Proving Time Bounds for Randomized Distributed Algorithms *

Nancy Lynch Isaac Saias Roberto Segala
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

A method of analyzing time bounds for randomized distributed algorithms is presented, in the context of a new and general framework for describing and reasoning about randomized algorithms. The method consists of proving auxiliary statements of the form $U \xrightarrow[p]{t} U'$, which means that whenever the algorithm begins in a state in set U, with probability p, it will reach a state in set U' within time t. The power of the method is illustrated by its use in proving a constant upper bound on the expected time for some process to reach its critical region, in Lehmann and Rabin's Dining Philosophers algorithm.

1 Introduction

Randomization has proved to be a useful tool in the design of distributed algorithms, sometimes yielding efficient solutions to problems that are inherently complex, or even unsolvable, in the setting of deterministic algorithms [1, 2, 8, 9]. But this powerful tool has a price: even simple randomized algorithms can be extremely hard to verify and analyze. Because of this, many randomized distributed algorithms appear in the literature with only informal proofs of correctness, and only informal derivation of complexity bounds. In fact, it is sometimes hard for the reader to ascertain that the proofs and complexity bounds presented are really correct. Even where proofs are carefully and correctly done, the arguments tend to be ad hoc.

A key difficulty in reasoning about randomized algorithms is the fact that their executions usually contain a combination of nondeterministic and probabilistic choices, with subtle interactions between them. The probabilistic choices are typically only those that involve an explicit use of randomness by the algorithm (e.g., by using a random-number generator). All other choices (e.g., the order of process steps, the times at which requests arrive) are usually considered to be nondeterministic. It is customary to define an *adversary* as a way of modeling the entity that resolves the nondeterministic choices.[1] In defining an adversary, one must be especially careful to specify the *knowledge* of the execution that the adversary is permitted to use in resolving nondeterministic choices. This might range from no knowledge at all, in which case the adversary is said to be *oblivious*, to complete knowledge of the past execution (including past random choices).

Even after one has defined the desired notion of adversary, it is still not easy to carry out correctness proofs and complexity analyses for randomized algorithms; most existing proofs seem rather ad hoc. It would be useful to have a collection of general proof rules and methods, which could be established once and for all, and then applied in a reasonably systematic way to verify and analyze numerous algorithms. Some examples of work that has already been done on the development of such methods are [4, 6, 11, 12]. The work of [11] presents a technique, based on progress functions defined on states, for establishing liveness properties for randomized algorithms; the work of [12] extends UNITY [3] to handle probability, and provides a completeness result for some properties that hold with probability 1; the work of [4, 6] presents model checking techniques.

In this paper, we present such a new method: a way of proving *upper bounds on time* for randomized algorithms. Our method consists of proving auxiliary statements of the form $U \xrightarrow[p]{t} U'$, which means that whenever the algorithm begins in a state in set U, with probability p, it will reach a state in set U' within time t. Of course, this method can only be used for randomized algorithms that include timing assumptions. A key theorem about our method is the composability of these $U \xrightarrow[p]{t} U'$ arrows, as expressed by Theorem 3.4. This composability result holds in the case of (many classes of) oblivious as well as non-oblivious adversaries.

We also present two complementary proof rules that help in reasoning about sets of distinct random choices. Independence arguments about such choices are often crucial to correctness proofs, yet there are subtle ways in which a non-oblivious adversary can introduce dependencies. For example, a non-oblivious adversary has the power to use the outcome of one random choice to decide whether to schedule another random choice. Our proof rules help to systematize certain kinds of reasoning about independence.

Our proof rules are presented in the context of a new

*Supported by NSF grant CCR-89-15206, and CCR-92-25124, by DARPA contracts N00014-89-J-1988 and N00014-92-J-4033, and by ONR contract N00014-91-J-1046.

[1] In this paper, we ignore the possibility that the adversary itself uses randomness.

and general formal framework [14] for describing and reasoning about randomized algorithms. This framework integrates randomness and nondeterminism into one model, and permits the modeling of timed as well as untimed systems. The model of [14] is, in turn, based on existing models for untimed and timed distributed systems [7, 10], and adopts many ideas from the probabilistic models of [6, 16].

In order to illustrate our method, we use it in this paper to prove an upper bound for Lehmann and Rabin's Dining Philosophers algorithm [9], in the face of an adversary with complete knowledge of the past. This upper bound asserts that $\mathcal{T} \xrightarrow[1/8]{13} \mathcal{C}$, where $\mathcal{T}$ is the set of states in which some process is in its trying region, while $\mathcal{C}$ is the set of states in which some process is in its critical region. That is, whenever the algorithm is in a state in which some process is in the trying region, with probability 1/8, within time 13, it will reach a state in which some process is in its critical region. This bound depends on the timing assumption that processes never wait more then time 1 between steps. A consequence of this claim is an upper bound (of 63) on the expected time for some process to reach its critical region.

For comparison, we note that [9] contains only proof sketches of the results claimed. The paper [11] contains a proof that Lehmann and Rabin's algorithm satisfies an *eventual* progress condition, in the presence of an adversary with complete knowledge of the past; this proof is carried out as an instance of Pnueli and Zuck's general method, based on a notion of *extreme fairness*, for proving liveness properties. Our results about this protocol can be regarded as a refinement of the results of Pnueli and Zuck, in that we obtain explicit constant time bounds rather than liveness properties. It is worth noting that the method based on extreme fariness can be embedded in our model. Similarly, the state chart representation that is used in [11] can be adapted to our proof method.

The rest of the paper is organized as follows. Section 2 presents a simplified version of the model of [14]. Section 3 presents our main proof technique based on time-bound statements. Section 4 presents the additional proof rules for independence of distinct probabilistic choices. Section 5 presents the Lehmann-Rabin algorithm. Section 6.2 formalizes the algorithm in terms of the model of Section 2, and gives an overview of our time bound proof, and Section 7 contains the details of the time bound proof. Section 8 gives some concluding remarks.

2 The Model

In this section, we present the model that is used to formulate our proof technique. It is a simplified version of the probabilistic automaton model of [14]. Here we only give the parts of the model that we need to describe our proof method and its application to the Lehmann-Rabin algorithm; we refer the reader to the full version of this paper and to [14] for more details.

Definition 2.1 A *probability space* is a triplet $(\Omega, \mathcal{F}, P)$ where Ω is a set, $\mathcal{F}$ is a collection of subsets of Ω that is closed under complement and countable union and such that $\Omega \in \mathcal{F}$, and P is a function from $\mathcal{F}$ to $[0,1]$ such that $P[\Omega] = 1$ and for any collection $\{C_i\}_i$ of at most countably many pairwise disjoint elements of $\mathcal{F}$, $P[\cup_i C_i] = \sum_i P[C_i]$.

A probability space $(\Omega, \mathcal{F}, P)$ is *discrete*[2] if $\mathcal{F} = 2^\Omega$ and for each $C \subseteq \Omega$, $P[C] = \sum_{x \in C} P[\{x\}]$. It is immediate to verify that for every discrete probability space there are at most countably many points with a positive probability measure. ∎

Definition 2.2 A *probabilistic automaton*[3] M consists of four components:

- a set $states(M)$ of states.
- a nonempty set $start(M) \subseteq states(M)$ of start states.
- an action signature $sig(M) = (ext(M), int(M))$ where $ext(M)$ and $int(M)$ are disjoint sets of external and internal actions, respectively.
- a transition relation $steps(M) \subseteq states(M) \times acts(M) \times Probs(states(M))$, where the set $Probs(states(M))$ is the set of discrete probability spaces $(\Omega, \mathcal{F}, P)$ such that $\Omega \subseteq states(M)$. Discreteness is needed for technical convenience.

A probabilistic automaton is *fully probabilistic* if it has a unique start state and from each state there is at most one step enabled. ∎

Thus, a probabilistic automaton is a state machine with a labeled transition relation such that the state reached during a step is determined by some probability distribution. For example, the process of flipping a coin is represented by a step labeled with an action `flip` where the next state contains the outcome of the coin flip and is determined by a probability distribution over the two possible outcomes. A probabilistic automaton also allows nondeterministic choices over steps. An example of nondeterminism is the choice of which process takes the next step in a multi-process system.

A reader familiar with other models for randomized computation may be puzzled because we do not put any restriction on the size of Ω in our probability spaces. The reader may assume that Ω finite without compromising the understanding of the paper; however, such a restriction is not necessary for any of our results, and thus we do not include it in Definition 2.2.

An *execution fragment* α of a probabilistic automaton M is a (finite or infinite) sequence of alternating states and actions starting with a state and, if the execution fragment is finite, ending in a state, $\alpha = s_0 a_1 s_1 a_2 s_2 \cdots$, where for each i there exists a probability space $(\Omega, \mathcal{F}, P)$ such that $(s_i, a_{i+1}, (\Omega, \mathcal{F}, P)) \in steps(M)$ and $s_{i+1} \in \Omega$. Denote by $fstate(\alpha)$ the first state of α and, if α is finite, denote by $lstate(\alpha)$ the last state of α. Furthermore, denote by $frag^*(M)$ and $frag(M)$ the sets of finite and all execution fragments of M, respectively. An *execution* is an execution fragment whose first state is a start state. Denote by $exec^*(M)$ and $exec(M)$ the sets of finite and all executions of M, respectively. A state s of M is *reachable* if there exists a finite execution of M that ends in s. Denote by $rstates(M)$ the set of reachable states of M.

A finite execution fragment $\alpha_1 = s_0 a_1 s_1 \cdots a_n s_n$ of M and an execution fragment $\alpha_2 = s_n a_{n+1} s_{n+1} \cdots$ of M can

[2] If we accept the Axiom of Choice, then the requirement $\mathcal{F} = 2^\Omega$ is sufficient.

[3] In [14] the probabilistic automata of this definition are called *simple probabilistic automata*. This is because [14] also includes the case of randomized adversaries.

be *concatenated*. The concatenation, written $\alpha_1 \frown \alpha_2$, is the execution fragment $s_0 a_1 s_1 \cdots a_n s_n a_{n+1} s_{n+1} \cdots$. An execution fragment α_1 of M is a *prefix* of an execution fragment α_2 of M, written $\alpha_1 \leq \alpha_2$, if either $\alpha_1 = \alpha_2$ or α_1 is finite and there exists an execution fragment α_1' of M such that $\alpha_2 = \alpha_1 \frown \alpha_1'$.

In order to study the probabilistic behavior of a probabilistic automaton, some mechanism to remove nondeterminism is necessary. To give an idea of why the nondeterministic behavior should be removed, consider a probabilistic automaton with three states s_0, s_1, s_2 and with two steps enabled from its start state s_0; the first step moves to state s_1 with probability $1/2$ and to s_2 with probability $1/2$; the second step moves to state s_1 with probability $1/3$ and to s_2 with probability $2/3$. What is the probability of reaching state s_1? The answer depends on how the nondeterminism between the two steps is resolved. If the first step is chosen, then the probability of reaching state s_1 is $1/2$; if the second step is chosen, then the probability of reaching state s_1 is $1/3$. We call the mechanism that removes the nondeterminism an *adversary*, because it is often viewed as trying to thwart the efforts of a system to reach its goals. In distributed systems the adversary is often called the *scheduler*, because its main job may be to decide which process should take the next step.

Definition 2.3 An *adversary* for a probabilistic automaton M is a function $\mathcal{A}$ taking a finite execution fragment of M and giving back either nothing (represented as δ) or one of the enabled steps of M if there are any. Denote the set of adversaries for M by $Advs(M)$[4]. ■

Once an adversary is chosen, a probabilistic automaton can run under the control of the chosen adversary. The result of the interaction is called an execution automaton. The definition of an execution automaton, given below, is rather complicated because an execution automaton must contain all the information about the different choices of the adversary, and thus the states of an execution automaton must contain the complete history of a probabilistic automaton. Note that there are no nondeterministic choices left in an execution automaton.

Definition 2.4 An *execution automaton* H of a probabilistic automaton M is a fully probabilistic automaton such that

1. $states(H) \subseteq frag^*(M)$. Denote a generic state of H by $q, q_1, q_2, \ldots$
2. for each step $(q, a, (\Omega, \mathcal{F}, P))$ of H there exists a step $(lstate(q), a, (\Omega', \mathcal{F}', P'))$ of M, called the corresponding step, such that $\Omega = \{qas | s \in \Omega'\}$ and $P'[qas] = P[s]$ for each $s \in \Omega'$.
3. each state of H is reachable, i.e., for each $q \in states(H)$ there exists an execution of H leading to state q. ■

Definition 2.5 Given a probabilistic automaton M, an adversary $\mathcal{A} \in Advs(M)$, and an execution fragment $\alpha \in frag^*(M)$, the execution $H(M, \mathcal{A}, \alpha)$ of M under adversary $\mathcal{A}$ with starting fragment α is the execution automaton of M whose start state is α and such that

1. for each state q there is a step enabled iff $\mathcal{A}(q) \neq \perp$;
2. for each state q, if there is a step $(q, a, (\Omega, \mathcal{F}, P)) \in steps(H(M, \mathcal{A}, \alpha))$, then its corresponding step is $\mathcal{A}(q)$. ■

Given an execution automaton H, an event is expressed by means of a set of maximal executions of H, where a maximal execution of H is either infinite, or it is finite and its last state does not enable any step in H. For example, the event "eventually action a occurs" is the set of maximal executions of H where action a does occur. A more formal definition follows. The sample space Ω_H is the set of maximal executions of H. The σ-algebra $\mathcal{F}_H$ is the smallest σ-algebra that contains the set of *rectangles* R_α, consisting of the executions of Ω_H having α as a prefix[5]. The probability measure P_H is the unique extension of the probability measure defined on rectangles as follows: $P_H[R_\alpha]$ is the product of the probabilities of each step of H generating α. In [14] it is shown that there is a unique probability measure having the property above, and thus $(\Omega_H, \mathcal{F}_H, P_H)$ is a well defined probability space. A similar probability space is built in [6, 16] for automata with a finitely branching transition relations. For the rest of this paper we do not need to refer to this formal definition any more.

Events of $\mathcal{F}_H$ are not sufficient for the analysis of a probabilistic automaton. Events are defined over execution automata, but a probabilistic automaton may generate several execution automata depending on the adversary it interacts with. Thus a more general notion of event is needed that can deal with all execution automata. Specific examples are given in Section 3.

Definition 2.6 An event schema e for a probabilistic automaton M is a function associating an event of $\mathcal{F}_H$ with each execution automaton H of M. ■

We now discuss briefly a simple way to handle time within probabilistic automata. The idea is to add a time component to the states of a probabilistic automaton (denote the time component of a state s by $s.now$), to assume that the time at a start state is 0, to add a special nonvisible action ν modeling the passage of time, and to add arbitrary time passage steps to each state. A time passage step should be non-probabilistic and should change only the time component of a state by adding any arbitrary positive real value to it. This construction is called the *patient* construction in [5, 15]. At this point the theory for the timed model works in the same identical way as the theory for the untimed model. A technical detail arises in the definition of an execution automaton, where the start state q_0 is forced to have time 0 and the time of each other state q is defined to be $lstate(q).now - lstate(q_0).now$. The timed model can be generalized even further in the same style as in [10], and the theory for the untimed model applies directly as well. The only restriction that we need to impose is that all the time passage steps are non-probabilistic. The reader is referred to [14] for more details about the general model.

We close this section with one final definition. Our time bound property for the Lehmann-Rabin algorithm states that if some process is in its trying region, then no matter how the steps of the system are scheduled, some process enters its critical region within time t with probability at least

[4] In [14] the adversaries of this definition are denoted by $DAdvs(M)$, where D stands for *Deterministic*. The adversaries of [14] are allowed to use randomness.

[5] Note that a rectangle R_α can be used to express the fact that the finite execution α occurs.

p. However, this claim can only be valid if each process has sufficiently frequent chances to perform a step of its local program. Thus, we need a way to state our properties in terms of specific sets of adversaries, rather than all possible adversaries. We call any arbitrary subset of $Advs(M)$ an *adversary schema*, and we denote a generic adversary schema by *Advs*. Any scheduling policy can be defined by means of an adversary schema: it is enough to consider only those adversaries whose choices agree with the specific scheduling policy. The adversary schema $Unit - Time$ of Section 6.2 identifies any scheduling policy for which at any time each process in its trying region is given a chance to perform a step of its program within time 1.

3 The Proof Method

In this section, we introduce our key statement $U \xrightarrow[p]{t}_{Advs} U'$ and the *composability theorem*, which is our main theorem about the proof method.

The meaning of the statement $U \xrightarrow[p]{t}_{Advs} U'$ is that, starting from any state of U and under any adversary $\mathcal{A}$ of *Advs*, the probability of reaching a state of U' within time t is at least p. The suffix *Advs* is omitted whenever we think it is clear from the context.

Definition 3.1 Let $e_{U',t}$ be the event schema that, applied to an execution automaton H, returns the set of maximal executions α of H where there exists some state q with $q.now \leq t$ and $lstate(q) \in U'$. Then $U \xrightarrow[p]{t}_{Advs} U'$ iff for each $s \in U$ and each $\mathcal{A} \in Advs$, $P_{H(M,\mathcal{A},s)}[e_{U',t}(H(M,\mathcal{A},s))] \geq p$. ■

Proposition 3.2 *Let U, U', U'' be sets of states of a probabilistic automaton M.*
If $U \xrightarrow[p]{t} U'$, then $U \cup U'' \xrightarrow[p]{t} U' \cup U''$. ■

In order to compose time bound statements, we need a restriction for adversary schemas stating that the power of the adversary schema is not reduced if a prefix of the past history of an execution is not known. Most adversary schemas that appear in the literature satisfy this restriction.

Definition 3.3 An adversary schema *Advs* for a probabilistic automaton M is *execution closed* if, for each $\mathcal{A} \in Advs$ and each finite execution fragment $\alpha \in frag^*(M)$, there exists an adversary $\mathcal{A}' \in Advs$ such that for each execution fragment $\alpha' \in frag^*(M)$ with $lstate(\alpha) = fstate(\alpha')$, $\mathcal{A}'(\alpha') = \mathcal{A}(\alpha \frown \alpha')$. ■

Theorem 3.4 *Let Advs be an execution closed adversary schema for a probabilistic timed automaton M, and let U, U', and U'' be sets of states of M.*
If $U \xrightarrow[p_1]{t_1}_{Advs} U'$ and $U' \xrightarrow[p_2]{t_2}_{Advs} U''$, then $U \xrightarrow[p_1 p_2]{t_1+t_2}_{Advs} U''$.

Proof sketch. Consider an adversary $\mathcal{A} \in Advs$ that acts on M starting from a state s of U. The execution automaton $H(M,\mathcal{A},s)$ contains executions where a state from U' is reached within time t_1. Consider one of those executions α and consider the part H of $H(M,\mathcal{A},s)$ after the first occurrence of a state from U' in α. The key idea of the proof is to use execution closure of *Advs* to show that there is an adversary that generates H without knowing the finite prefix of α that lead to U', to use $U' \xrightarrow[p_2]{t_2}_{Advs} U''$ to show that in H a state from U'' is reached within time t_2 with probability at least p_2, and to integrate this last result in the computation of the probability of reaching a state from U'' in $H(M,\mathcal{A},s)$ within time $t_1 + t_2$. ■

4 Independence

Example 4.1 Consider any distributed system where each process is allowed to flip fair coins. It is common to say "If the next coin flip of process P yields *head* and the next coin flip of process Q yields *tail*, then some good property ϕ holds." Can we conclude that the probability for ϕ to hold is 1/4? That is, can we assume that the coin flips of processes P and Q are independent? The two coin flips are indeed independent of each other, but the presence of non-oblivious adversaries may introduce some dependence. An adversary can schedule process P to flip its coin and then schedule process Q only if the coin flip of process P yielded *head*. As a result, if both P and Q flip a coin, the probability that P yields *head* and Q yields *tail* is 1/2. ■

Thus, it is necessary to be extremely careful about independence assumptions. It is also important to pay attention to potential ambiguities of informal arguments. For example, does ϕ hold if process P flips a coin yielding *head* and process Q does not flip any coin? Certainly such an ambiguity can be avoided by expressing each event in a formal model.

In this section we present two event schemas that play a key role in the detailed time bound proof for the Lehmann-Rabin algorithm (cf. Section 7), and we show some partial independence properties for them. The first event schema is a generalization of the informal statement of Example 4.1, where a coin flip is replaced by a generic action a, and where it is assumed that an event contains all the executions where a is not scheduled; the second event schema is used to analyze the outcome of the first random draw that occurs among a fixed set of random draws. A consequence of the partial independence results that we show below is that under any adversary the property ϕ of Example 4.1 holds with probability at least 1/4.

Let (a, U) be a pair consisting of an action of M and a set of states of M. The event schema $\textsc{first}(a, U)$ is the function that, given an execution automaton H, returns the set of maximal executions of H where either action a does not occur, or action a occurs and the state reached after the first occurrence of a is a state of U. This event schema is used to express properties like "the i^{th} coin yields `left`". For example a can be `flip` and U can be the set of states of M where the result of the coin flip is `left`.

Let $(a_1, U_1), \ldots, (a_n, U_n)$ be a sequence of pairs consisting of an action of M and a set of states of M such that for each i, j, $1 \leq i < j \leq n$, $a_i \neq a_j$. Define the event schema $\textsc{next}((a_1, U_1), \ldots, (a_n, U_n))$ to be the function that applied to an execution automaton H gives the set of maximal executions of H where either no action from $\{a_1, \ldots, a_n\}$ occurs, or at least one action from $\{a_1, \ldots, a_n\}$ occurs and, if a_i is the first action that occurs, the state reached after the first occurrence of a_i is in U_i. This kind of event schema is used to express properties like "the first coin that is flipped yields `left`."

Proposition 4.2 *Let H be an execution automaton of a probabilistic automaton M, and let $(a_1, U_1), \ldots, (a_n, U_n)$ be pairs consisting of an action of M and a set of states of M such that for each i, j, $1 \leq i < j \leq n$, $a_i \neq a_j$. Let $p_1, \ldots, p_n$ be real numbers between 0 and 1 such that for each i, $1 \leq i \leq n$, and each step $(s, a, (\Omega, \mathcal{F}, P)) \in steps(M)$ with $a = a_i$, the probability $P[U_i \cap \Omega]$ is greater than or equal to p_i, i.e., $P[U_i \cap \Omega] \geq p_i$. Then*

1. $P_H[(\text{FIRST}(a_1, U_1) \cap \cdots \cap \text{FIRST}(a_n, U_n))(H)] \geq p_1 \cdots p_n$,
2. $P_H[\text{NEXT}((a_1, U_1), \ldots, (a_n, U_n))(H)] \geq min(p_1, \ldots, p_n)$.

■

Of course, other event schemas can be defined and other properties can be proved about them. An example of another event schema is the one that given an execution automaton H returns the set of maximal executions of H that satisfy the extreme fairness condition of [11]. An interesting direction for further research is to define a whole collection of useful event schemas, possibly through the analysis of other examples.

5 The Lehmann-Rabin Algorithm

The Lehmann-Rabin algorithm is a randomized algorithm for the Dining Philosophers problem. This problem involves the allocation of n resources among n competing processes arranged in a ring. The resources are considered to be interspersed between the processes, and each process requires both its adjacent resources in order to reach its critical section. All processes are identical; the algorithm breaks symmetry by using randomization. The algorithm ensures the required exclusive possession of resources, and also ensures that, with probability 1, some process is always permitted to make progress into its critical region.

Figure 1 shows the code for a generic process i. The n resources are represented by n shared variables $\text{Res}_1, \ldots, \text{Res}_n$, each of which can assume values in $\{\texttt{free}, \texttt{taken}\}$. Each process i ignores its own name, i, and the names, Res_{i-1} and Res_i, of its adjacent resources. However, each process i is able to refer to its adjacent resources by relative names: $\text{Res}_{(i,\texttt{left})}$ is the resource located to the left (clockwise), and $\text{Res}_{(i,\texttt{right})}$ is the resource to the right (counterclockwise) of i. Each process has a private variable u_i, which can assume a value in $\{\texttt{left}, \texttt{right}\}$, and is used to keep track of the first resource to be handled. For notational convenience we define an operator opp that complements the value of its argument, i.e., $opp(\texttt{right}) = \texttt{left}$ and $opp(\texttt{left}) = \texttt{right}$.

The atomic actions of the code are individual resource accesses, and they are represented in the form <*atomic-action*> in Figure 1. We assume that at most one process has access to the shared resource at each time.

An informal description of the procedure is "choose a side randomly in each iteration. Wait for the resource on the chosen side, and, after getting it, just check *once* for the second resource. If this check succeeds, then proceed to the critical region. Otherwise, put down the first resource and try again with a new random choice."

Each process exchanges messages with an external user. In its idle state, a process is in its remainder region R. When triggered by a `try` message from the user, it enters the competition to get its resources: we say that it enters its trying region T. When the resources are obtained, it sends a `crit` message informing the user of the possession of these resources: we then say that the process is in its critical region C. When triggered by an `exit` message from the user, it begins relinquishing its resources: we then say that the process is in its exit region E. When the resources are relinquished its sends a `rem` message to the user and enters its remainder region.

6 Overview of the Proof

In this section, we give our high-level overview of the proof. We first introduce some notation, then sketch the proof strategy at a high level. Details of the proof appear in the Appendix.

6.1 Notation

In this section we define a probabilistic automaton M which describes the system of Section 5. We assume that process $i+1$ is on the right of process i and that resource Res_i is between processes i and $i+1$. We also identify labels modulo n so that, for instance, process $n+1$ coincides with process 1.

A state s of M is a tuple $(X_1, \ldots, X_n, \text{Res}_1, \ldots, \text{Res}_n, t)$ containing the local state X_i of each process i, the value of each resource Res_i, and the current time t. Each local state X_i is a pair (pc_i, u_i) consisting of a program counter pc_i and the local variable u_i. The program counter of each process keeps track of the current instruction in the code of Figure 1. Rather then representing the value of the program counter with a number, we use a more suggestive notation which is explained in the table below. Also, the execution of each instruction is represented by an action. Only actions $\texttt{try}_i$, $\texttt{crit}_i$, $\texttt{rem}_i$, $\texttt{exit}_i$ below are external actions.

Nr.	pc_i	Action	Informal meaning
0	R	$\texttt{try}_i$	**R**eminder region
1	F	$\texttt{flip}_i$	Ready to **F**lip
2	W	$\texttt{wait}_i$	**W**aiting for first resource
3	S	$\texttt{second}_i$	Checking for **S**econd resource
4	D	$\texttt{drop}_i$	**D**ropping first resource
5	P	$\texttt{crit}_i$	**P**re-critical region
6	C	$\texttt{exit}_i$	**C**ritical region
7	E_F	$\texttt{dropf}_i$	**E**xit: drop **F**irst resource
8	E_S	$\texttt{drops}_i$	**E**xit: drop **S**econd resource
9	E_R	$\texttt{rem}_i$	**E**xit: move to **R**eminder region

The start state of M assigns the value `free` to all the shared variables Res_i, the value R to each program counter pc_i, and an arbitrary value to each variable u_i. The transition relation of M is derived directly from Figure 1. For example, for each state where $pc_i = F$ there is an internal step $\texttt{flip}_i$ that changes pc_i into W and assigns `left` to u_i with probability $1/2$ and `right` to u_i with probability $1/2$; from each state where $X_i = (W, \texttt{left})$ there is a step $\texttt{wait}_i$ that does not change the state if $\text{Res}_{(i,\texttt{left})} = \texttt{taken}$, and changes pc_i into S and $\text{Res}_{(i,\texttt{left})}$ into `taken` if $\text{Res}_{(i,\texttt{left})} = \texttt{free}$; for each state where $pc_i = E_F$ there are two steps with action $\texttt{dropf}_i$: one step sets u_i to `right` and makes $\text{Res}_{(i,\texttt{left})}$ free, and the other step sets u_i to `left` makes $\text{Res}_{(i,\texttt{right})}$ free. The two separate steps correspond to a nondeterministic choice that is left to the adversary. For time passage steps we assume that at any point an arbitrary amount of

Shared variables: $\mathrm{Res}_j \in \{\texttt{free}, \texttt{taken}\}$, $j = 1, \dots, n$, initially `free`.

Local variables: $u_i \in \{\texttt{left}, \texttt{right}\}$, $i = 1, \dots, n$

Code for process i:

0. `try` ** beginning of Trying Section **
1. $<$ $u_i \leftarrow$ *random* $>$ ** choose `left` or `right` with equal probability **
2. $<$ **if** $\mathrm{Res}_{(i,u_i)} =$ `free` **then**
 $\mathrm{Res}_{(i,u_i)} :=$ `taken` ** pick up first resource **
 else goto 2. $>$
3. $<$ **if** $\mathrm{Res}_{(i,opp(u_i))} =$ `free` **then**
 $\mathrm{Res}_{(i,opp(u_i))} :=$ `taken`; ** pick up second resource **
 goto 5. $>$
4. $<$ $\mathrm{Res}_{(i,u_i)} :=$ `free`; goto 1. $>$ ** put down first resource **
5. `crit` ** end of Trying Section **
 ** Critical Section **
6. `exit` ** beginning of Exit Section **
7. $<$ $u_i \leftarrow$ `left` or `right` ** nondeterministic choice **
 $\mathrm{Res}_{(i,opp(u_i))} :=$ `free` $>$ ** put down first resources **
8. $<$ $\mathrm{Res}_{(i,u_i)} :=$ `free` $>$ ** put down second resources **
9. `rem` ** end of Exit Section **
 ** Remainder Section **

Figure 1: The Lehmann-Rabin algorithm

time can pass; thus, from each state of M and each positive δ there is a time passage step that increases the time component of δ and does not affect the rest of the state.

The value of each pair X_i can be represented concisely by the value of pc_i and an arrow (to the left or to the right) which describes the value of u_i. Thus, informally, a process i is in state $\underrightarrow{S}$ or $\underrightarrow{D}$ (resp. $\underleftarrow{S}$ or $\underleftarrow{D}$) when i is in state S or D while holding its right (resp. left) resource; process i is in state $\underrightarrow{W}$ (resp. $\underleftarrow{W}$) when i is waiting for its right (resp. left) resource to become free; process i is in state $\underrightarrow{E_S}$ (resp. $\underleftarrow{E_S}$) when i is in its exit region and it is still holding its right (resp. left) resource. Sometimes we are interested in sets of pairs; for example, whenever $pc_i = F$ the value of u_i is irrelevant. With the simple value of pc_i we denote the set of the two pairs $\{(pc_i, \texttt{left}), (pc_i, \texttt{right})\}$. Finally, with the symbol $\#$ we denote any pair where $pc_i \in \{W, S, D\}$. The arrow notation is used as before.

For each state $s = (X_0, \dots, X_{n-1}, \mathrm{Res}_1, \dots, \mathrm{Res}_{n-1}, t)$ of M we denote by $X_i(s)$ the pair X_i and by $\mathrm{Res}_i(s)$ the value of the shared variable Res_i in state s. Also, for any set S of states of a process i, we denote by $X_i \in S$, or alternatively $X_i = S$ the set of states s of M such that $X_i(s) \in S$. Sometimes we abuse notation in the sense that we write expressions like $X_i \in \{F, D\}$ with the meaning $X_i \in F \cup D$. Finally, we write $X_i = E$ for $X_i = \{E_F, E_S, E_R\}$, and we write $X_i = T$ for $X_i \in \{F, W, S, D, P\}$.

A first basic lemma states that a reachable state of M is uniquely determined by the local states its processes and the current time. Based on this lemma, our further specifications of state sets will not refer to the shared variables; however, we consider only reachable states for the analysis. The proof of the lemma is a standard proof of invariants.

Lemma 6.1 *For each reachable state s of M and each i, $1 \le i \le n$, $Res_i =$ `taken` iff $X_i(s) \in \{\underrightarrow{S}, \underrightarrow{D}, P, C, E_F, \underrightarrow{E_S}\}$ or $X_{i+1}(s) \in \{\underleftarrow{S}, \underleftarrow{D}, P, C, E_F, \underleftarrow{E_S}\}$. Moreover, for each reachable state s of M and each i, $1 \le i \le n$, it is not the case that $X_i(s) \in \{\underrightarrow{S}, \underrightarrow{D}, P, C, E_F, \underrightarrow{E_S}\}$ and $X_{i+1}(s) \in \{\underleftarrow{S}, \underleftarrow{D}, P, C, E_F, \underleftarrow{E_S}\}$, i.e., only one process at a time can hold one resource.* ■

6.2 Proof Sketch

In this section we show that the RL-algorithm guarantees time bounded progress, i.e., that from every state where some process is in its trying region, some process subsequently enters its critical region within an expected constant time bound. We assume that each process that is ready to perform a step does so within time 1: process i is ready to perform a step whenever it enables an action different from $\mathtt{try}_i$ or $\mathtt{exit}_i$. Actions $\mathtt{try}_i$ and $\mathtt{exit}_i$ are supposed to be under the control of the user, and hence, by assumption, under the control of the adversary.

Formally, consider the probabilistic timed automaton M of Section 6.1. Define $\mathit{Unit} - \mathit{Time}$ to be the set of adversaries $\mathcal{A}$ for M having the properties that, for every finite execution fragment α of M and every execution α' of $H(M, \mathcal{A}, \alpha)$, 1) the time in α' is not bounded and 2) for every process i and every state of α' enabling an action of process i different from $\mathtt{try}_i$ and $\mathtt{exit}_i$, there exists a step in α' involving process i within time 1. Then $\mathit{Unit} - \mathit{Time}$ is execution-closed according to Definition 3.3. An informal justification of this fact is that the constraint that each ready process is scheduled within time 1 knowing that $\alpha \frown \alpha'$ has occurred only reinforces the constraint that each ready process is scheduled within time 1 knowing that α' has occurred. Let

$$\mathcal{T} \triangleq \{s \in rstates(M) \mid \exists_i X_i(s) \in \{T\}\}$$

denote the sets of reachable states of M where some process is in its trying region, and let

$$\mathcal{C} \triangleq \{s \in rstates(M) \mid \exists_i X_i(s) = C\}$$

denote the sets of reachable states of M where some process is in its critical region. We show that

$$\mathcal{T} \xrightarrow[1/8]{13}_{Unit-Time} \mathcal{C},$$

i.e., that, starting from any reachable state where some process is in its trying region, for all the adversaries of $Unit-Time$, with probability at least $1/8$, some process enters its critical region within time 13. Note that this property is trivially satisfied if some process is initially in its critical region.

Our proof is divided into several phases, each one concerned with the property of making a partial time bounded progress toward a "success state", i.e., a state of $\mathcal{C}$. The sets of states associated with the different phases are expressed in terms of $\mathcal{T}, \mathcal{RT}, \mathcal{F}, \mathcal{G}, \mathcal{P}$, and $\mathcal{C}$. Here,

$$\mathcal{RT} \triangleq \{s \in \mathcal{T} \mid \forall_i X_i(s) \in \{E_R, R, T\}\}$$

is the set of states where at least one process is in its trying region and where no process is in its critical region or holds resources while being in its exit region.

$$\mathcal{F} \triangleq \{s \in \mathcal{RT} \mid \exists_i X_i(s) = F\}$$

is the set of states of $\mathcal{RT}$ where some process is ready to flip a coin.

$$\mathcal{P} \triangleq \{s \in rstates(M) \mid \exists_i X_i(s) = P\}$$

is the sets of reachable states of M where some process is in its pre-critical region. The set $\mathcal{G}$ is the most important for the analysis. It parallels the set of "Good Pairs"in [11] or the set described in Lemma 4 of [9]. To motivate the definition, we define the following notions. We say that a process i is *committed* if $X_i \in \{W, S\}$, and that a process i *potentially controls* Res_i (resp. Res_{i-1}) if $X_i \in \{\underrightarrow{W}, \underrightarrow{S}, \underrightarrow{D}\}$ (resp. $X_i \in \{\underleftarrow{W}, \underleftarrow{S}, \underleftarrow{D}\}$). Informally said, a state in $\mathcal{RT}$ is in $\mathcal{G}$ if and only if there is a committed process whose second resource is not potentially controlled by another process. Such a process is called a *good* process. Formally,

$$\begin{aligned} \mathcal{G} \triangleq \{s \in \mathcal{RT} \mid \exists_i \\ & X_i(s) \in \{\underleftarrow{W}, \underleftarrow{S}\} \text{ and } X_{i+1}(s) \in \{E_R, R, F, \underrightarrow{\#}\}, \text{ or} \\ & X_i(s) \in \{\underrightarrow{W}, \underrightarrow{S}\} \text{ and } X_{i-1}(s) \in \{E_R, R, F, \underleftarrow{\#}\}\} \end{aligned}$$

Reaching a state of $\mathcal{G}$ is a substantial progress toward reaching a state of $\mathcal{C}$. Actually, the proof of Proposition 7.11 establishes that, if i a is good process, then, with probability $1/4$, one of the three processes $i-1, i$ and $i+1$ soon succeeds in getting its two resources. The hard part is to establish that, with constant probability, within a *constant* time, $\mathcal{G}$ is reached from any state in $\mathcal{T}$. A close inspection of the proof given in [11] shows that, there, the timed version of the techniques used is unable to deliver this result. The phases of our proof are formally described below.

$$\begin{aligned} &\mathcal{T} \xrightarrow{2} \mathcal{RT} \cup \mathcal{C} && \text{(Proposition 7.3)}, \\ &\mathcal{RT} \xrightarrow{3} \mathcal{F} \cup \mathcal{G} \cup \mathcal{P} && \text{(Proposition 7.15)}, \\ &\mathcal{F} \xrightarrow[1/2]{2} \mathcal{G} \cup \mathcal{P} && \text{(Proposition 7.14)}, \\ &\mathcal{G} \xrightarrow[1/4]{5} \mathcal{P} && \text{(Proposition 7.11)}, \\ &\mathcal{P} \xrightarrow{1} \mathcal{C} && \text{(Proposition 7.1)}. \end{aligned}$$

The first statement states that, within time 2, every process in its exit region relinquishes its resources. By combining the statements above by means of Proposition 3.2 and Theorem 3.4 we obtain

$$\mathcal{T} \xrightarrow[1/8]{13} \mathcal{C},$$

which is the property that was to be proven. Using the results of the proof summary above, we can furthermore derive a constant upper bound on the expected time required to reach a state of $\mathcal{C}$ when departing from a state of $\mathcal{T}$. Note that, departing from a state in $\mathcal{RT}$, with probability at least $1/8$, $\mathcal{P}$ is reached in time (at most) 10; with probability at most $1/2$, time 5 is spent before failing to reach $\mathcal{G} \cup \mathcal{P}$ ("failure at the third arrow"); with probability at most $7/8$, time 10 is spent before failing to reach $\mathcal{P}$ ("failure at the fourth arrow"). If failure occurs, then the state is back into $\mathcal{RT}$. Let V denote a random variable satisfying the following induction

$$V = 1/8 \cdot 10 + 1/2\,(5 + V_1) + 3/8\,(10 + V_2)\ ,$$

where V_1 and V_2 are random variables having the same distribution as V. The previous discussion shows that the expected time spent from $\mathcal{RT}$ to $\mathcal{P}$ is at most $E[V]$. By taking expectation in the previous equation, and using that $E[V] = E[V_1] = E[V_2]$, we obtain that $E[V] = 60$ is an upper bound on the expected time spent from $\mathcal{RT}$ to $\mathcal{P}$, and that, consequently, the expected time for progress starting from a state of $\mathcal{T}$ is at most 63.

7 The Detailed Proof

In this appendix we prove the five relations used in Section 6.2. However, for the sake of clarity, we do not prove the relations in the order they were presented. Throughout the proof we abuse notation by writing expressions of the kind $\textsc{first}(\mathtt{flip}_i, \mathtt{left})$ for the event schema $\textsc{first}(\mathtt{flip}_i, \{s \in states(M) \mid X_i(s) = \underleftarrow{W}\})$.

Proposition 7.1 *If some process is in P, then, within time 1, it enters C, i.e.,*

$$\mathcal{P} \xrightarrow[1]{1} \mathcal{C}.$$

Proof. This step corresponds to the action `crit`: within time 1, process i informs the user that the critical region is free. ■

Lemma 7.2 *If some process is in its Exit region, then, within time 3, it will enter R.*

Proof. The process needs to take first two steps to relinquish its two resources, and then one step to send a `rem` message to the user. ■

Proposition 7.3 $\mathcal{T} \xrightarrow{2} \mathcal{RT} \cup \mathcal{C}$.

Proof. From Lemma 7.2 within time 2 every process that begins in E_F or E_S relinquishes its resources. If no process begins in C or enters C in the meantime, then the state reached at this point is a state of $\mathcal{RT}$; otherwise, the starting state or the state reached when the first process enters C is a state of $\mathcal{C}$. ∎

We now turn to the proof of $\mathcal{G} \xrightarrow[1/4]{5} \mathcal{P}$. The following lemmas form a detailed cases analysis of the different situations that can arise in states of $\mathcal{G}$. Informally, each lemma shows that some event of the form of Proposition 4.2 is a sub-event of the properties of reaching some other state.

Lemma 7.4

1. *Let $X_{i-1} \in \{E_R, R, F\}$ and $X_i = \underleftarrow{W}$. If* $\textsc{first}(\texttt{flip}_{i-1}, \texttt{left})$, *then, within time* 1, *either $X_{i-1} = P$ or $X_i = S$.*
2. *Let $X_{i-1} = D$ and $X_i = \underleftarrow{W}$. If* $\textsc{first}(\texttt{flip}_{i-1}, \texttt{left})$, *then, within time* 2, *either $X_{i-1} = P$ or $X_i = S$.*
3. *Let $X_{i-1} = S$ and $X_i = \underleftarrow{W}$. If* $\textsc{first}(\texttt{flip}_{i-1}, \texttt{left})$, *then, within time* 3, *either $X_{i-1} = P$ or $X_i = S$.*
4. *Let $X_{i-1} = W$ and $X_i = \underleftarrow{W}$. If* $\textsc{first}(\texttt{flip}_{i-1}, \texttt{left})$, *then, within time* 4, *either $X_{i-1} = P$ or $X_i = S$.*

Proof. The four proofs start in the same way. Let s be a state of M satisfying the respective properties of items *1* or *2* or *3* or *4*. Let f be an adversary of $Unit - Time$, and let α be the execution of M that corresponds to an execution of $H(M, \{s\}, f)$ where the result of the first coin flip of process $i-1$ is `left`.

1. By hypothesis, $i-1$ does not hold any resource at the beginning of α and has to obtain Res_{i-2} (its left resource) before pursuing Res_{i-1}. Within time 1, i takes a step in α. If $i-1$ does not hold Res_{i-1} when i takes this step, then i progresses into configuration S. If not, it must be the case that $i-1$ succeeded in getting it in the meanwhile. But, in this case, Res_{i-1} was the second resource needed by $i-1$ and $i-1$ therefore entered P.
2. If $X_i = S$ within time 1, then we are done. Otherwise, after one unit of time, X_i is still equal to $\underleftarrow{W}$, i.e., $X_i(s') = \underleftarrow{W}$ for all states s' reached in time 1. However, also process $i-1$ takes a step within time 1. Let $\alpha = \alpha_1 \frown \alpha_2$ such that the last step of α_1 is the first step taken by process $i-1$. Then $X_{i-1}(\mathit{fstate}(\alpha_2)) = F$ and $X_i(\mathit{fstate}(\alpha_2)) = \underleftarrow{W}$. Since process $i-1$ did not flip any coin during α_1, from the execution closure of $Unit - Time$ and item *1* we conclude.
3. If $X_i = S$ within time 1, then we are done. Otherwise, after one unit of time, X_i is still equal to $\underleftarrow{W}$, i.e., $X_i(s') = \underleftarrow{W}$ for all states s' reached in time 1. However, also process $i-1$ takes a step within time 1. Let $\alpha = \alpha_1 \frown \alpha_2$ such that the last step of α_1 is the first step taken by process $i-1$. If $X_{i-1}(\mathit{fstate}(\alpha_2)) = P$ then we are also done. Otherwise it must be the case that $X_{i-1}(\mathit{fstate}(\alpha_2)) = D$ and $X_i(\mathit{fstate}(\alpha_2)) = \underleftarrow{W}$. Since process $i-1$ did not flip any coin during α_1, from the execution closure of $Unit - Time$ and item *2* we conclude.
4. If $X_i = S$ within time 1, then we are done. Otherwise, after one unit of time, X_i is still equal to $\underleftarrow{W}$, i.e., $X_i(s') = \underleftarrow{W}$ for all states s' reached in time 1. However, since within time 1 process i checks its left resource and fails, process $i-1$ gets its right resource within time 1, and hence reaches at least state S. Let $\alpha = \alpha_1 \frown \alpha_2$ where the last step of α_1 is the first step of α that leads process $i-1$ to state S. Then $X_{i-1}(\mathit{fstate}(\alpha_2)) = S$ and $X_i(\mathit{fstate}(\alpha_2)) = \underleftarrow{W}$. Since process $i-1$ did not flip any coin during α_1, from the execution closure of $Unit - Time$ and item *3* we conclude. ∎

Lemma 7.5 *Assume that $X_{i-1} \in \{E_R, R, T\}$ and $X_i = \underleftarrow{W}$. If* $\textsc{first}(\texttt{flip}_{i-1}, \texttt{left})$, *then, within time* 4, *either $X_{i-1} = P$ or $X_i = S$.*

Proof. The lemma follows immediately from Lemma 7.4 after observing that $X_{i-1} \in \{E_R, R, T\}$ is equivalent to $X_{i-1} \in \{E_R, R, F, W, S, D, P\}$. ∎

The next lemma is a useful tool for the proofs of Lemmas 7.7, 7.8, and 7.9.

Lemma 7.6 *Let $X_i \in \{\underleftarrow{W}, \underleftarrow{S}\}$ or $X_i \in \{E_R, R, F, \underleftarrow{D}\}$ with* $\textsc{first}(\texttt{flip}_i, \texttt{left})$. *Furthermore, let $X_{i+1} \in \{\underrightarrow{W}, \underrightarrow{S}\}$ or $X_{i+1} \in \{E_R, R, F, \underrightarrow{D}\}$ with* $\textsc{first}(\texttt{flip}_{i+1}, \texttt{right})$. *Then the first of the two processes i or $i+1$ testing its second resource enters P after having performed this test (if this time ever comes).*

Proof. By Lemma 6.1 Res_i is free. Moreover, Res_i is the second resource needed by both i and $i+1$. Whichever tests for it first gets it and enters P. ∎

Lemma 7.7 *If $X_i = \underleftarrow{S}$ and $X_{i+1} \in \{\underrightarrow{W}, \underrightarrow{S}\}$ then, within time* 1, *one of the two processes i or $i+1$ enters P. The same result holds if $X_i \in \{\underleftarrow{W}, \underleftarrow{S}\}$ and $X_{i+1} = \underrightarrow{S}$.*

Proof. Being in state S, process i tests its second resource within time 1. An application of Lemma 7.6 finishes the proof. ∎

Lemma 7.8 *Let $X_i = \underleftarrow{S}$ and $X_{i+1} \in \{E_R, R, F, \underrightarrow{D}\}$. If* $\textsc{first}(\texttt{flip}_{i+1}, \texttt{right})$, *then, within time* 1, *one of the two processes i or $i+1$ enters P. The same result holds if $X_i \in \{E_R, R, F, D\}$, $X_{i+1} = \underrightarrow{S}$ and* $\textsc{first}(\texttt{flip}_i, \texttt{left})$.

Proof. Being in state S, process i tests its second resource within time 1. An application of Lemma 7.6 finishes the proof. ∎

Lemma 7.9 *Assume that $X_{i-1} \in \{E_R, R, T\}$, $X_i = \underleftarrow{W}$, and $X_{i+1} \in \{E_R, R, F, \underrightarrow{W}, \underrightarrow{D}\}$. If* $\textsc{first}(\texttt{flip}_{i-1}, \texttt{left})$ *and* $\textsc{first}(\texttt{flip}_{i+1}, \texttt{right})$, *then within time* 5 *one of the three processes $i-1$, i or $i+1$ enters P.*

Proof. Let s be a state of M such that $X_{i-1}(s) \in \{E_R, R, T\}$, $X_i(s) = \underleftarrow{W}$, and $X_{i+1}(s) \in \{E_R, R, F, \underrightarrow{W}, \underrightarrow{D}\}$. Let f be an adversary of $Unit - Time$, and let α be the execution of M that corresponds to an execution of $H(M, \{s\}, f)$ where the result of the first coin flip of process $i-1$ is `left` and the result of the first coin flip of process $i+1$ is `right`. By

Lemma 7.5, within time 4 either process $i-1$ reaches configuration P in α or process i reaches configuration $\underleftarrow{S}$ in α. If $i-1$ reaches configuration P, then we are done. If not, then let $\alpha = \alpha_1 \frown \alpha_2$ such that $lstate(\alpha_1)$ is the first state s' of α with $X_i(s') = \underleftarrow{S}$. If $i+1$ enters P before the end of α_1, then we are done. Otherwise, $X_{i+1}(fstate(\alpha_2))$ is either in $\{\underrightarrow{W}, \underrightarrow{S}\}$ or it is in $\{E_R, R, F, \underrightarrow{D}\}$ and process $i+1$ has not flipped any coin yet in α. From execution closure of $Unit - Time$ we can then apply Lemma 7.6: within one more time process i tests its second resource and by Lemma 7.6 process i enters P if process $i+1$ did not check its second resource in the meantime. If process $i+1$ checks its second resource before process i does the same, then by Lemma 7.6 process $i+1$ enters P. Since process i checks its second resource within time 1, process $i+1$ enters P within time 1. ■

Lemma 7.10 *Assume that* $X_{i+2} \in \{E_R, R, T\}$, $X_{i+1} = \underrightarrow{W}$, *and* $X_i \in \{E_R, R, F, \underleftarrow{W}, \underleftarrow{D}\}$. *If* $\text{FIRST}(\texttt{flip}_i, \texttt{left})$ *and* $\text{FIRST}(\texttt{flip}_{i+2}, \texttt{right})$, *then within time* 5 *one of the three processes* i, $i+1$ *or* $i+2$, *enters* P.

Proof. The proof is analogous to the one of Lemma 7.9. This lemma is the symmetric case of Lemma 7.9. ■

Proposition 7.11 *Starting from a global configuration in* $\mathcal{G}$, *then, with probability at least* $1/4$ *and within time at most* 5, *some process enters* P. *Equivalently:*

$$\mathcal{G} \xrightarrow[1/4]{5} \mathcal{P}.$$

Proof. Lemmas 7.7 and 7.8 jointly treat the case where $X_i = \underleftarrow{S}$ and $X_{i+1} \in \{E_R, R, F, \underrightarrow{\#}\}$ and the case where $X_i \in \{E_R, R, F, \underleftarrow{\#}\}$ and $X_{i+1} = \underrightarrow{S}$; Lemmas 7.9 and 7.10 jointly treat the case where $X_i = \underleftarrow{W}$ and $X_{i+1} \in \{E_R, R, F, \underrightarrow{W}, \underrightarrow{D}\}$ and the symmetric case where $X_i \in \{E_R, R, F, \underleftarrow{W}, \underleftarrow{D}\}$ and $X_{i+1} = \underrightarrow{W}$.

Specifically, each lemma shows that a compound event of the kind $\text{FIRST}(\texttt{flip}_i, x)$ and $\text{FIRST}(\texttt{flip}_j, y)$ leads to $\mathcal{P}$. Each of the basic events $\text{FIRST}(\texttt{flip}_i, x)$ has probability 1/2. From Proposition 4.2 each of the compound events has probability at least 1/4. Thus the probability of reaching $\mathcal{P}$ within time 5 is at least 1/4. ■

We now turn to $\mathcal{F} \xrightarrow[1/2]{2} \mathcal{G} \cup \mathcal{P}$. The proof is divided in two parts and constitute the global argument of the proof of progress.

Lemma 7.12 *Start with a state* s *of* $\mathcal{F}$. *If there exists a process* i *for which* $X_i(s) = F$ *and* $(X_{i-1}, X_{i+1}) \neq (\underrightarrow{\#}, \underleftarrow{\#})$, *then, with probability at least* $1/2$ *a state of* $\mathcal{G} \cup \mathcal{P}$ *is reached within time* 1.

Proof. If $s \in \mathcal{G} \cup \mathcal{P}$, then the result is trivial. Let s be a state of $\mathcal{F} - (\mathcal{G} \cup \mathcal{P})$ and let i be such that $X_i(s) = F$ and $(X_{i-1}, X_{i+1}) \neq (\underrightarrow{\#}, \underleftarrow{\#})$. Assume without loss of generality that $X_{i+1} \neq \underleftarrow{\#}$, i.e., $X_{i+1} \in \{E_R, R, F, \underrightarrow{\#}\}$. The case for $X_{i-1} \neq \underrightarrow{\#}$ is similar. Furthermore, we can assume that $X_{i+1} \in \{E_R, R, F, \underrightarrow{D}\}$ since if $X_{i+1} \in \{\underrightarrow{W}, \underrightarrow{S}\}$ then s is already in $\mathcal{G}$. We show that the event schema $\text{NEXT}((\texttt{flip}_i, \texttt{left}), (\texttt{flip}_{i+1}, \texttt{right}))$, which by Proposition 4.2 has probability at least 1/2, leads in time at most 1 to a state of $\mathcal{G} \cup \mathcal{P}$. Let f be an adversary of $Unit - Time$, and let α be the execution of M that corresponds to an execution of $H(M, \{s\}, f)$ where if process i flips before process $i+1$ then process i flips left, and if process $i+1$ flips before process i then process $i+1$ flips right.

Within time 1, i takes one step and reaches W. Let $j \in \{i, i+1\}$ be the first of i and $i+1$ that reaches W and let s_1 be the state reached after the first time process j reaches W. If some process reached P in the meantime, then we are done. Otherwise there are two cases to consider. If $j = i$, then, $\texttt{flip}_i$ gives $\texttt{left}$ and $X_i(s_1) = \underleftarrow{W}$ whereas X_{i+1} is (still) in $\{E_R, R, F, \underrightarrow{D}\}$. Therefore, $s_1 \in \mathcal{G}$. If $j = i+1$, then $\texttt{flip}_{i+1}$ gives $\texttt{right}$ and $X_{i+1}(s_1) = \underrightarrow{W}$ whereas $X_i(s_1)$ is (still) F. Therefore, $s_1 \in \mathcal{G}$. ■

Lemma 7.13 *Start with a state* s *of* $\mathcal{F}$. *If there exists a process* i *for which* $X_i(s) = F$ *and* $(X_{i-1}(s), X_{i+1}(s)) = (\underrightarrow{\#}, \underleftarrow{\#})$. *Then, with probability at least* $1/2$, *within time* 2, *a state of* $\mathcal{G} \cup \mathcal{P}$ *is reached.*

Proof. The hypothesis can be summarized into the form $(X_{i-1}(s), X_i(s), X_{i+1}(s)) = (\underrightarrow{\#}, F, \underleftarrow{\#})$. Since $i-1$ and $i+1$ point in different directions, by moving to the right of $i+1$ there is a process k pointing to the left such that process $k+1$ either points to the right or is in $\{E_R, R, F, P\}$, i.e., $X_k(s) \in \{\underleftarrow{W}, \underleftarrow{S}, \underleftarrow{D}\}$ and $X_{k+1}(s) \in \{E_R, R, F, \underrightarrow{W}, \underrightarrow{S}, \underrightarrow{D}, P\}$.

If $X_k(s) \in \{\underleftarrow{W}, \underleftarrow{S}\}$ and $X_{k+1}(s) \neq P$ then $s \in \mathcal{G}$ and we are done; if $X_{k+1}(s) = P$ then $s \in \mathcal{P}$ and we are done. Thus, we can restrict our attention to the case where $X_k(s) = \underleftarrow{D}$.

We show that the event schema $\text{NEXT}((\texttt{flip}_k, \texttt{left}), (\texttt{flip}_{k+1}, \texttt{right}))$, which by Proposition 4.2 has probability at least 1/2, leads in time at most 2 to $\mathcal{G} \cup \mathcal{P}$. Let f be an adversary of $Unit - Time$, and let α be the execution of M that corresponds to an execution of $H(M, \{s\}, f)$ where if process k flips before process $k+1$ then process k flips left, and if process $k+1$ flips before process k then process $k+1$ flips right.

Within time 2, process k takes at least two steps and hence goes to configuration W. Let $j \in \{k, k+1\}$ be the first of k and $k+1$ that reaches W and let s_1 be the state reached after the first time process j reaches W. If some process reached P in the meantime, then we are done. Otherwise there are two cases to consider. If $j = k$, then, $\texttt{flip}_k$ gives $\texttt{left}$ and $X_k(s_1) = \underleftarrow{W}$ whereas X_{k+1} is (still) in $\{E_R, R, F, \underrightarrow{\#}\}$. Therefore, $s_1 \in \mathcal{G}$. If $j = k+1$, then $\texttt{flip}_{k+1}$ gives $\texttt{right}$ and $X_{k+1}(s_1) = \underrightarrow{W}$ whereas $X_k(s_1)$ is (still) in $\{\underleftarrow{D}, F\}$. Therefore, $s_1 \in \mathcal{G}$. ■

Proposition 7.14 *Start with a state* s *of* $\mathcal{F}$. *Then, with probability at least* $1/2$, *within time* 2, *a state of* $\mathcal{G} \cup \mathcal{P}$ *is reached. Equivalently:*

$$\mathcal{F} \xrightarrow[1/2]{2} \mathcal{G} \cup \mathcal{P}.$$

Proof. The hypothesis of Lemmas 7.12 and 7.13 form a partition of $\mathcal{F}$. ■

Finally, we prove $\mathcal{RT} \xrightarrow{3} \mathcal{F} \cup \mathcal{G} \cup \mathcal{P}$.

Proposition 7.15 *Starting from a state s of $\mathcal{RT}$, then, within time 3, a state of $\mathcal{F} \cup \mathcal{G} \cup \mathcal{P}$ is reached. Equivalently:*

$$\mathcal{RT} \xrightarrow{3} \mathcal{F} \cup \mathcal{G} \cup \mathcal{P}.$$

Proof. Let s be a state of $\mathcal{RT}$. If $s \in \mathcal{F} \cup \mathcal{G} \cup \mathcal{P}$, then we are trivially done. Suppose that $s \notin \mathcal{F} \cup \mathcal{G} \cup \mathcal{P}$. Then in s each process is in $\{E_R, R, W, S, D\}$ and there exists at least process in $\{W, S, D\}$. Let f be an adversary of $Unit - Time$, and let α be the execution of M that corresponds to an execution of $H(M, \{s\}, f)$.

We first argue that within time 1 some process reaches a state of $\{S, D, F\}$ in α. This is trivially true if in state s there is some process in $\{S, D\}$. If this is not the case, then all processes are either in E_R or R or W. Within time 1 some process in R or W takes a step. If the first process not in E_R taking a step started in E_R or R, then it reaches F and we are done; if the first process taking a step is in W, then it reaches S since in s no resource is held. Once a process i is in $\{S, D, F\}$, then within two more time units process i reaches either state F or P, and we are done. ■

8 Concluding Remarks

This paper has presented a formal model and a formal proof technique for the estimation of time performance of randomized algorithms that run under the control of general classes of adversaries. The salient aspect of this technique is to prove probabilistic time bounded progress properties and to compose them by means of a powerful composability theorem. The power of the proof method has been illustrated by proving a constant upper bound on the expected time for progress in the Lehmann-Rabin Dining Philosophers algorithm.

We believe that this technique is applicable towards the time analysis of many randomized protocols. It is desirable that the general model and this technique be used for the analysis of other algorithms, so that the power of the method can be tested and/or increased by means of other additional tools. In particular, it is very likely that new event schemas and partial independence results similar to those of Section 4 can be developed.

The specific results about the Lehmann-Rabin Dining Philosophers algorithm can be complemented and extended in many ways. We cite two. First, it would be very satisfying to derive a non trivial lower bound on the time for progress, which should be lower than our upper bound since the upper bound could be easily improved by means of a finer analysis. Second, it would be interesting to consider topologies that are more general than rings.

Acknowledgments. We thank Lenore Zuck for valuable criticism on a draft version of this paper and in particular for pointing out that the state chart notation of [11] can be adapted to our proof method.

References

[1] J. Aspnes and M.P. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 15(1):441–460, September 1990.

[2] M. Ben-Or. Another advantage of free choice: completely asynchronous agreement protocols. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing,* Montreal, Quebec, Canada, August 1983.

[3] K.M. Chandi and J. Misra". *Parallel Program Design: A Foundation.* Addison-Wesley, 1988.

[4] L. Christoff and I. Christoff. Efficient algorithms for verification of equivalences for probabilistic processes. In *Proceedings of the 3rd Workshop on Computer-Aided Verification*, 1991.

[5] R. Gawlick, R. Segala, J.F. Søgaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. Technical Report MIT/LCS/TR-587, Laboratory for Computer Science, MIT, Cambridge, MA, November 1993.

[6] H. Hansson. *Time and Probability in Formal Design of Distributed Systems.* PhD thesis, Department of Computer Science, Uppsala University, 1991.

[7] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, Englewood Cliffs, 1985.

[8] E. Kushilevitz and M. Rabin. Randomized mutual exclusion algorithms revisited. In *Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, pages 275–284, 1992.

[9] 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 8th Annual ACM Symposium on Principles of Programming Languages*, pages 133–138, January 1981.

[10] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations – part II: Timing-based systems. Technical Report MIT/LCS/TM-487, Laboratory for Computer Science, MIT, Cambridge, MA, September 1993.

[11] A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing,* 1(1):53–72, 1986.

[12] J.R. Rao. Reasoning about probabilistic algorithms. In *Proceedings of the 9th Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, August 1990.

[13] I. Saias. Proving probabilistic correctness: the case of Rabin's algorithm for mutual exclusion. In *Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing,* Quebec, Canada, August 1992.

[14] R. Segala and N. Lynch. A model for randomized concurrent systems. Manuscript, 1994.

[15] F.W. Vaandrager and N.A. Lynch. Action transducers and timed automata. In W.R. Cleaveland, editor, *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of *Lecture Notes in Computer Science*, pages 436–455. Springer-Verlag, 1992.

[16] 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.