imath} = i_1 = (0, 0, \mathrm{T}, \mathrm{T});$$
$$i_2 = (0, 1, \mathrm{T}, \mathrm{T});$$
$$i_3 = (1, 0, \mathrm{T}, \mathrm{T});$$
$$i_4 = (1, 1, \mathrm{T}, \mathrm{T});$$
$$i_5 = (0, 1, \mathrm{T}, \mathrm{X});$$
$$i_6 = (1, 1, \mathrm{T}, \mathrm{X}).$$

The relevant portions of the transition matrices are then as shown in Figure 4.

Figure 2



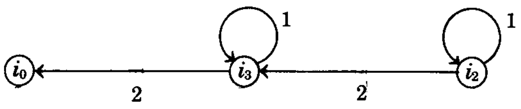


Figure 3

Figure 4

	P^1							P^2						
	i_1	i_2	i_3	i_4	i_5	i_6	X	i_1	i_2	i_3	i_4	i_5	i_6	X
i_1	$\frac{1}{2}$						$\frac{1}{2}$	$\frac{1}{2}$				$\frac{1}{2}$		
i_2		$\frac{1}{2}$		$\frac{1}{2}$				$\frac{1}{2}$				$\frac{1}{2}$		
i_3	$\frac{1}{2}$						$\frac{1}{2}$			$\frac{1}{2}$	$\frac{1}{2}$			
i_4		$\frac{1}{2}$		$\frac{1}{2}$						$\frac{1}{2}$	$\frac{1}{2}$			
i_5					$\frac{1}{2}$	$\frac{1}{2}$		1						
i_6					$\frac{1}{2}$	$\frac{1}{2}$				1				

Applying the algorithm described above, we find that the whole set $\hat{I}$ is K-ergodic with $K_{\hat{I}}(i) = \{1, 2\}$ for $i \neq i_1, i_3$, and $K_{\hat{I}}(i_1) = K_{\hat{I}}(i_3) = \{2\}$. Hence p1 can be locked out from $\hat{\imath}$ with probability one by a suitable adversary fair schedule. However, it can be shown that, if the schedule is strictly fair, there is always some positive probability of entering X from $\hat{\imath}$ (although this probability can be made as small as one wishes); the situation is quite similar to the simple example given earlier.

Remark. As can be seen from the first example, the algorithm for detection of K-ergodic sets in $\hat{I}$ can be improved as follows: First note that each transition graph G_m constructed at the mth step of the algorithm can be obtained from the previously constructed graph G_{m-1} by deleting all the nodes of I_m and all edges that correspond to process transitions that can transfer us out of $I \backslash J_m$ into I_m. More precisely, delete all transitions of the form $i \rightarrow j$ under k from G_{m-1} if $P^k_{i,I_m} > 0$. Moreover, if G_{m-1} has been partitioned into strongly connected components, then a similar decomposition of G_m would result in a refinement of the decomposition of G_{m-1}. These two observations suggest the following improvements of the algorithm:

Construct an initial relation graph G_0 and decompose it into strongly connected components. At each step of the algorithm delete from G_0 nodes and edges as explained above, and redecompose each old strongly connected component C into new components, if edges in C have been deleted. The rest of the algorithm steps remain the same.

Example 3. The following is a two-process synchronization that uses a shared variable c having only two values 1, 2. Here we do *not* have to assume indivisible

	P^1				P^2			
	i_1	i_2	i_3	X	i_1	i_2	i_3	X
i_1				1	1			
i_2		1					1	
i_3			1		$\frac{1}{2}$	$\frac{1}{2}$		

Figure 5

test-and-set operations on *c*. We assume that both processes remain live indefinitely. The code for p1 (respectively, p2) is

```
Try: if c = 1 [respectively, 2] then go to Ex
                                else go to Try fi;
Ex:  —critical region—
     c := Random(1, 2); go to Try
```

We have four states:

$$i_1 = (1, \mathrm{T}, \mathrm{T});$$

$$i_2 = (2, \mathrm{T}, \mathrm{T});$$

$$i_3 = (2, \mathrm{T}, \mathrm{X});$$

$$i_4 = (1, \mathrm{X}, \mathrm{T}).$$

Taking $X = \{i_4\}$, the transition matrices are as shown in Figure 5. Hence $I_1 = \{i_1\}$, and the other strongly connected component of G is $\{i_2, i_3\}$. At the next step of the algorithm, we delete all 2-edges from i_3, which causes $\{i_2, i_3\}$ to be broken down into two components $\{i_2\}$ and $\{i_3\}$, and we easily find that $I_2 = \{i_3\}$ and $I_3 = \{i_2\}$; and it follows that $h^*_{i,X} = 1$ for any i. Hence this protocol is lockout-free.

3. PROBABILISTIC VERSUS DETERMINISTIC PROTOCOLS

In this section we compare the capabilities of probabilistic protocols with those of deterministic ones. An example motivating this study is an algorithm given by Rabin in [9] for synchronization of N processes, which uses only an $O(\log N)$-valued shared variable on which indivisible test-and-set operations are allowed. On the other hand, it is shown in [1] that, in any deterministic synchronization algorithm for N processes using test-and-set operations on a shared variable and providing both mutual exclusion and freedom from lockout, the shared variable must have about $\sqrt{2N}$ values, and this lower bound increases to about $N/2$ values if certain natural constraints are imposed on the structure of the processes involved. The algorithm given in [9] is shown to provide mutual exclusion and to be lockout-free. However, the schedules under which this algorithm is assumed to operate form a restricted subset of the general fair schedules that we consider. It is helpful in this regard to ignore the "demonic" nature of a schedule and simply interpret it as an integral part of the execution tree. Thus the schedule does not really "use" the past history of an execution to determine the next process to be scheduled, but simply records at each execution step which process is next ready to perform an atomic action. Certain other families of schedules considered in the literature impose certain constraints on the scheduling, for example, bounding speed ratios between processes [11] or, as is the case in [9], allowing the schedule to "base its decisions" only upon partial information concerning past execution history, so that it must make the same scheduling choice for all sequences having the same partial structure. Such a restriction on the behavior of the schedule can improve the performance of a synchronization algorithm but may make it less robust, in the sense that it may fail to meet some of its requirements if more general (fair) schedules are allowed.

We show in this section that Rabin's algorithm is not lockout-free if general fair schedules are allowed. As it turns out, this fact does not depend on the particular form of the algorithm but follows from a general lower bound on the size of a shared variable necessary to ensure both mutual exclusion and freedom from lockout. Specifically, we show that lower bounds similar to those given in [1] are required for probabilistic synchronization algorithms of the same kind. This follows from a generalization of the proofs given in [1] to the probabilistic case. Thus any algorithm attempting to use fewer values for such a shared variable is doomed to failure against general fair schedules, although it may still be successful against "weaker" schedules, as is indeed the case in [9].

We begin with a brief description of Rabin's algorithm. It contains N processes, each of which consists of an infinite cycle of the following four activities: (1) being idle, that is, not wishing to use a common resource but only manipulating its internal private variables; (2) trying to enter a critical section to access the common resource; (3) being in its critical section; and (4) exiting from this section, returning to the "idle" section. A process may die out only while being idle. To synchronize their actions, the processes use a shared variable v consisting of three fields (s, b, r). Each process can, in one indivisible step, test the values of all three fields and modify some or all of them (a so-called *test-and-set* operation). The fields have the ranges $s = 0, 1$; $b = 0, \ldots, \log N + 4$; $r = 1, \ldots, M$ for some fixed integer M. s serves as a semaphore providing mutual exclusion to the critical sections; r is a "round count," each value of which designates a round of number drawings by some of the processes; and b keeps track of the maximal number drawn so far by some process participating in the present round. Each process[3] pi also has two internal variables: r_i, the most recent round count seen by pi, and b_i, the number drawn by pi in that round.

The algorithm works as follows: a *trying round* roughly designates the period between two successive entrances to critical sections by the processes. Each process pi wishing to enter its critical section is allowed to draw a number b_i in the range $0, \ldots, \log N + 4$ only once in any trying round. If $s = 0$ (i.e., nobody is at a critical section), then pi checks whether $b_i \geq b$ (i.e., pi is a "winner" in the lottery) and $r_i = r$ (i.e., b_i has been drawn in the current drawing round), and, if so, it enters its critical section, setting (s, b, r) to $(1, 0, \mathit{Random}(1 \,..\, M))$. In any case p$i$ can draw a number only if $r_i \neq r$ (it then sets r_i to r). If $s = 1$, and pi has drawn b_i, it sets b to $\max(b, b_i)$ and waits until $s = 0$. If $r_i = r$, pi does not draw but just waits until $s = 0$. However, if r is changed meanwhile by some other process entering its critical section, then pi is allowed to participate in the new drawing round. For fuller details, see [9].

The problem with this scheme is that there is only a fixed number of round counts. Hence, it may happen that some process, say p1, has recorded in r_1 an old round count, but the new round count r has the same value as r_1, and so p1 will

[3] In order to make our notation agree with that in the other papers we mention [1, 9], in this section we use i as an index for processes and q as a generic state.

wrongly disqualify itself from drawing again in the new round. This observation leads to the construction of the following fair schedule σ that can lock out p1: Let $q \in I$ be a state in which p1 is at its trying section and has drawn $b_1 = 0$ in a round $r_1 = 1$ (say) and some other process is at its critical section. σ then runs all the other processes (which we assume to remain live throughout execution; i.e., none of them remains indefinitely in its idle section) in any fair manner until the global round count r becomes 1 again and some process is at its critical section (this must eventually happen almost surely), and only then schedules p1 again. p1 will then perform a no-op, since $s = 1$ and $r = r_1$, and the same scheduling pattern is repeated again. Note that, since p1 has drawn the smallest number b_1 in state q, this cannot block the other processes from entering their critical sections; in fact, as far as the other processes are concerned, p1 might as well be idle in state q and all subsequent states. Thus σ locks p1 out of entering its critical section from state q almost surely.

However, schedules of this kind are excluded from the analysis given in [9]. The admissible schedules allowed in [9] can base their decisions only on the sequence of processes scheduled so far and on the status of each process when it has been scheduled, where a status can have only the following three values: T, indicating that the process has been at its trying section and has failed to enter its critical section at this step; TC, indicating that the process has been at its trying section and has managed to enter its critical section during this step; and E, indicating that the process has exited its critical section at this step. Such schedules are too weak to lock out a process from its critical section, since they must make the same choice for all execution sequences having the same status structure, and so cannot schedule p1 only when $r = r_1$. Hence p1 will get enough chances to draw again and will be able to master its way into its critical section almost surely (see [9] for a precise proof).

It is instructive to note the exact point in which Rabin's proof fails in the presence of general fair schedules. It is in asserting that there is a probability of $1 - 1/M$ that the new value of the round count r, set by the process that has most recently entered its critical section, is different from r_i for some process pi that will participate in the subsequent trying round (cf. [9, p. 410, 2d paragraph]). Under general fair schedules this statement is wrong because the event A = "the last randomization of r yielded a value different from r_i" is not independent of the event B = "pi participates in the following trying round."

Therefore, the probability of A given B need not be equal to the probability of $A = 1 - 1/M$ (as a matter of fact, the schedule presented above makes the former probability zero). However, for schedules of the sort considered by Rabin [9] the two events are indeed independent, since such a schedule cannot base its decisions on the value of the internal variables r and r_i. Hence for such schedules the probability of A given B is indeed $1 - 1/M$.

These results can be summarized as follows:

PROPOSITION 3.1 *The algorithm in* [9] *is almost surely lockout-free against the restricted family of* (*fair*) *schedules introduced there, but it fails to have this property if general fair schedules are allowed.*

We next show that this problem cannot be fixed by any superficial modification of the algorithm, but is a consequence of the fact that the algorithm uses too few values for its shared variable. This follows from extension to the probabilistic case of the lower bounds stated in [1] concerning the number of values for a shared variable required for proper synchronization of N processes. Specifically, we have the following theorem (see [1] for a detailed description of the assumptions of this theorem).

THEOREM 3.2 *Suppose that N processes participate in a synchronization protocol using a shared variable v on which indivisible test-and-set operations are allowed. Then at least $\sqrt{2N}$ values of v are required to ensure both mutual exclusion and freedom from lockout. Furthermore, if each process has only a single state while in its idle section (i.e., if it cannot remember any past experience), then at least $(N + 1)/2$ values for v are required.*

In proving this theorem we follow the notations and conventions used in [1]. Specifically, for each process pi, $i = 1, \ldots, N$, we denote by R_i the set of all states at which pi is at its idle section, and by C_i the set of all states at which pi is at its critical section. For each state q we denote by $V(q)$ the value of the shared variable at this state. V denotes the set of all values assumed by the shared variable.

Before we extend the proof of [1] to the probabilistic case, we need a few preliminary technical results:

We say that a property (A) is a property of *finite* execution paths if, whenever (A) holds on a path $\pi \equiv \{i_n\}_{n=0}^{\infty} \in \Omega$, then there exists $m \equiv m(\pi) \geq 0$ such that (A) holds on all paths $\pi' \equiv \{i'_n\}_{n=0}^{\infty} \in \Omega$ with $i'_n = i_n$ for all $n = 0, 1, \ldots, m$. Examples of such properties include reaching a certain set of states, scheduling a finite pattern of processes, etc. Note that the complement of a property (A) of finite paths need *not* be a property of finite paths. We write $\{(A)\}$ for the set of paths on which property (A) holds.

LEMMA 3.3 *Let (A) be a property of finite execution paths. The following statements are equivalent:*

(1) *There exists $\alpha > 0$ such that for each state q there exists a schedule $\sigma = \sigma(q)$ starting at q such that $\mu_\sigma\{(A)\} \geq \alpha$.*
(2) *There exists $\alpha > 0$ such that for each state q there exists a* fair *schedule $\sigma = \sigma(q)$ starting at q such that $\mu_\sigma\{(A)\} \geq \alpha$.*
(3) *For each state q there exists a schedule $\sigma = \sigma(q)$ starting at q such that $\mu_\sigma\{(A)\} = 1$.*
(4) *For each state q there exists a* fair *schedule $\sigma = \sigma(q)$ starting at q such that $\mu_\sigma\{(A)\} = 1$.*
(5) *For each state q there exists a schedule $\sigma = \sigma(q)$ starting at q such that $\mu_\sigma\{(A)$ holds infinitely often$\} = 1$.*
(6) *For each state q there exists a* fair *schedule $\sigma = \sigma(q)$ starting at q such that $\mu_\sigma\{(A)$ holds infinitely often$\} = 1$.*

PROOF. We have only to show (1) $\Rightarrow$ (4) and (3) $\Rightarrow$ (6), the rest being then immediate.

(1) $\Rightarrow$ (4). Starting at q, define $\bar{\sigma} = \bar{\sigma}(q)$ as follows. Use $\sigma(q)$ given by (1) for m steps, where m is large enough so that $\mu_{\sigma(q)}\{(A)$ holds in no more than m steps$\} \geq \alpha/2$. Then schedule each process pi once (to guarantee fairness), and let q' be

the new (random) state. From then on use $\sigma(q')$ (given by (1)) long enough as before, and so on. After r such iterations, the probability that (A) is not yet satisfied is no greater than

$$\left(1 - \frac{\alpha}{2}\right)^r \underset{r\to\infty}{\to} 0.$$

(3) $\Rightarrow$ (6). Define $\bar{\sigma} = \bar{\sigma}(q)$ as follows: Use $\sigma(q)$ as given by (3) until (A) holds, which happens eventually (i.e., in finite time) with probability one. Then schedule each pi once, and from the new state q' use $\sigma(q')$ (as given by (3)) as before, and so on. Q.E.D.

Recall that an infinite path π is said to be *fair* if each process pi is scheduled infinitely often along π. Nqte that the set $F \equiv F(\sigma) \subset \Omega$ of fair execution paths of a schedule σ is measurable in the cylindrical σ-field Σ on Ω.

LEMMA 3.4 *Let σ be an arbitrary (not necessarily fair) schedule, and let $\varepsilon > 0$. Then there exists a* strictly fair *schedule $\bar{\sigma}$ such that $(\mu_\sigma - \mu_{\bar{\sigma}})(A) \le \varepsilon$ for every subset $A \in \Sigma$ of the set $F \equiv F(\sigma)$ of fair execution paths of σ.*

PROOF. For every $\nu = 1, 2, \ldots$, choose m_ν large enough such that $\mu_\sigma(F \cap F_\nu) \ge \mu_\sigma(F) - \varepsilon/2^\nu$ where F_ν is the event that all processes have been scheduled at least ν times each in the first m_ν steps. Let G_ν be the complement of F_ν and $G = \bigcup_{\nu=1}^{\infty} G_\nu$; then $\mu_\sigma(F \cap G) \le \sum_{\nu=1}^{\infty} \varepsilon/2^\nu = \varepsilon$. We modify σ to obtain a new schedule $\bar{\sigma}$ as follows: For every path $\pi \in G$, let $\bar{\nu}$ be the smallest $\nu \ge 1$ such that $\pi \in G_\nu$; from step $m_{\bar{\nu}}$ on, we then schedule the processes in a round-robin way. Otherwise, we use σ. Clearly, $\bar{\sigma}$ is strictly fair, and $(\mu_\sigma - \mu_{\bar{\sigma}})(A) = (\mu_\sigma - \mu_{\bar{\sigma}})(A \cap G) \le \mu_\sigma(A \cap G) \le \mu_\sigma(F \cap G) \le \varepsilon$ for every measurable $A \subset F$. Q.E.D.

COROLLARY. *Let σ be any schedule and (B) a property of arbitrary (infinite) paths such that $\mu_\sigma\{\pi : \pi$ a fair path and satisfies $(B)\} > 0$. Then there exists a (strictly) fair schedule $\bar{\sigma}$ such that $\mu_{\bar{\sigma}}\{(B)\} > 0$.*

PROOF. Take $0 < \varepsilon < \mu_\sigma\{$fair path and $(B)\}$ in the lemma. Q.E.D.

PROOF OF THEOREM 3.2. We adapt the proofs in [1] to the probabilistic case. We show here how to obtain the first assertion of this theorem (corresponding to Theorem 4.4 in [1]). The second assertion (corresponding to Theorem 4.5 there) can be proved in a similar fashion.

For two processes, at least two different values of v (the common message variable) are needed, in the following sense: For any starting state q there exists a schedule σ under which two values of v will be used almost surely. This is shown by using exactly the same proof given in [1] for the deterministic case; see Theorem 4.1 there.

The induction hypothesis is that, given $N \ge (k^2 - k)/2 + 1$ processes[4] that are lockout-free and satisfy mutual exclusion, at least k different values of v are needed, in the following sense: There exists some $\alpha \equiv \alpha_N > 0$ such that for any starting state q there exists a schedule $\sigma \equiv \sigma(q)$ for which the probability of using at least k values of v is $\ge\alpha$. This is a stronger assertion than needed to obtain our result; it is required, however, for the induction.

We now assume that $N \ge [(k + 1)^2 - (k + 1)]/2 + 1$ processes are given, and

[4] In [1], the use of Lemma 4.2 reduces this to $(k^2 - k)/2 - 1$.

we prove that at least $k + 1$ values of v are needed, namely, that for every starting state $\bar{q}$ there exists a schedule $\sigma = \sigma(\bar{q})$ such that $\mu_\sigma\{$at least $k + 1$ values of v are used$\} \geq \alpha$ where $\alpha \equiv \alpha_N = (1/|V|)^{N-1} > 0$. In view of Lemma 3.3 (i.e., (1) $\Rightarrow$ (2) there), we can assume that σ is fair, thus proving our theorem. (As a matter of fact, Lemma 3.3 even implies that there exists a fair schedule σ under which $k + 1$ values of v will be used almost surely infinitely often.)

Let $\bar{q}$ be any starting state. We define a schedule $\sigma' \equiv \sigma'(\bar{q})$ as follows. First, we schedule only those processes pi that are not yet in R_i; we shall eventually reach (almost surely) a (random) state q_0 in $\bigcap_{i=1}^{N} R_i$ (since there is no lockout, and no process can remain indefinitely in its critical section). From q_0 we schedule p1 repeatedly until a (random) state $q_1 \in C_1$ is reached. (This will happen almost surely, since all other processes are in R_i and there is no lockout.)

Next, assume that after q_1 we schedule *only* the process p2. Since there are $|V|$ different values of v, there exists some constant value $v_2 \in V$ such that the probability of using v_2 infinitely often (when scheduling only p2 from q_1) is at least $1/|V|$. We then continue σ' from q_1 by scheduling p2 repeatedly until a state q_2 with $V(q_2) = v_2$ is reached (if at all). Thus, the (random) state q_2 materializes with probability $\geq(1/|V|)$.

From q_2 (i.e., from each of the nodes q_2 in the tree), we proceed in a similar manner with p3. That is, assuming only p3 is scheduled, there is a constant $v_3 \in V$ such that the conditional probability that v_3 is used infinitely often after q_2, given that a state q_2 has been reached, is $\geq(1/|V|)$. σ' then continues from q_2 by scheduling p3 repeatedly until a (random) state q_3, for which $V(q_3) = v_3$, is reached, if at all. (Note that the probability of reaching q_3 from $\bar{q}$ is $\geq(1/|V|^2)$.) We continue in this manner with p4, ..., pN, obtaining a sequence $\{v_i\}_{i=2}^{N} \subset V$ and, with probability at least $(1/|V|)^{N-1} \equiv \alpha$, a sequence $\{q_i\}_{i=2}^{N}$ of random states with $V(q_i) = v_i$, $i = 2, \ldots, N$. To be more specific, for each $i = 2, 3, \ldots, N$, let β_i^0 be the (conditional) probability of reaching q_i (i.e., a state at which $v = v_i$) given that we have already reached q_{i-1} (i.e., we consider the forest collection of subtrees whose roots are all the nodes q_{i-1}), under repeated scheduling of pi only. For $r = 1, 2, \ldots$, let β_i^r be the (conditional) probability that, under repeated scheduling of pi only, v_i will be used $r + 1$ times, given that it has already been used r times from q_{i-1}. Then our construction implies that, for all $i = 2, \ldots, N$,

$$\prod_{r=0}^{\infty} \beta_i^r \geq \frac{1}{|V|}.$$

Two cases are now possible. If all the values v_i, for $N - k \leq i \leq N$, are different from each other, then our inductive assertion follows immediately, since, under σ', the probability of using $k + 1$ values of v is greater than or equal to the probability of reaching $q_N \geq \alpha$. Otherwise, let $N - k \leq j < l \leq N$ be such that $v_j = v_l$. We then construct a schedule $\sigma \equiv \sigma(\bar{q})$, which will use $k + 1$ different values of v with probability at least α, as follows.

We begin σ by scheduling processes according to σ' (as defined above) until the random state q_l is reached (if at all). For any such state q_l, let $\sigma''(q_l)$ be a schedule using only p1, ..., pj and having the following two properties (almost surely):

(1) σ'' is fair for p1, ..., pj; that is, each one of p1, ..., pj is scheduled infinitely often.
(2) At least k values of v are used infinitely often.

Such a schedule exists by our induction hypothesis together with (1) $\Leftrightarrow$ (6) in Lemma 3.3, since $j \geq N - k \geq (k^2 - k)/2 + 1$.

The schedule σ proceeds after reaching q_l as follows: σ schedules p1, ..., pj exactly as $\sigma''(q_l)$. However, every time a state q with $V(q) = v_i$ for some $j + 1 \leq i \leq l$ is reached, σ schedules pi repeatedly until v_i is attained once more (i.e., another state q' with $V(q') = V(q) = v_i$ is reached), if at all. After this q', the schedule σ continues according to $\sigma''(q_l)$ (from the place it stopped). Note that, as far as the processes p1, ..., pj are concerned, the states q and q' are identical, since they can differ only in the internal state of pi where $i > j$. Hence, as long as the "interrupts" that σ takes to schedule such processes pi terminate, the behavior of p1, ..., pj under σ is identical to their behavior under σ''.

Similarly, subject to the termination of these interrupts, for each $i > j$ the overall behavior of process pi under σ is identical to its behavior under repeated scheduling of only itself from state q_i. In particular, it never reaches a state in C_i (since p1 is at its critical section at q_i and, by mutual exclusion, no execution of pi alone can get it to its critical region from q_i).

Let Λ be the set of all infinite execution paths π occurring under σ such that

(3) $q_1, q_2, \ldots, q_l$ are all defined along π;
(4) π is fair for processes p1, ..., pj;
(5) at least k values of v appear infinitely often along π;
(6) each state q with $V(q) = v_i$ for some $j < i \leq l$, at which σ begins to schedule pi repeatedly, is followed (eventually) by another state q' with $V(q') = V(q) = v_i$ (from which σ continues according to σ'').

By our construction of σ, it follows that (3) and (6) imply (4) and (5) almost surely, so that

$$\mu_\sigma(\Lambda) = \mu_\sigma\{(3) \text{ and } (6)\}.$$

For every $\pi \in \Lambda$ and $j < i \leq l$, consider the concatenation of all the segments of π corresponding to the interrupts that σ takes to schedule pi; it is equivalent to an execution sequence of pi alone, starting at q_i. Let M_i be the number of these segments (M_i may be infinite). Therefore, whatever $M_{j+1}, \ldots, M_l$ are, we obtain that the conditional probability that π satisfies (6), given that it satisfies (3) and given the M_i's, is

$$\begin{aligned}\mu_\sigma\{(6) \mid (3) \text{ and } M_{j+1}, \ldots, M_l\} &\geq \prod_{i=j+1}^{l} \left(\prod_{r=1}^{M_i} \beta_i^r \right)\\ &\geq \prod_{i=j+1}^{l} \left(\prod_{r=1}^{\infty} \beta_i^r \right) \equiv \gamma.\end{aligned}$$

Hence

$$\mu_\sigma\{(6) \mid (3)\} \geq \gamma$$

and

$$\begin{aligned}\mu_\sigma(\Lambda) &= \mu_\sigma\{(3) \text{ and } (6)\} = \mu_\sigma\{(6) \mid (3)\} \cdot \mu_\sigma\{(3)\}\\ &\geq \gamma \cdot \prod_{i=2}^{l} \beta_i^0 \geq \prod_{i=2}^{l} \left(\prod_{r=0}^{\infty} \beta_i^r \right) \geq \left(\frac{1}{|V|} \right)^{l-1} \geq \alpha.\end{aligned}$$

Let Ξ be the set of all paths π such that

(7) no more than k values of v appear on π.

We want to show that $\mu_\sigma(\Xi^c) \geq \alpha$; we do this by proving that $\mu_\sigma(\Lambda \cap \Xi) = 0$. Indeed, assume $\pi \in \Lambda \cap \Xi$, and let $V' \equiv V'(\pi) \subset V$ be the set of values of v appearing on π; thus, $|V'| \leq k$ by (7). By (5), $|V'| = k$, and, furthermore, each $v \in V'$ appears infinitely often along π. For every i with $j + 1 \leq i \leq l$, $v_i \in V'$ (since q_i is a state along π by (3)); hence v_i appears infinitely often, and therefore pi is scheduled infinitely often on π (by (6)). Together with (4), this implies that π is fair for p1, ..., pl. However, it follows from an earlier comment that π locks out p$(j+1)$, ..., pl. By Lemma 3.4, if $\mu_\sigma(\Lambda \cap \Xi) > 0$, there exists another *fair* schedule with a positive probability of locking out p$(j+1)$, ..., pl, which contradicts our assumptions. Therefore, $\mu_\sigma(\Lambda \cap \Xi) = 0$, and $\mu_\sigma(\Xi^c) \geq \mu_\sigma(\Lambda) \geq \alpha$. This proves the induction step and, consequently, our theorem. Q.E.D.

Under what circumstances does randomization really help us to obtain "better" algorithms? The above negative result shows that we cannot hope to save space by incorporating randomization into synchronization protocols of the kind discussed in [1] and [9] (if we admit general fair schedules). We next give another negative result. Consider a synchronization protocol involving $N \geq 2$ identical processes, all of which use a common variable v that they may read from, or write into, in one indivisible step (all processes access v in exactly the same manner). We call such a protocol *fully symmetric*, meaning that all processes involved are indistinguishable from each other and their initial internal states are all identical. Thus, the actions that pi takes at a state q are a function of its internal state and of the value of the common variable v only; this function is the same for all processes pi. It is well known that no deterministic solution to this problem that provides mutual exclusion can exist. Surprisingly enough, an analogous proof shows that a similar result holds in the probabilistic case as well:

THEOREM 3.5 *There does not exist a probabilistic fully symmetric protocol for $N > 2$ identical processes using a shared variable on which separate read and write operations are allowed that ensures both mutual exclusion and freedom from lockout with probability one.*

PROOF. Suppose that there exists such a protocol. We assume that no lockout can occur with positive probability. Hence, if we start from a symmetric configuration q in which all processes are idle, and we let only one process (say p1) execute by itself and leave the other processes in their idle state, p1 must eventually enter its critical section with probability one. This implies that there exists at least one finite execution path π in which p1 executes alone, and all other processes remain idle, such that all transitions on π have positive probabilities and such that p1 gets from its idle section to its critical section. Let the product of the transition probabilities along π be $p > 0$.

Let us now execute the protocol as follows. Start at the symmetric configuration q, and assume that all processes become active (i.e., get out of their idle section) together. Schedule the processes in a round-robin fashion, letting each of them perform one atomic action at its turn. We show that there exists an execution path (having positive probability) such that, at the end of each round, the configuration is again symmetric. Assume that this were the case at the beginning of some round r. Then each process is at the same internal state and so performs the same action in its turn. If this action does not involve randomization, then either all the processes in this round manipulate their internal variables in the same way, or they all read the (same value of the) shared variable, or they all write the same value into the shared variable. Hence, in either case, their final configuration is again symmetric. Note that, in this mode of scheduling, each process is not aware that other processes are also executing, so that each process will follow a sequence of states identical to those that p1 passed by itself along the sequence π (as long as there is no randomization involved). Now suppose that the common action of the processes in round r did involve randomization. Then, with some positive probability $\alpha_r > 0$, p1 will make the choice that will keep it in the sequence π, and with the same probability each other process will make a symmetrically identical choice. Consequently, with probability α_r^N (where N is the number of processes involved) all processes will make this common choice and will again reach a symmetric configuration; moreover, in this new configuration each process is unaware of the existence of the other processes, and its state is the symmetric image of the rth state of p1 on π.

Continuing in this manner, with probability $p^N > 0$ all processes will enter their critical sections together after $|\pi|$ steps, since p1 does so and the behavior of the other processes remains symmetric to that of p1 under this particular execution sequence. This contradicts our assumptions and so proves the theorem. Q.E.D.

4. CONCLUSIONS

In this paper we have presented a decision procedure for verification of almost-sure convergence (or termination) of concurrent probabilistic programs having a finite state space. This procedure either (1) constructs a decomposition of the state space into a sequence of components having the property that any fair execution of the program must eventually (almost surely) move down this sequence until it reaches a desired goal state (in which case almost-sure convergence is assured) or (2) finds an *ergodic* set of states through which the program can loop forever almost surely. This procedure generalizes the proof method given in [7] for nonprobabilistic programs, provided that the state space is finite. For the generalization of the theory developed in this paper to infinite state spaces, see [4].

The second part of this paper has exhibited certain negative results concerning the power of probabilistic synchronization protocols versus the power of deterministic ones. Two instances in which both kinds of protocols have the same power have been presented. One concerns the size of a shared variable, subject to test-and-set operations, necessary to provide mutual exclusion and freedom from lockout, and the other concerns the nonexistence of symmetric protocols, using a shared variable on which separate read and write operations are allowed and having the same properties as in the preceding case.

However, there are certain known situations in which probabilistic methods do behave better than deterministic ones for certain concurrent problems. Such methods are given in [6, 8]. These phenomena call for further study to understand better the distinction between those concurrent problems that admit probabilistic solutions that are better than deterministic solutions, and those problems that do not benefit from introduction of randomization.

Another issue that deserves further study and clarification is the dependence of the performance of a probabilistic concurrent algorithm on the class of schedules under which it is assumed to operate. As shown in this paper, this dependence can be quite significant in certain cases.

REFERENCES

1. BURNS, J.E., FISCHER, M.J., JACKSON, P., LYNCH, N.A., AND PETERSON, G.L. Shared data requirements for implementation of mutual exclusion using a test-and-set primitive. In Proceedings of the 1978 International Conference on Parallel Processing, Aug. 22–25, 1978, pp. 79–87.
2. DUBINS, L.E., AND SAVAGE, L.J. *Inequalities for Stochastic Processes: How to Gamble If You Must.* Dover, New York, 1976.
3. FRANCEZ, N., AND RODEH, M. A distributed data type implemented by a probabilistic communication scheme. In Proceedings, 21st Symposium on the Foundations of Computer Science, 1980, pp. 373–379.
4. HART, S., AND SHARIR, M. Concurrent probabilistic programs, or: How to schedule if you must. Tech. Rep., School of Mathematical Sciences, Tel Aviv Univ., Tel Aviv, Israel, May 1982.
5. HART, S., SHARIR, M., AND PNUELI, A. Termination of probabilistic concurrent programs. In Conference Record of the Ninth Annual ACM Symposium on Principles of Programming Languages, Albuquerque, N.M., Jan. 25–27, 1982, pp. 1–6.
6. ITAI, A., AND RODEH, M. The lord of the ring, or probabilistic methods for breaking symmetry in distributive networks. Tech. Rep. RJ 3110, IBM Corp., San Jose, Calif., 1981.
7. LEHMANN, D., PNUELI, A., AND STAVI, J. Impartiality, justice and fairness: The ethics of concurrent termination. In *Lecture Notes in Computer Science*, vol. 115: *Automata, Languages and Programming; Eighth Colloquium, Acre (Akko), Israel, July 13–17, 1981*, S. Even and O. Kariv (Eds.). Springer-Verlag, New York, 1981, pp. 264–277.
8. LEHMANN, D., AND RABIN, M.O. On the advantages of free choice: A symmetric and fully distributed solution to the dining philosophers problem. In Conference Record of the Eighth Annual ACM Symposium on Principles of Programming Languages, Williamsburg, Va., Jan. 26–28, 1981, pp. 133–138.
9. RABIN, M.O. N process synchronization by a (4 log N)-valued shared variable. In Proceedings, 21st Symposium on the Foundations of Computer Science, 1980, pp. 407–410.
10. RABIN, M.O. The choice coordination problem. *Acta Inf. 17* (1982), 121–134.
11. REIF, J., AND SPIRAKIS, P. Distributed algorithms for synchronizing interprocess communication within real time. In Conference Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing, Milwaukee, Wis., May 11–13, 1981, pp. 133–145.
12. SHARIR, M., PNUELI, A., AND HART, S. The verification of probabilistic programs. *SIAM J. Comput.*, to appear.

Received March 1982; revised October 1982; accepted November 1982