Formal Verification of Timed Properties of Randomized Distributed Algorithms*

Anna Pogosyants and Roberto Segala
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, MA 02139 USA
email: anya,segala@theory.lcs.mit.edu

Abstract

In [11] a method for the analysis of the expected time complexity of a randomized distributed algorithm is presented. The method consists of proving auxiliary probabilistic time bound statements of the form $U \xrightarrow[p]{t} U'$, which mean that whenever the algorithm begins in a state in set U, it will reach a state in set U' within time t with probability at least p. However, [11] does not provide a formal methodology to prove the validity of a specific probabilistic time bound statement from scratch: each statement is proved by means of ad hoc operational arguments. Unfortunately, operational reasoning is generally error-prone and difficult to check. In this paper we overcome the problem by developing a new technique to prove probabilistic time bound statements, which consists of reducing the analysis of a time bound statement to the analysis of a statement that does not involve probability. As a consequence, several existing techniques for non-randomized algorithms can be applied, and correctness proofs can be verified mechanically.

1 Introduction

Randomization is a useful tool for the solution of problems in distributed computation (e.g., [3, 4, 9]), but even simple randomized algorithms are sometimes hard to verify and analyze. As a result, many randomized algorithms are presented with only informal proofs, and even when formal proofs are presented, they are often ad hoc; mistakes are common. Most of the mistakes come from considering together probabilistic and non-deterministic choices. Another problem with informal and ad hoc arguments is that it is hard to be convinced that all the possible cases are considered.

In this paper we provide a methodology that allows us to verify timed properties of randomized algorithms. Our methodology consists of the following steps.

1. Decompose the timed properties of a randomized algorithm into *probabilistic time bound statements* as suggested in [11]. A probabilistic time bound statement is an expression of the form $U \xrightarrow[p]{t} U'$, where U and U' are sets of states of the algorithm, p is a probability, and t is a positive real number. Its meaning is that whenever an algorithm is started from a state of U, then a state of U' is reached within time t with probability at least p.

2. Reduce each problem of checking whether a probabilistic time bound statement $U \xrightarrow[p]{t} U'$ is satisfied by a randomized algorithm to the problem of checking whether the non-probabilistic time bound statement $U \xrightarrow{t} U'$ is satisfied by some specific non-randomized algorithm. The non-randomized algorithm is derived from the original randomized algorithm.

3. Prove each of the non-probabilistic time bound statements using some of the existing techniques for ordinary, non-randomized, distributed algorithms.

Here we focus on the second and third steps. We provide a way of reducing a probabilistic system, denoted by M, to a non-probabilistic system, denoted by A, so that the validity of $U \xrightarrow{t} U'$ for A implies the validity of $U \xrightarrow[p]{t} U'$ for M. We call the system A a *generating automaton*. Informally, A is obtained from M by forcing some of the probabilistic choices of M to give some specific outcomes, and by changing all the other probabilistic choices of M into nondeterministic choices. The forced outcomes should be chosen so that their probability in M is always greater than or equal to p.

Once the reduction is carried out, since generating automata describe ordinary (non-randomized) algorithms, several existing techniques can be used to prove the new time bound statements. We present a technique based on *progress functions* in the style of [10]. A progress function is a function that associates real numbers with some of the states of a system; roughly speaking, a progress function gives an upper bound on the time left until reaching a state of U'. The existence of a progress function is sufficient for proving a time bound statement. Moreover, proofs based on the progress function method can be machine checked.

The main advantage of our methodology is that it separates completely probability from non-determinism, thus

*Research supported in part by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under contracts N00014-92-J-1795 and N00014-92-J-4033, by the National Science Foundation under grants 9115797-CCR and 8915206-CCR, and by the Office of Naval Research under contract N00014-91-J-1046.

eliminating the most common source of mistakes in correctness proofs. Moreover, our method eliminates ad hoc arguments and allows for proofs to be carried out at different levels of details. A high level description of a proof can be given by specifying the key parameters of our proof method; then, some interesting cases can be considered in more detail, and, if necessary, the most critical parts can be checked mechanically by an automatic theorem prover. Even for high level proofs the advantages of our methodology are evident: it provides a systematic way of listing all possible cases, thus reducing the possibility of not considering some cases. Moreover, a skeptical reader can always fill in the details easily.

We applied the new technique for the formal verification of the randomized Dining Philosophers algorithm of Lehmann and Rabin [9] (same example as in [11]). We used the proof sketch from [11], and we reproved all main lemmas using generating automata and progress functions. We carried out the proofs in full detail and automated the routine parts of the proofs using the Larch Prover [7]. The complete proof (hand and automatic) can be found in [15]. We sketched the proof of correctness of the randomized algorithm for consensus with stopping faults of Ben-Or [4], and, based on the experience that we gained, we are confident that our methodology applies to most of the standard randomized distributed algorithms as well.

Previous work on verification of randomized distributed algorithms appears in [6, 8, 11, 14, 16]. The work of [14] presents a technique, based on ranking functions defined on states, for establishing liveness properties for randomized algorithms; the work of [16] extends UNITY [5] to handle probability, and provides a completeness result for some properties that hold with probability 1; the work of [6, 8] presents model checking techniques; the work of [11] presents a technique based on probabilistic time bound statements, where each statement is proved by means of ad hoc operational arguments. Our methodology, combined with the extensive collection of verification techniques available on ordinary non-randomized algorithms, overcomes several of the limitations of [6, 8, 11, 14, 16].

The rest of the paper is organized as follows. Section 2 presents the non-probabilistic model, introduces the time bound statements, and describes the proof method based on progress functions; Section 3 presents a simplified version of the probabilistic models of [17], introduces the probabilistic time bound statements, and shows how to prove them through generating automata; Section 4 presents the algorithm of Lehmann and Rabin and illustrates our technique by giving some fragments of the correctness proof; Section 5 contains some concluding remarks.

2 The Non-probabilistic Framework

In this section we introduce the non-probabilistic framework. We define semi-timed automata, which are a simplification of the timed automata of [12], and we formalize the statements $U \xrightarrow{t} U'$. Furthermore, we introduce the verification technique based on progress functions.

2.1 Semi-Timed Automata and Time Bound Statements

Definition 2.1 A *semi-timed automaton* A consists of four components

- a set $States(A)$ of states;
- a non-empty set $Start(A) \subseteq States(A)$ of start states;
- a timed action signature
$$Sig(A) = (int(A), ext(A), time(A)),$$
where $int(A), ext(A)$ and $time(A)$ are disjoint sets: $int(A)$ is the set of internal actions, $ext(A)$ is the set of external actions, and $time(A) = \{\nu(t) \mid t \in R^+\}$ is the set of time-passage actions;
- a transition relation
$$Trans(A) \subseteq States(A) \times Actions(A) \times States(A),$$
where $Actions(A)$ denotes the set $int(A) \cup ext(A) \cup time(A)$. The elements of $Trans(A)$ are called transitions. ∎

The main difference between a semi-timed automaton defined here and a timed automaton of [12] is that a semi-timed automaton is not required to satisfy the *trajectory axioms*, which restrict the transition relation so that the actions of $time(A)$ model actual passage of time. The trajectory axioms do not play a fundamental role for the results of this paper, and thus we omit them from the discussion.

An *execution fragment* of a semi-timed automaton A is a sequence α of alternating states and actions of A starting with a state, and, if α is finite, ending with a state, $\alpha = s_0a_1s_1a_2s_2...$, such that for each $i \geq 0$ there exists a transition(s_i, a_{i+1}, s_{i+1}) of A. 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 $ltime(\alpha)$ the sum of the reals in the time passage actions of α. Denote by $frag^*(A)$ the set of finite execution fragments of A. An *execution* is an execution fragment whose first state is a start state. An execution α of A is *admissible* if $ltime(\alpha) = \infty$. Denote by $exec^{\omega}(A)$ and $exec(A)$ the sets of admissible and all executions of A, respectively.

A state s of A is said to *enable* a transition if there is a transition (s, a, s') in $Trans(A)$; an action a is said to be *enabled* from s if there is a transition (s, a, s') in $Trans(A)$; an execution α is said to be complete in A iff either α is infinite or α is finite and $lstate(\alpha)$ does not enable any transition. A state s of A is *reachable* if there exists a finite execution of A that ends in s. Denote by $rstates(A)$ the set of reachable states of A. A finite execution fragment $\alpha_1 = s_0a_1s_1 \cdots a_ns_n$ of A and an execution fragment $\alpha_2 = s_na_{n+1}s_{n+1} \cdots$ of A can be *concatenated*. The concatenation, written $\alpha_1 \frown \alpha_2$, is the execution fragment $s_0a_1s_1 \cdots a_ns_na_{n+1}s_{n+1} \cdots$. An execution fragment α_1 of A is a *prefix* of an execution fragment α_2 of A, written $\alpha_1 \leq \alpha_2$, iff either $\alpha_1 = \alpha_2$ or α_1 is finite and there exists an execution fragment α_1' of A such that $\alpha_2 = \alpha_1 \frown \alpha_1'$. If $\alpha = \alpha_1 \frown \alpha_2$, then α_2 is called a *suffix* of α, and it is denoted alternatively by $\alpha \rhd \alpha_1$.

In this paper we concentrate on the admissible executions of a semi-timed automaton, since they are those executions that represent realistic behaviors: no system can stop time, and thus time grows unboundedly. We can now define formally a time bound statement.

Definition 2.2 Let A be a semi-timed automaton, $U, U' \subseteq States(A)$, $t \in R^{\geq 0}$. Then $U \xrightarrow{t} U'$ denotes a predicate that is true for A iff every admissible execution fragment α of A that starts from a state of U has a finite prefix α' such that $ltime(\alpha') \leq t$ and $lstate(\alpha') \in U'$. ∎

2.2 Proving Time Bound Statements

There are several techniques to prove properties of semi-timed automata. Here we present a technique, based on progress functions, which is useful to prove the validity of time bound statements. The key idea of the technique is based on the following theorem.

Theorem 2.3 *Let A be a semi-timed automaton, $U, U' \subseteq States(A)$. Suppose that there exist two functions*

$$p : States(A) \rightarrow Bool, \text{ and}$$
$$\mathcal{P} : States(A) \rightarrow R^{\geq 0},$$

such that the following hold:

1. *If $s \in U$ then either $p(s) \wedge \mathcal{P}(s) \leq t$ or $s \in U'$;*
2. *For all transitions$(s, a, s') \in Trans(A)$ such that $p(s)$*
   (a) *if a is a time-passage action $\nu(\Delta t)$ then $p(s') \wedge (\mathcal{P}(s) - \Delta t \geq \mathcal{P}(s'))$;*
   (b) *if a is not a time-passage action then either $p(s') \wedge (\mathcal{P}(s) \geq \mathcal{P}(s'))$ or $s' \in U'$.*

Then $U \xrightarrow{t} U'$ is true for A. ∎

A pair $(p, \mathcal{P})$ satisfying the conditions of Theorem 2.3 defines a partial function from states of A onto nonnegative real numbers. The domain of the function consists of states of A such that $p(s)$ is true, and the values of the function are given by $\mathcal{P}(s)$. Informally, this partial function describes an upper bound on the time left until reaching a state of U'. Condition 1 guarantees that the upper bound for the states of U is at most t; Conditions $2(a, b)$ ensure that the upper bound associated with each state is correct. We refer to the pair $(p, \mathcal{P})$ as a *progress function*. The idea of a progress function is borrowed from [10].

3 The Probabilistic Framework

In this section we present our methodology for proving probabilistic time bound statements of the form $U \xrightarrow[p]{t} U'$. We start by introducing the probabilistic semi-timed automaton model, which is a simplification of the model of [17], and by formalizing the statements $U \xrightarrow[p]{t} U'$ within this model. Then we show how such statements can be verified formally by reducing the problem to a new problem that does not involve probability.

3.1 Probabilistic Semi-Timed Automata and Time Bound Statements

Definition 3.1 A *probability space* is a triplet $(\Omega, \mathcal{F}, P)$ where Ω is a set, also called the *sample space*, $\mathcal{F}$ is a collection of subsets of Ω that is closed under complement and countable union and such that $\Omega \in \mathcal{F}$, also called a *σ-field*, and P is a function from $\mathcal{F}$ to $[0,1]$ such that $P[\Omega] = 1$ and such that 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* 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. For any arbitrary set X, let $Probs(X)$ denote the set of discrete probability distributions whose sample space is a subset of X.

A probability space is called a *Dirac distribution* if its sample space contains exactly one element. Denote by $\mathcal{D}x$ the Dirac space whose sample set is $\{x\}$. ∎

Definition 3.2 A *probabilistic semi-timed automaton* M consists of four components:

- a set $States(M)$ of states;
- a non-empty set $Start(M) \subseteq States(M)$ of start states;
- a timed action signature $Sig(M)$;
- a transition relation

  $$Trans(M) \subseteq States(M) \times Actions(M) \times Probs(States(M)),$$

  such that each transition with a time-passage action leads to a Dirac distribution, i.e., if $(s, \nu(t), (\Omega, \mathcal{F}, P)) \in Trans(M)$, then $|\Omega| = 1$.

A probabilistic semi-timed automaton M is *fully probabilistic* if M has exactly one start state and each state of M enables at most one transition. ∎

Execution fragments and executions are defined similarly to the non-probabilistic case. An *execution fragment* of M is a sequence α of alternating states and actions of M starting with a state, and, if α is finite ending with a state, $\alpha = s_0 a_1 s_1 a_2 s_2 ...$, such that for each $i \geq 0$ there exists a transition$(s_i, a_{i+1}, (\Omega, \mathcal{F}, P))$ of M such that $s_{i+1} \in \Omega$. All the terminology that is used for executions in the non-probabilistic case applies to the probabilistic case as well.

An execution fragment of M is the result of resolving both the probabilistic and the nondeterministic choices of M. If only the nondeterministic choices are resolved, then we obtain a fully probabilistic semi-timed automaton, which we call a *probabilistic execution fragment* of M. Probabilistic execution fragments are the objects where the probabilistic behavior of a probabilistic automaton can be studied. Informally, a probabilistic execution can be thought of as the result of unfolding the transition relation of a probabilistic automaton and then choosing at most one transition for each state of the unfolding.

Definition 3.3 A *probabilistic execution fragment* H of a probabilistic semi-timed automaton M is a fully probabilistic semi-timed automaton, such that:

- $States(H) \subseteq frag^*(M)$; let q range over the states of H;
- $Sig(H) = Sig(M)$;
- $Start(H) \subseteq States(M)$;
- for each transition$(q, a, (\Omega, \mathcal{F}, P))$ of H there exists a transition$(lstate(q), a, (\Omega', \mathcal{F}', P'))$ of M such that $\Omega = \{qas \mid s \in \Omega'\}$ and $P'[qas] = P[s]$ for all $s \in \Omega'$;
- all the states of H are reachable, i.e., for each state q of H there is an execution of H that leads to q.

We say that a probabilistic execution fragment H starts with a state s of M if $Start(H) = \{s\}$. A *probabilistic execution* H of a probabilistic semi-timed automaton M is a probabilistic execution fragment of M that starts with a state of $Start(M)$.

A probabilistic execution fragment H of M is said to be *admissible* if every complete execution of H is admissible. ■

Remark 3.1 The definition of a probabilistic execution that we have given here is simpler than the definition given in [17]. The main differences are the following:

1. the start state of a probabilistic execution fragment of [17] can be an arbitrary finite execution fragment rather than just a state;

2. each transition that leaves from a state of a probabilistic execution fragment of [17] can lead to a probability distribution over pairs (action, state) plus a special symbol δ that models deadlock.

The first feature is necessary to prove what we are going to state in Proposition 3.5; the second feature is necessary if we want to be able to resolve the nondeterminism in a probabilistic automaton using randomization. In this paper we assume that randomization cannot be used to resolve the nondeterminism; however, all the results that we prove here carry over to the randomized case. ■

There is a strong correspondence between the executions of a probabilistic semi-timed automaton and the executions of one of its probabilistic executions. Specifically, if H is a probabilistic execution fragment of M, then every execution of H can be represented by an execution fragment of M. Below we define two operations on execution fragments that exploit the correspondence. Let q_0 be the start state of H, and let $\alpha = q_0 a_1 q_1 a_2 q_2 \cdots$ be an execution of H. Denote by $\alpha{\downarrow}$ the execution $q_0 \frown a_1 \mathit{lstate}(q_1) a_2 \mathit{lstate}(q_2)\ldots$. Note that, from the definition of a probabilistic execution fragment, $\alpha{\downarrow}$ is an execution fragment of M. For each execution fragment α' of M starting with q_0 ($\alpha' = q_0 \frown a_1 s_1 a_2 s_2 \ldots$), let $\alpha'{\uparrow}H$ denote the sequence $q_0 a_1 (q_0 a_1 s_1) a_2 (q_0 a_1 s_1 a_2 s_2)$. Then, it is easy to show that for each execution fragment α of H, $(\alpha{\downarrow}){\uparrow}H = \alpha$.

We now define the probability space associated with a probabilistic execution fragment, so that its probabilistic behavior can be studied. Given a probabilistic execution fragment H, the sample space is the set $\Omega_H = \{\alpha{\downarrow} \mid \alpha$ is a complete execution of $H\}$. The σ-field $\mathcal{F}_H$ is the smallest σ-field that contains the set of *cones* C_α, consisting of the executions of Ω_H having α as a prefix[1]. The probability measure P_H is the unique extension of the probability measure defined on cones as follows: $P_H[C_\alpha]$ is the product of the probabilities of each transition of H generating α. In [17] 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.

An *event* E of H is a *measurable* subset of Ω_H, i.e., an elament of $\mathcal{F}_H$. However, events alone are not sufficient for the analysis a probabilistic semi-timed automaton, since a probabilistic semi-timed automaton may have several probabilistic execution fragments, and each probabilistic execution fragment is associated with a different probability space. Thus, a more abstract structure is needed. An *event schema* is a function that associates an event of $\mathcal{F}_H$ with each probabilistic execution fragment of M.

We can now give the formal definition of a probabilistic time bound statement.

Definition 3.4 Let M be a probabilistic semi-timed automaton, $U, U' \subseteq States(M)$, $t \in R^{\geq 0}$. Then $U \xrightarrow[p]{t} U'$ denotes a predicate that is true for M iff for each admissible probabilistic execution fragment H of M that starts with a state of U, $P_H[e_{U',t}] \geq p$, where $e_{U',t}(H)$ is the set of executions α of Ω_H such that α has a prefix α' such that $ltime(\alpha') \leq t$ and $lstate(\alpha') \in U'$. ■

Some inference rules of [11] that can be used to derive new probabilistic time bound statements from existing ones are the following.

Proposition 3.5 *Let M be a probabilistic semi-timed automaton, $U, U', U'', U''' \subseteq States(M)$. Then,*

1. *if $U \xrightarrow[p]{t} U'$ and $U' \xrightarrow[p']{t'} U''$, then $U \xrightarrow[pp']{t+t'} U''$;*

2. *if $U \xrightarrow[p]{t} U'$, then $U \cup U'' \xrightarrow[p]{t} U' \cup U''$;*

3. *if $U \xrightarrow[p]{t} U'$ and $U'' \xrightarrow[p']{t'} U'''$, then $U \cup U'' \xrightarrow[min(p,p')]{max(t,t')} U' \cup U'''$.* ■

3.2 Proving Probabilistic Time Bound Statements

A technique to prove the validity of a probabilistic time bound statement $U \xrightarrow[p]{t} U'$ for a probabilistic semi-timed automaton M, which is hinted at in [11], consists of the following steps.

1. Choose a set of random draws that may occur in a probabilistic execution of M, and choose some of the possible outcomes;

2. Show that starting from a state of U, no matter how the nondeterminism is resolved, the chosen random draws give the chosen outcomes with some minimum probability p;

3. Show that whenever the chosen draws give the chosen outcome, a state of U' is reached within time t.

This technique corresponds to the informal arguments of correctness that appear in the literature. Usually the intuition behind an algorithm is exactly that success is guaranteed whenever some specific random draws give some specific results.

The first two steps can be reformulated in terms of the set Θ of admissible execution fragments of M that start in a state of U and such that each of the chosen random draws gives one of the chosen outcomes. Then the minimum probability requirement of the second step can be formulated as follows: for every admissible probabilistic execution fragment H of M that starts in a state of U, $P[\Theta \cap \Omega_H] \geq p$.

Once the first two steps are reformulated in terms of Θ, the third step can be carryed out by showing that for any execution fragment α of Θ a state of U' is reached within time t. If we build a non-probabilistic automaton, called

[1] Note that a cone C_α can be used to express the fact that the finite execution α occurs.

a *generating automaton*, so that its admissible executions contain Θ, and we show that each admissible execution of the generating automaton leads to U' within time t, then we are done. To carry out this last step the progress function of Section 2.2 or any other technique for non-probabilistic automata can be applied. Given a set Θ, a generating automaton for Θ with the properties described above can be built as follows.

Definition 3.6 Let M be a probabilistic semi-timed automaton, and let Θ be a set of admissible execution fragments of M. Define a semi-timed automaton A such that

1. $Sig(A) = Sig(M)$;
2. $States(A) = \{\alpha \in frag^*(M) \mid \exists_{\alpha' \in \Theta} \alpha \leq \alpha'\}$, i.e., $States(A)$ is the set of finite prefixes of elements of Θ. Denote a generic state of A by q;
3. $Start(A) = \{q \in States(A) \mid |q| = 0\}$;
4. $(q, a, q') \in Trans(A)$ iff $q' = qas$, where $s = lstate(q')$.

We call A the *generating automaton* for Θ, and we denote it by $G_M(\Theta)$. ∎

Lemma 3.7 *Let M be a probabilistic semi-timed automaton, and let Θ be a set of admissible execution fragments of M. Then $\Theta \subseteq exec^\omega(G_M(\Theta))$.* ∎

We can now formulate the main theorem that justifies our reduction.

Theorem 3.8 *Let M be a probabilistic semi-timed automaton, and let U and U' be sets of states of M. Let Θ be a set of admissible execution fragments of M such that the following hold.*

1. *For every admissible probabilistic execution fragment H of M starting with a state of U, $P_H[\Theta \cap \Omega_H] \geq p$.*
2. *$Start(G_M(\Theta)) \xrightarrow{t} U'_\uparrow$, where $U'_\uparrow$ is the set of states q of $G_M(\Theta)$ such that $lstate(q) \in U'$.*

Then $U \xrightarrow[p]{t} U'$ is valid for M. ∎

Theorem 3.8 states that by chosing a set Θ that satisfies Condition 1 we can reduce the proof of $U \xrightarrow[p]{t} U'$ for M (the probabilistic problem) to the proof of $Start(G_M(\Theta)) \xrightarrow{t} U'_\uparrow$ for $G_M(\Theta)$ (a non-probabilistic problem). We can use any standard technique for non-probabilistic automata to solve the non-probabilistic problem. In particular, we can use the technique based on progress functions of Section 2, which is also machine checkable.

Remark 3.2 Although the problem is reduced to the analysis of a non-probabilistic system, it is important to understand that building a generating automaton is not trivial in general. To build a generating automaton it is necessary to identify the right set Θ and make sure that Θ is expressible by an automaton. This process involves understanding the main idea of a randomized algorithm, which is usually the creative part of a correctness proof. Fortunately, the designer of an algorithm usually has enough intuition about why the algorithm works so that the task of building a generating automaton is simplified. ∎

4 Example

In this section we use the randomized Dining Philosophers algorithm of Lehmann and Rabin to illustrate our method. The same problem was analized in [11]. Here we concentrate more on a fragment of the proof of [11] and we show how it is possible to use generating automata and progress functions to prove the validity of a time bound statement. We start by presenting the algorithm of Lehmann and Rabin; then we consider a specific time bound statement and we define the corresponding generating automaton and the corresponding progress function; finally, we prove by structural induction that indeed we have defined a progress function. The whole analysis is carried out in a simplified model, called the *probabilistic MMT automaton model*, where time-passage is constrained according to some specific restricted kinds of rules. Since the details of the MMT model are necessary only to prove that we have defined a progress function, we postpone the presentation of the simplified model to the point where it is necessary.

4.1 The Problem, the Protocol, and its Formalization

The problem consists of allocating n resources among n processes arranged in a ring. The resources are interspersed between the processes, and each process requires both its adjacent resources to perform its task.

According to the Lehmann-Rabin algorithm, every process that is trying to obtain its resources executes the following protocol. It first flips a fair coin to choose between its left and right resources. Then it waits for the chosen resource to be free and takes it. After that, it attempts to access the other resource: if the other resource is free, then the process gets it an proceeds to its task; otherwise, the process returns the first resource and starts the protocol again. The algorithm ensures exclusive possession of resources and also ensures that, with probability 1, if some process is trying to get its resources, then eventually some process (may be different one) will proceed to its task.

To formalize the algorithm, we use the probabilistic MMT automaton model, which is a useful special case of probabilistic semi-timed automata. A probabilistic MMT automaton is an automaton whose actions are partitioned into classes. Each class T is associated with an upper bound and a lower bound, which give restrictions to the time at which any action from T can occur (cf. Sections 4.5 and 4.6).

Figure 1 describes the probabilistic MMT automaton for the Lehmann-Rabin algorithm. Each state of the automaton is a tuple $(pc_1, \ldots, pc_n, u_1, \ldots, u_n, Res_1, \ldots, Res_n)$, where pc_i is a program counter of process i, u_i are auxiliary variables, and Res_i represents the state of resource i. The table below explains the mnemonic meaning of the values of the program counter and gives the names of the actions that each process enables from each of its states.

Nr.	pc	Action	Informal meaning
0	R	try_i	**R**eminder region
1	F	$flip_i$	Ready to **Flip**
2	W	$wait_i$	**W**aiting for first resource
3	S	$second_i$	Checking for **S**econd resource
4	D	$drop_i$	**D**ropping first resource
5	P	$crit_i$	**P**re-critical region
6	C	$exit_i$	**C**ritical region
7	E_F	$dropf_i$	**E**xit: drop **F**irst resource
8	E_S	$drops_i$	**E**xit: drop **S**econd resource
9	E_R	rem_i	**E**xit: move to **R**eminder region

State

$pc_i \in \{R, F, W, S, D, P, C, E_F, E_S, E_R\}$, initially R
$Res_i \in \{free, taken\}$, initially $free$; $u_i \in \{left, right\}$

Actions

output try_i
Pre: $pc_i = R$
Eff: $pc_i \leftarrow F$

internal $flip_i$
Pre: $pc_i = F$
Eff: $Pr(u_i \leftarrow left) = 1/2 \wedge$
$Pr(u_i \leftarrow right) = 1/2$;
$pc_i \leftarrow W$

internal $wait_i$
Pre: $pc_i = W$
Eff: if $u_i = left \wedge Res_i = free$
then $Res_i \leftarrow taken$; $pc_i \leftarrow S$
elseif $u_i = right \wedge Res_{i-1} = free$
then $Res_{i-1} \leftarrow taken$; $pc_i \leftarrow S$
else $pc_i \leftarrow W$

internal $second_i$
Pre: $pc_i = S$
Eff: if $u_i = right \wedge Res_i = free$
then $Res_i \leftarrow taken$; $pc_i \leftarrow P$
elseif $u_i = left \wedge Res_{i-1} = free$
then $Res_{i-1} \leftarrow taken$; $pc_i \leftarrow P$
else $pc_i \leftarrow D$

internal $drop_i$
Pre: $pc_i = D$
Eff: if $u_i = left$ then $Res_i \leftarrow free$
elseif $u_i = right$ then $Res_{i-1} = free$;
$pc_i \leftarrow F$

output $crit_i$
Pre: $pc_i = P$
Eff: $pc_i \leftarrow C$

output $exit_i$
Pre: $pc_i = C$
Eff: $pc_i \leftarrow E_F$

internal $dropf_i$
Pre: $pc_i = E_F$
Eff: $[u_i \leftarrow right; Res_i \leftarrow free] \vee$
$[u_i \leftarrow left; Res_{i-1} \leftarrow free]$
$pc_i \leftarrow E_S$

internal $drops_i$
Pre: $pc_i = E_S$
Eff: if $u_i = right$ then $Res_{i-1} \leftarrow free$
elseif $u_i = left$ then $Res_i \leftarrow free$; $pc_i \leftarrow E_R$

output rem_i
Pre: $pc_i = E_R$
Eff: $pc_i \leftarrow R$

Classes:
$T_i = \{flip_i, wait_i, second_i, drop_i, crit_i, dropf_i, drops_i, rem_i\}$: $[0, l]$
try_i: $[0, \infty]$
$exit_i$: $[0, \infty]$

Figure 1: The Lehmann-Rabin Algorithm

We assume that processes are numbered clockwise from 1 to n. We also assume that resource Res_i is between processes i and $i+1$. Labels are identified modulo n so that, e.g., process $n+1$ coincides with process 1.

All actions of a process i except for try_i and $exit_i$ belong to the same class T_i, which has the trivial lower bound 0 and the upper bound l. This means that if an action of process i different from try_i and $exit_i$ is enabled, then process i must perform a transition within time l. Classes try_i and $exit_i$ have an upper bound ∞, which means that there is no limitation to the amount of time that can elapse before actions try_i and $exit_i$ occur after they are enabled.

Let LR be the probabilistic semi-timed automaton that corresponds to the MMT automaton above. The states of LR include, along with components specified in Figure 1, some other variables. For the purpose of our example, the relevant variables are the variable now, which specifies the current absolute time, and the variable $last(T)$ for every class T, which specifies an upper bound on the absolute time at which an action from T can occur (see Section 4.6 for more detail). For a notational convenience, if $a \in T$ we sometimes write $s.last(a)$ instead of $s.last(T)$, where $s.last(T)$ denotes the value of $last(T)$ in s.

We denote by X_i the pair (pc_i, u_i), which constitutes the local state of process i. The value of each pair can be represented by the value of pc_i and an arrow (to the left or to the right) that describes the value of u_i. We omit the arrow if we want to denote both the possible pairs. E.g., $X_i = \overrightarrow{W}$ means that $X_i \in \{(W, right)\}$ and $X_i = W$ means that $X_i \in \{(W, left), (W, right)\}$. We say that a process i is in its *trying* region if $X_i \in \{F, W, S, D, P\}$ and we say that a process i is in its *critical* region if $X_i = C$.

4.2 A Fragment of the Correctness Proof

Let $\mathcal{T}$ be the set of all reachable states of LR in which some process is in its trying region, $\mathcal{C}$ be the set of all reachable states of LR in which some process is in its critical region. In [11] it is proved that if a process is in its trying region then within time $13l$ and with probability at least $1/8$ some process gets to its critical region. In other words,

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

is true for LR. The high level proof in [11] breaks the problem into several intermediate probabilistic time bound statements that are later combined using inference rules. Each intermediate statement is proved operationally. Here we present a fragment of the formal proof of one of the intermediate statements. The interested reader is referred to [15] for the formal proofs of all the other statements used in [11]. Consider the following sets of states.

$\mathcal{RT}$: the set of states of $\mathcal{T}$ in which all processes are either in R or in its trying region;

$\mathcal{F}$: the set of states of $\mathcal{RT}$ in which some process is in F;

$\mathcal{L}$: the set of reachable states of LR in which some process is in P;

$\mathcal{G}$: the set of states of $\mathcal{RT}$ in which there is a process i such that

$$(X_i \in \{\overleftarrow{W} \cup \overleftarrow{S}\}) \wedge (X_{i-1} \in \{\overrightarrow{W} \cup \overrightarrow{S} \cup \overrightarrow{D} \cup R \cup F\}),$$
or
$$(X_i \in \{\overrightarrow{W} \cup \overrightarrow{S}\}) \wedge (X_{i+1} = \{\overleftarrow{W} \cup \overleftarrow{S} \cup \overleftarrow{D} \cup R \cup F\}).$$

The statement we want to analyze here is

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

The meaning of this statement is that if some process is ready to flip a coin, then within time $2l$ and with probability at least $1/2$ either some process is in P, which means that this same process will go to C on the next transition, or $\mathcal{G}$ is reached. Reaching $\mathcal{G}$ is a substantial progress towards reaching $\mathcal{C}$, since in each state of $\mathcal{G}$ there must exist two adjacent processes that act asymmetrically. Asymmetric behavior is a key element for reaching $\mathcal{C}$.

Just for completeness of presentation, the other statements that are shown in [11] are the following.

$$\mathcal{T} \xrightarrow{2} \mathcal{RT} \cup \mathcal{C},$$
$$\mathcal{RT} \xrightarrow{3} \mathcal{F} \cup \mathcal{G} \cup \mathcal{L},$$
$$\mathcal{G} \xrightarrow[1/4]{5} \mathcal{L},$$
$$\mathcal{L} \xrightarrow{1} \mathcal{C}.$$

In the rest of the paper we analyze only $\mathcal{F} \xrightarrow[1/2]{2l} \mathcal{G} \cup \mathcal{L}$.

4.3 Help in Building a Generating Automaton

Before proving the time bound statement above, we state an auxiliary result, that can be used to discharge the first condition of Theorem 3.8. Let M be a probabilistic semi-timed automaton, $\mathcal{S}$ be a collection of pairs (a_i, U_i), $1 \leq i \leq k$, consisting of an action a_i of M and a set of states U_i of M. Assume that the actions a_i's are all distinct. Let us restrict our attention to probabilistic execution fragments that start from a state of U, and let $\mathit{first}_{\mathcal{S}}(M, U)$ be the set of admissible execution fragments α of M such that the first state of α is in U and such that either none of the a_i's occur in α, or, if a_j is the first of the a_i's that occur in α, the first occurrence of a_j in α leads to a state of U_j. The following Lemma follows from results of [11].

Lemma 4.1 *Let H be an admissible probabilistic execution fragment of M staring with a state of U, $p_1, ..., p_n$ be real numbers between 0 and 1 such that for each $1 \leq j \leq n$ and for each transition$(s, a_j, (\Omega, \mathcal{F}, P))$ the probability $P[U_j \cap \Omega] \geq p_j$. Then $P_H[\mathit{first}_{\mathcal{S}}(M, U) \cap \Omega_H] \geq \mathit{min}(p_1, ..., p_n)$.* ■

Lemma 4.1 is called a *coin lemma* in [11] since in most of the algorithms randomization is introduced by flipping fair coins. Each action a_i identifies one of the possible random draws and the set U_i identifies the outcomes that are allowed. If each a_i identifies the process of flipping a coin, then Lemma 4.1 can be used to say that no matter how the nondeterminism is resolved the probability that either no coin is flipped or the first coin that is flipped gives head is at least $1/2$.

4.4 Building the Generating Automaton and the Progress Function

We now return to the proof of $\mathcal{F} \xrightarrow[1/2]{2l} \mathcal{G} \cup \mathcal{L}$ for LR. The main intuition is that if some process flips a coin, then one of the two outcomes leads to a state where the symmetry is broken. Since in $\mathcal{F}$ at least one process is ready to flip a coin, within time l some process flips. Once a process has flipped and the coin flipped has given the desired result, it may be that another transition is necessary before reaching $\mathcal{G}$. For this reason the time bound associated with the time bound statement that we are proving is $2l$.

Consider a state s of $\mathcal{F}$ and consider a number k such that $X_k = F$. We know that k exists from definition of $\mathcal{F}$. Let $\mathcal{F}_1$ be the set of states of $\mathcal{F}$ such that $X_{k-1} \in \{\overrightarrow{S} \cup \overrightarrow{W} \cup \overrightarrow{D} \cup R \cup F\}$, $\mathcal{F}_2$ be the set of states of $\mathcal{F}$ such that $X_{k+1} \in \{\overleftarrow{S} \cup \overleftarrow{W} \cup \overleftarrow{D} \cup R \cup F\}$, and $\mathcal{F}_3$ be the set of states of $\mathcal{F}$ such that $X_{k-1} \in \{\overrightarrow{S} \cup \overrightarrow{W} \cup \overrightarrow{D}\}$ and $X_{k+1} \in \{\overleftarrow{S} \cup \overleftarrow{W} \cup \overleftarrow{D}\}$. The sets $\mathcal{F}_1$, $\mathcal{F}_2$ and $\mathcal{F}_3$ form a partition of $\mathcal{F}$. We prove a probabilistic time bound statement for each of the three sets, and later combine the statements using Proposition 3.5.

Lemma 4.2 $\mathcal{F}_1 \xrightarrow[1/2]{l} \mathcal{G} \cup \mathcal{L}$.

Proof. Denote by left_i and right_i the sets of states of LR where u_i is equal to *left* and *right*, respectively. Let $\mathcal{S} = \{(\mathit{flip}_k, \mathit{left}_k), (\mathit{flip}_{k-1}, \mathit{right}_{k-1})\}$. We prove Lemma 4.2 by showing that the set $\Theta_1 = \mathit{first}_{\mathcal{S}}(LR, \mathcal{F}_1)$ satisfies the conditions of Theorem 3.8 with $t = l$ and $p = 1/2$. From Lemma 4.1, $P_H[\Theta_1 \cap \Omega_H] \geq 1/2$ for all admissible probabilistic execution fragments H that start with a state of $\mathcal{F}_1$.

Let A_1 be the generating automaton for Θ_1. It is left to show that $\mathit{Start}(A_1) \xrightarrow{l} \mathcal{G}_\uparrow \cup \mathcal{L}_\uparrow$ is true for A_1. We do it using the progress function method.

Let s be a state of A_1. Let $\mathit{flipped}_{k,k-1}(s)$ be a predicate that is true iff either flip_k or flip_{k-1} occur in s. For shorthand we write $s.X_i$, $s.\mathit{last}(T_i)$ and $s.\mathit{now}$ instead of $\mathit{lstate}(s).X_i$, $\mathit{lstate}(s).\mathit{last}(T_i)$ and $\mathit{lstate}(s).\mathit{now}$ respectively.

Define the predicate p on $\mathit{States}(A_1)$, and the function $\mathcal{P} : \mathit{States}(A_1) \rightarrow R^{\geq 0}$ as follows:

$$\begin{array}{rcl} p(s) & \Leftrightarrow & s \in \mathit{rstates}(A_1) \wedge \neg \mathit{flipped}_{k,k-1}(s) \\ & & \wedge s.X_k = F \wedge (s.X_{k-1} \in \{\overrightarrow{D} \cup F \cup R\}) \\ & & \wedge s \in \mathcal{RT}_\uparrow \setminus \mathcal{L}_\uparrow \\ \mathcal{P}(s) & = & (\mathit{min}(s.\mathit{last}(\mathit{flip}_k), s.\mathit{last}(\mathit{flip}_{k-1})) - s.\mathit{now}). \end{array}$$

The complete proof that $(p(s), \mathcal{P}(s))$ is a progress function can be carried out by showing that the conditions of Theorem 2.3 are satisfied. This can be done by standard structural induction, which considers the effect of all actions, one at a time (cf. Section 4.7). The proof can be verified mechanically by a computer. The complete proof together with its automation can be found in [15]. ■

Informally, $p(s)$ says that s is a reachable state of A_1 such that $s \notin \mathcal{G}_\uparrow \cup \mathcal{L}_\uparrow$ and the processes k and $k-1$ did not flip any coin yet. That is,

$$p(s) \Leftrightarrow s \in \mathit{rstates}(A_1) \wedge \neg \mathit{flipped}_{k,k-1}(s) \wedge s \notin \mathcal{G}_\uparrow \cup \mathcal{L}_\uparrow.$$

However, to carry out the inductive proof that $(p(s), \mathcal{P}(s))$ is a progress function, we need to specify in more detail what it means for s not to be in $\mathcal{G}_\uparrow \cup \mathcal{L}_\uparrow$, and so we obtain the additional clause $s.X_k = F \wedge (s.X_{k-1} \in \{\overrightarrow{D} \cup F \cup R\})$ that appears in the proof of Lemma 4.2. This clause is present also in the operational argument of [11], but it is not stated explicitly. Using our method we can check that this clause is correct, since otherwise the verification that $(p, \mathcal{P})$ is a progress function would break.

The idea behind the definition of $\mathcal{P}(s)$ is that a state of $\mathcal{G}_\uparrow$ is reached for sure as soon as the first flip action of either process k or $k-1$ occurs. Thus, $\mathcal{P}(s)$ gives an upper bound on the time left until either process k or $k-1$ flips. By carrying out the complete progress function proof we can check that the informal reasoning is correct.

Lemma 4.3 $\mathcal{F}_2 \xrightarrow[1/2]{l} \mathcal{G} \cup \mathcal{L}$.

Proof. Symmetric to the proof of Lemma 4.2. ■

Lemma 4.4 $\mathcal{F}_3 \xrightarrow[1/2]{2l} \mathcal{G} \cup \mathcal{L}$.

Proof. If $s \in \mathcal{F}_3$ then from the fact that processes are arranged in a ring we get that there exists a process j such that

$$s.X_j \in \{\overrightarrow{W} \cup \overrightarrow{S} \cup \overrightarrow{D} \cup R \cup F\} \wedge s.X_{j+1}\{\overleftarrow{W} \cup \overleftarrow{S} \cup \overleftarrow{D}\}.$$

Let $\mathcal{S} = \{(\mathit{flip}_j, \mathit{right}_j), (\mathit{flip}_{j+1}, \mathit{left}_{j+1})\}$. We prove Lemma 4.4 by showing that the set $\Theta_3 = \mathit{first}_{\mathcal{S}}(LR, \mathcal{F}_3)$ satisfies the conditions of Theorem 3.8 with $t = 2l$ and $p = 1/2$. We know that $P_H(\Theta_3 \cap H) \geq 1/2$ for all admissible H that start with a state of $\mathcal{F}_3$ (Lemma 4.1).

Let A_3 be the generating automaton for Θ_3. It is left to show $\mathit{Start}(A_3) \xrightarrow{2l} \mathcal{G}_\uparrow \cup \mathcal{L}_\uparrow$ for A_3. We do it using the progress function method.

Let s be a state of A_3. Let $\mathit{flipped}_{j,j+1}(s)$ be a predicate that is true iff either flip_j or flip_{j+1} occur in s.

Define the predicate p on $\mathit{States}(A_3)$, and the function $\mathcal{P} : \mathit{States}(A_3) \to R^{\geq 0}$ as follows:

$$\begin{aligned} p(s) \Leftrightarrow\ & s \in \mathit{rstates}(A_3) \wedge \neg \mathit{flipped}_{j,j+1}(s) \\ & \wedge s.X_j \in \{\overrightarrow{W} \cup \overrightarrow{S} \cup \overrightarrow{D} \cup R \cup F\} \\ & \wedge s.X_{j+1} \in \{\overleftarrow{D} \cup F\} \wedge s \in \mathcal{RT}_\uparrow \setminus \mathcal{L}_\uparrow; \end{aligned}$$

if $X_{j+1} = \overleftarrow{D}$, then

$$\mathcal{P}(s) = (s.\mathit{last}(\mathit{drop}_{j+1}) + l) - s.\mathit{now};$$

otherwise,

$$\mathcal{P}(s) = \mathit{min}(s.\mathit{last}(\mathit{flip}_j), s.\mathit{last}(\mathit{flip}_{j+1})) - s.\mathit{now}.$$

The complete proof that $(p(s), \mathcal{P}(s))$ is a progress function can be carried out by induction as in the proof of Lemma 4.2. The complete proof together with its automation can be found in [15]. ■

Informally, $p(s)$ says that s is a reachable state of A_3 such that $s \notin \mathcal{G}_\uparrow \cup \mathcal{L}_\uparrow$ and the processes j and $j+1$ did not flip any coin yet. That is,

$$p(s) \Leftrightarrow s \in \mathit{rstates}(A_3) \wedge \neg \mathit{flipped}_{j,j+1}(s) \wedge s \notin \mathcal{G}_\uparrow \cup \mathcal{L}_\uparrow.$$

However, to carry out the inductive proof that $(p(s), \mathcal{P}(s))$ is a progress function, we need to specify in more detail what it means for s not to be in $\mathcal{G}_\uparrow \cup \mathcal{L}_\uparrow$, and so we obtain the additional clause $s.X_j \in \{\overrightarrow{W} \cup \overrightarrow{S} \cup \overrightarrow{D} \cup R \cup F\} \wedge s.X_{j+1} \in \{\overleftarrow{D} \cup F\}$ that appears in the proof of Lemma 4.4. The clause is present in the operational argument of [11], but it is not stated explicitly. Using our method we can verify that this clause is correct, since otherwise the verification that $(p, \mathcal{P})$ is a progress function would break.

The idea behind the definition of $\mathcal{P}(s)$ is that if $s.X_{j+1} = F$ then $\mathcal{G}_\uparrow$ will be reached for sure after the first *flip* of processes j or $j+1$. This is expressed by the second clause for $\mathcal{P}(s)$. If $s.X_{j+1} = \overleftarrow{D}$ then we have to wait for process $j+1$ to perform *drop* (at most $\mathit{last}(\mathit{drop}_{j+1}) - \mathit{now}$), and then for one of the processes of j and $j+1$ to perform a *flip* (at most l). This is expressed by the first clause for $\mathcal{P}$. By carrying out the complete progress function proof we can check that the informal reasoning is correct.

4.5 Probabilistic MMT Automata

We want to complete our example by showing how to verify the correctness of a progress function. To do so, we need to give a more precise definition of probabilistic MMT automata. The model presented in this section is an extension of the non-probabilistic MMT automaton model of [13].

A *Probabilistic I/O Automaton M* consists of the following five components:

- a set $\mathit{States}(M)$ of states;
- a nonempty set $\mathit{Start}(M) \subseteq \mathit{States}(M)$ of start states;
- an action signature

  $$\mathit{Sig}(M) = (\mathit{in}(M), \mathit{out}(M), \mathit{int}(M)),$$

  where $\mathit{in}(M)$, $\mathit{out}(M)$ and $\mathit{int}(M)$ are three disjoint sets of input, output and internal actions of M.
- a transition relation

  $$\begin{aligned} & \mathit{Trans}(M) \subseteq \\ & \quad \mathit{States}(M) \times \mathit{Actions}(M) \times \mathit{Probs}(\mathit{States}(M)) \end{aligned}$$

  such that for every input action a and every state s there is a transition $(s, a, (\Omega, \mathcal{F}, P))$ in $\mathit{Trans}(M)$.
- a task partition $\mathit{Part}(M)$, which partitions $\mathit{int}(M) \cup \mathit{out}(M)$ into disjoint classes (tasks).

A class $T \in \mathit{Part}(M)$ is said to be *enabled* in a state s if at least one of its actions is enabled in s.

Definition 4.5 Let M be a probabilistic I/O automaton with finitely many tasks. A *boundmap* b for M is a pair of mappings *lower* and *upper*, that give lower and upper bounds for each class of M, such that for all classes T $0 \leq \mathit{lower}(T) < \infty$, $0 < \mathit{upper}(T) \leq \infty$, and $\mathit{lower}(T) \leq \mathit{upper}(T)$. A pair (M, b) is called an *MMT probabilistic automaton*. ■

4.6 Probabilistic Semi-Timed I/O Automata

A *probabilistic semi-timed I/O automaton* is a probabilistic semi-timed automaton M, augmented with an external action signature $\mathit{Esig}(M) = (\mathit{in}(M), \mathit{out}(M))$, which partitions actions of $\mathit{ext}(\mathit{Sig}(M))$ into input and output actions.

$Trans(M)$ is required to have a transition$(s, a, (\Omega, \mathcal{F}, P))$ for every input action a and every state s.

In this section we describe how to incorporate the timing information of a probabilistic MMT automaton (M, b) into the state yielding a probabilistic semi timed I/O automaton $Time(M, b)$ of a special form. This construction is an extension to the probabilistic case of the construction of [10].

Each state s of $Time(M, b)$ consists of the following components:

$basic \in States(M)$, initially a start state of M;

$now \in R^{\geq}$, initially 0;

for each class $T \in Part(M)$,

> $first(T) \in R^{\geq}$, initially $lower(T)$ if some action of T is enabled in $basic$, otherwise 0;
>
> $last(T) \in R^{+} \cup \{\infty\}$ initially $upper(T)$ if some action of T is enabled in $basic$, otherwise ∞.

The transition relation of $Time(M, b)$ is defined as follows: if $a \in Actions(M)$, then a transition $(s, a, (\Omega, \mathcal{F}, P))$ is in $Trans(Time(M, b))$ iff all the following conditions hold

1. $\forall s' \in \Omega\ (s'.now = s.now)$;
2. $(s.basic, a, (\Omega', \mathcal{F}', P'))$ is a transition of M, where $\Omega' = \{s.basic \mid s \in \Omega\}$, and $P'[s.basic] = P[s]$ for all $s \in \Omega$;
3. for each T in $Part(M)$,
   (a) for all $s' \in \Omega$ such that T is not enabled in s', $s'.first(T) = 0$ and $s'.last(T) = \infty$;
   (b) for all $s' \in \Omega$ such that T is enabled in s', if $a \in T$ or T is not enabled in s, then $s'.first = s.now + lower(T)$ and $s'.last(T) = s.now + upper(T)$; if T is enabled in s and $a \notin T$, then $s'.first(T) = s.first(T)$ and $s'.last(T) = s.last(T)$;
4. if a is a time-passage action, say $\nu(t)$, then $(s, \nu(t), s')$ is in $Trans(Time(M, b))$ iff all the following conditions hold:
   (a) $s'.now = s.now + t$;
   (b) $s'.basic = s.basic$;
   (c) for each $T \in Part(M)$,
      i. $s'.now \leq s.last(T)$,
      ii. $s'.first(T) = s.first(T)$, and
      iii. $s'.last(T) = s.last(T)$.

We often omit the $.basic$ part of the selector while referring to components of a state of M. We also write $s.last(a)$, where a is an action of M instead of writing $s.last(T)$, where T is a class of a in $Part(M)$.

Based on the definition of $Time(M, b)$, it is easy to show that the following invariants are satisfied.

Proposition 4.6 *The following hold in any reachable state s of $Time(M, b)$ for any class $T \in Part(M)$:*

- $0 \leq s.now$;
- $s.now \leq s.last(T)$;
- $s.last(T) \leq s.now + upper(T)$;
- $first(T) \leq last(T)$. ■

The invariants that are true for $Time(M, b)$ carry over to some of the generating automata obtained from $Time(M, b)$.

Proposition 4.7 *Let U be a subset of reachable states of M', and let Θ be a set of admissible execution fragments of M' starting in U. Then the following hold in any reachable state q of $G_{M'}(\Theta)$ for any class $T \in Part(M)$:*

- $0 \leq lstate(q).now$;
- $lstate(q).now \leq lstate(q).last(T)$;
- $lstate(q).last(T) \leq lstate(q).now + upper(T)$;
- $lstate(q).first(T) \leq lstate(q).last(T)$. ■

4.7 Checking the Conditions for the Progress Functions

We give an example of verification of a progress function by considering the progress function $(p(s), \mathcal{P}(s))$ of Lemma 4.2 and verifying formally that it satisfies the conditions of Theorem 2.3.

Recall that in the proof of Lemma 4.2 we have chosen Θ_1 to be $first_S(LR, \mathcal{F}_1)$ and we have considered the generating automaton A_1 for Θ_1. We have stated that the time bound statement $Start(A_1) \xrightarrow{l} \mathcal{G}_\uparrow \cup \mathcal{L}_\uparrow$ is true for A_1 and we have defined the following progress function to support the claim:

$$
\begin{array}{rcl}
p(s) & \Leftrightarrow & s \in rstates(A_1) \wedge \neg flipped_{k,k-1}(s) \\
 & & \wedge s.X_k = F \wedge (s.X_{k-1} \in \{\vec{D} \cup F \cup R\}) \\
 & & \wedge s \in \mathcal{RT}_\uparrow \setminus \mathcal{L}_\uparrow \\
\mathcal{P}(s) & = & (min(s.last(flip_k), s.last(flip_{k-1})) - s.now).
\end{array}
$$

We verify that $(p(s), \mathcal{P}(s))$ is a progress function by analyzing the two conditions of Theorem 2.3 separately.

1. Let $s \in Start(A_1)$. We distinguish two cases.
   (a) $p(s) = false$. In this case $s \in \mathcal{G}_\uparrow \cup \mathcal{L}_\uparrow$ trivially.
   (b) $p(s) = true$. In this case $\mathcal{P}(s) \leq s.last(flip_k) - s.now$ by definition, and $s.last(flip_k) - s.now \leq l$ from Proposition 4.7. Thus, $\mathcal{P}(s) \leq l$.
2. Consider a transition (s, a, s') of A_1 such that $p(s) = true$. We distinguish the following cases.
   (a) $a \in \{try_i, flip_i, wait_i, drop_i, exit_i, dropf_i, drops_i, rem_i\}$, where $i \neq k, k-1$.
   Then $p(s')$ is true and $\mathcal{P}(s) = \mathcal{P}(s')$, since a does not modify the state of the processes k and $k-1$.
   (b) $a = second_i$.
   Note that in this case $i \neq k, k-i$. If $s'.X_i = P$ then $s' \in \mathcal{L}_\uparrow$, otherwise $p(s')$ is true and $\mathcal{P}(s) = \mathcal{P}(s')$, since a does not modify the state of the processes k and $k-1$.
   (c) $a \in \{try_{k-1}, drop_{k-1}\}$.
   In this case $p(s')$ is true. Note that $\mathcal{P}(s) = s.last(flip_k) - s.now$, since $s.last(flip_{k-1}) = \infty$. Furthermore, since a does not modify the state of process k and the now component, $\mathcal{P}(s) = s'.last(flip_k) - s'.now$. Therefore, since $\mathcal{P}(s') = \min(s'.last(flip_k), s'.last(flip_{k-1})) - s'.now$, we conclude that $\mathcal{P}(s) \geq \mathcal{P}(s')$.

(d) $a = flip_k$.

In this case $s'.X_k = \overleftarrow{W} \wedge s'.X_{k-1} \in \{\overrightarrow{D} \cup F \cup R\} \wedge s' \in \mathcal{RT}_\uparrow \setminus \mathcal{L}_\uparrow$, which implies $s' \in \mathcal{G}_\uparrow$.

(e) $a = flip_{k-1}$.

In this case $s'.X_{k-1} = \overrightarrow{W} \wedge s'.X_k = F$, which implies $s' \in \mathcal{G}_\uparrow$.

(f) $a \in \{try_k,\ wait_k,\ drop_k,\ exit_k,\ dropf_k,\ drops_k,\ rem_k\}$.

This case is not possible since if $p(s)$ is true then $s.X_k = F$.

(g) $a = \nu(\Delta t)$.

In this case $p(s')$ is true. By substituting the definition of $\mathcal{P}(s)$ in $\mathcal{P}(s) - \Delta t$ we ontain $\mathcal{P}(s) - \Delta t = \min(s.last(flip_k), s.last(flip_{k-1})) - (s.now + \Delta t)$. Observe that $s'.last(flip_k) = s.last(flip_k)$, that $s'.last(flip_{k-1}) = s.last(flip_{k-1})$, and that $s.now + \Delta t = s', now$. Therefore,
$\mathcal{P}(s) - \Delta t = \mathcal{P}(s')$.

The proof above consists of a very detailed and structured cases analysis, where the effect of each action is considered. Although the analysis is long and tedious, the cases structure and the reasoning for each case are simple and can be automated using a theorem prover.

5 Conclusion

In this paper we have improved the results of [11] by introducing a stylized way of proving probabilistic time bound statements. This allows us to eliminate the operational reasoning from correctness proofs, and therefore, it allows us to produce proofs that are formal and easy to verify.

Our proof method separates the key insights of a correctness proof from the routine parts, and makes it possible to structure a proof in several levels of detail. This allows a designer to work out the specific details of a correctness proof only where it is desired. Moreover, the routine parts of the proof, which are the most tedious ones, are the parts that can be verified automatically.

The method can be used to verify time bounded progress of several randomized algorithms. We have carried out and verified automatically the complete proof for the algorithm of Lehmann and Rabin, and we have done some work in applying the method to prove the correctness of Ben Or's randomized agreement protocol [4]. We believe that the method can be applied also to the randomized self-stabilizing spanning tree algorithm by Sudhanshu Aggarwal and Shay Kutten [2]. In [1] the algorithm is verified using time bound statements and inference rules, but the individual basic time bound statements are justified by long operational arguments, which can be eliminated using our method.

Further work involves the analysis of other existing algorithms so that the full power of our method can be tested. At the moment we do not know its limitations, and it is likely that new features will be added to the method in the future. However, we think that the main idea of reducing the analysis of a probabilistic property to the analysis of a non-probabilistic property is always applicable.

Acknowledgments We thank Nancy Lynch for helpful comments and suggestions.

References

[1] S. Aggarwal. Time optimal self-stabilizing spanning tree algorithms. Technical Report MIT/LCS/TR-632, MIT Laboratory for Computer Science, 1994.

[2] S. Aggarwal and S. Kutten. Time optimal self stabilizing spanning tree algorithms. In R.K. Shyamasundar, editor, *13th International Conference on Foundations of Software Technology and Theoretical Computer Science*, LNCS 761, 1993

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

[4] M. Ben-Or. Another advantage of free choice: completely asynchronous agreement protocols. In *Proceedings of the 2^{nd} Annual ACM PODC*, 1983

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

[6] 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.

[7] S.J. Garland and J.V. Guttag. A guide to LP, the Larch Prover. Technical Report 82, DEC Systems Research Center, December 1991.

[8] H. Hansson. *Time and Probability in Formal Design of Distributed Systems*, volume 1 of *Real-Time Safety Critical Systems.* Elsevier, 1994.

[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 8^{th} Annual ACM PODC*, pages 133–138, January 1981.

[10] N.A. Lynch and H. Attiya. Using mappings to prove timing properties. *Distributed Computing*, 6:121–139, 1992.

[11] N.A. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *Proceedings of the 13^{th} Annual ACM PODC*, 1994.

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

[13] M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings of CONCUR 91*, LNCS 527, 1991.

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

[15] A. Pogosyants and R. Segala. Automatic verification of time properties of randomized distributed algorithms. In progress, 1995.

[16] J.R. Rao. Reasoning about probabilistic algorithms. In *Proceedings of the 9^{th} Annual ACM PODC*, 1990.

[17] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, MIT, Dept. of Electrical Eng. and Computer Science, 1995.